Estratégia Concursos

Aula 00

****NÃO ATIVAR***Desenvolvimento p/ PEFOCE (Perito- Análise de Sistemas) - 2021 - Pré-Edital*

Autor:
Equipe Informática e TI, Pedro Henrique Chagas Freitas

01 de Fevereiro de 2021

Sumário

1 ESTRUTURAS DE DADOS

1.1 INTRODUÇÃO

Pessoal, um programa pode ser visto como uma especificação formal da solução de um problema. Wirth expressa esse conceito por meio de uma equação:

PROGRAMA = ALGORITMO + ESTRUTURA DE DADOS

Nosso foco aqui é em Estruturas de Dados! Na evolução do mundo computacional, um fator extremamente importante trata da forma de armazenar informações. De nada adianta o enorme desenvolvimento de hardware e software se a forma de armazenamento e tratamento de dados não evoluir harmonicamente. E é por isso que as estruturas de dados são tão fundamentais.

As estruturas de dados, na maioria dos casos, baseiam-se nos tipos de armazenamento vistos dia a dia, i.e., nada mais são do que a transformação de uma forma de armazenamento já conhecida e utilizada no mundo real adaptada para o mundo computacional. Por isso, cada tipo de estrutura de dados possui vantagens e desvantagens e cada uma tem sua área de atuação otimizada.

Bem, não vou enrolar muito explicando o que é uma Estrutura de Dados! A melhor forma de saber é vendo exemplos. Antes disso, eu gostaria de falar sobre um conceito importante: Dados Homogêneos e Heterogêneos. Os primeiros são aqueles que possuem só um tipo básico de dados (Ex: Inteiros); os segundos são aqueles que possuem mais de um tipo básico de dados (Ex: Inteiros + Caracteres). Os tipos básicos de dados também são chamados de tipos primitivos.

Entenderam? Existem estruturas de dados que tratam de dados homogêneos, i.e., todos os dados são apenas de um tipo básico, tais como Vetores! Ora, em um vetor, todos os elementos são do mesmo tipo. Existem estruturas de dados que tratam de dados heterogêneos, i.e., os dados são de tipos básicos diferentes, tais como Listas! Ora, em uma lista, todos os elementos são, em geral, de tipos básicos diferentes.

Além dessa classificação, existe outra também importante: Estruturas Lineares e Estruturas Não-Lineares. As Estruturas Lineares são aquelas em que cada elemento pode ter um único predecessor (exceto o primeiro elemento) e um único sucessor

(exceto o último elemento). Como exemplo, podemos citar Listas, Pilhas, Filas, Arranjos, entre outros.

Já as Estruturas Não-Lineares são aquelas em que cada elemento pode ter mais de um predecessor e/ou mais de um sucessor. Como exemplo, podemos citar Árvores, Grafos e Tabelas de Dispersão. Essa é uma classificação muito importante e muito simples de entender. Pessoal, vocês perceberão que esse assunto é cobrado de maneira superficial na maioria das questões, mas algumas são nível doutorado!

Por fim, vamos falar sobre Tipos Abstratos de Dados (TAD). Podemos defini-lo como um modelo matemático (v,o), em que v é um conjunto de valores e o é um conjunto de operações que podem ser realizadas sobre valores. Eu sei, essa definição é horrível de entender! Para compreender esse conceito, basta lembrar de abstração, i.e., apresentar interfaces e esconder detalhes.

Os Tipos Abstratos de Dados são simplesmente um modelo para um certo tipo de estrutura de dados. *Como assim, professor?* Quando eu falo em pilha, eu estou falando de um tipo abstrato de dados que tem duas operações com comportamentos bem definidos e conhecidos: *push* (para inserir elementos na pilha); e *pop* (para retirar elementos da pilha).

E a implementação dessas operações? Isso não importa! Aliás, não importa implementação nem paradigma nem linguagem de programação. Não importa se a pilha é implementada com um paradigma orientado a objetos ou com um paradigma estruturado; não importa se a pilha é implementada em Java ou Pascal; não importa como é a implementação interna – isso serve para outras estruturas[1].

Podemos concluir, portanto, que um tipo abstrato de dados contém um modelo que contém valores e operações associadas, de forma que essas operações sejam precisamente independentes de uma implementação particular. Em geral, um TAD é especificado por meio de uma especificação algébrica que, em geral, contém três partes: Especificação Sintática, Semântica e de Restrições.

A Especificação Sintática define o nome do tipo, suas operações e o tipo dos argumentos das operações, definindo a assinatura do TAD. A Especificação Semântica descreve propriedades e efeitos das operações de forma independente de uma implementação específica. E a Especificação de Restrições estabelece as condições que devem ser satisfeitas antes e depois da aplicação das operações.

[1] Filas, Pilhas, Árvores, Deques, entre outros.

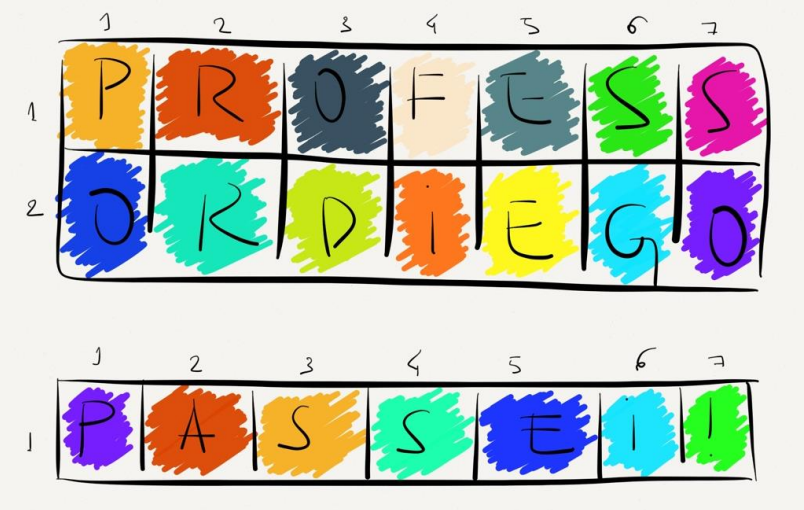
Em outras palavras, o nível semântico trata do comportamento de um tipo abstrato de dados; e o nível sintático trata da apresentação de um tipo abstrato de dados. Podemos dizer, então, que o TAD encapsula uma estrutura de dados com características semelhantes – podendo ser formado por outros TADs –, e esconde a efetiva implementação dessa estrutura de quem a manipula.

1.2 ESTRUTURAS DE DADOS: VETORES E MATRIZES

Um Vetor é uma estrutura de dados linear que necessita de somente um índice para que seus elementos sejam indexados. É uma estrutura homogênea, portanto armazena somente uma lista de valores do mesmo tipo. Ele pode ser estático ou dinâmico, com dados armazenados em posições contíguas de memória e permite o acesso direto ou aleatório a seus elementos.

Observem que, diferentemente das listas, filas e pilhas, ele vem praticamente embutido em qualquer linguagem de programação. *E a Matriz, professor?* Não muda muita coisa! Trata-se de um arranjo bidimensional ou multidimensional de alocação estática e sequencial. Ela necessita de um índice para referenciar a linha e outro para referenciar a coluna para que seus elementos sejam endereçados.

Da mesma forma que um vetor, uma matriz também pode ter tamanhos variados, todos os elementos são do mesmo tipo, cada célula contém somente um valor e os tamanhos dos valores são os mesmos. Os elementos ocupam posições contíguas na memória. A alocação dos elementos pode ser feita colocando os elementos linha-por-linha ou coluna-por-coluna.



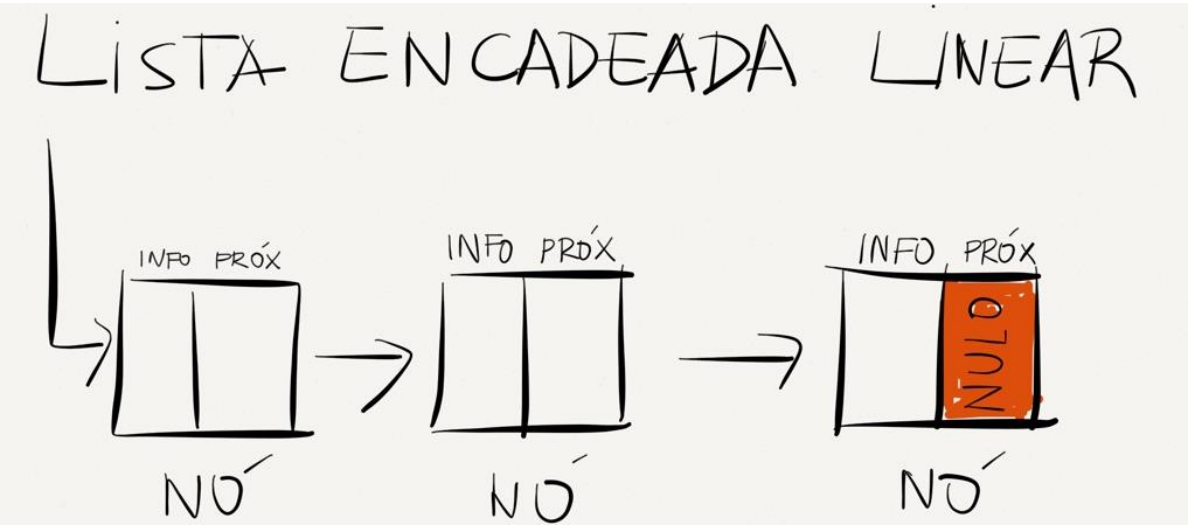
MATRIZ 2X7 E VETOR (7 POSIÇÕES)

1.3 ESTRUTURA DE DADOS: LISTA ENCADEADA

Também conhecida como *Lista Encadeada Linear*, *Lista Ligada Linear* ou *Linked List*, trata-se de uma estrutura de dados dinâmica formada por uma sequência encadeada de elementos chamados nós, que contêm dois campos: campo de informação e campo de endereço. O primeiro armazena o real elemento da lista e o segundo contém o endereço do próximo nó da lista.

Esse endereço, que é usado para acessar determinado nó, é conhecido como ponteiro. A lista ligada inteira é acessada a partir de um ponteiro externo que aponta para o primeiro nó na lista, i.e., contém o endereço do primeiro nó[2]. Por ponteiro "externo", entendemos aquele que não está incluído dentro de um nó. Em vez disso, seu valor pode ser acessado diretamente, por referência a uma variável.

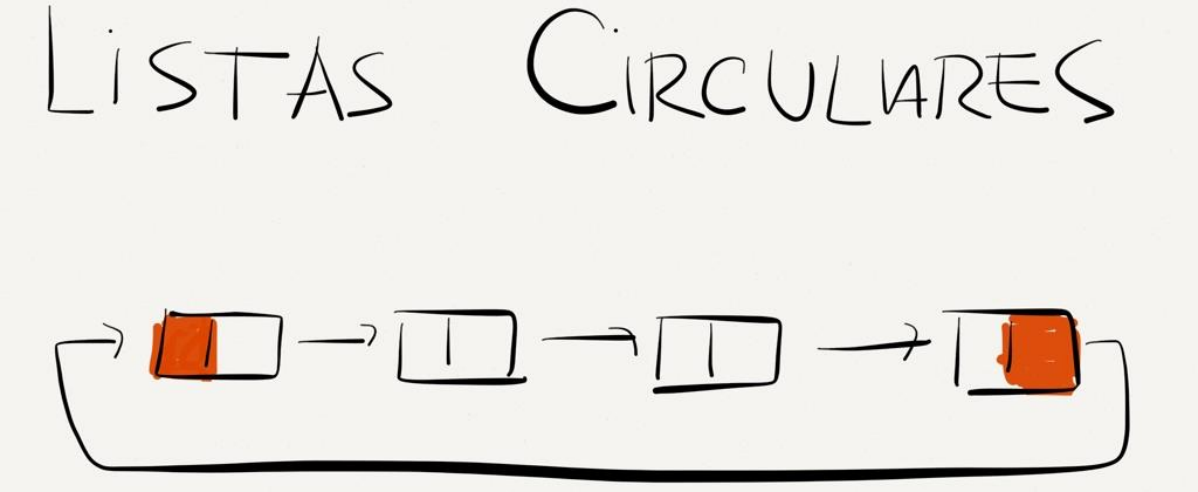
O campo do próximo endereço do último nó na lista contém um valor especial, conhecido como NULL, que não é um endereço válido. Esse ponteiro nulo é usado para indicar o final de uma lista. Uma lista é chamada Lista Vazia ou Lista Nula caso não tenha nós ou tenha apenas um nó sentinela. O valor do ponteiro externo para esta lista é o ponteiro nulo. Uma lista pode ser inicializada com uma lista vazia.



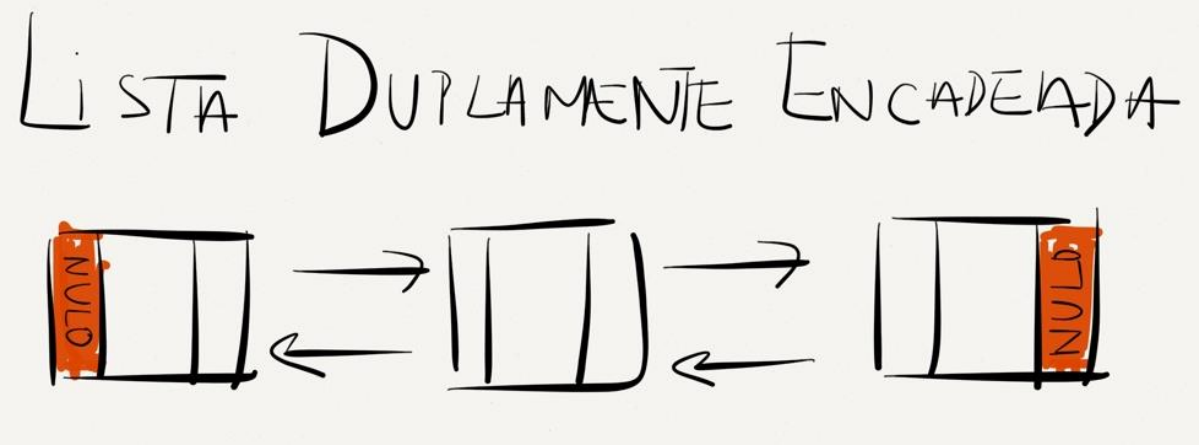
Suponha que seja feita uma mudança na estrutura de uma lista linear, de modo que o campo *próximo* no último nó contenha um ponteiro de volta para o primeiro nó,

[2] O endereço do primeiro nó pode ser encapsulado para facilitar possíveis futuras operações sobre a lista sem a necessidade de se conhecer sua estrutura interna. O primeiro elemento e o último nós são muitas vezes chamados de *Sentinela*.

em vez de um ponteiro nulo. Esse tipo de lista é chamado Lista Circular (ou Fechada[3]), i.e., a partir de qualquer ponto, é possível atingir qualquer outro ponto da lista. *Certinho, até agora?*



Observe que uma Lista Circular não tem um primeiro ou último nó natural. Precisamos, portanto, estabelecer um primeiro e um último nó por convenção. Uma convenção útil é permitir que o ponteiro externo para a lista circular aponte para o último nó, e que o nó seguinte se torne o primeiro nó. Assim podemos incluir ou remover um elemento convenientemente a partir do início ou do final de uma lista.



Embora uma lista circularmente ligada tenha vantagens sobre uma lista linear, ela ainda apresenta várias deficiências. Não se pode atravessar uma lista desse tipo no sentido contrário nem um nó pode ser eliminado de uma lista circularmente ligada

[3] Se Listas Circulares são conhecidas como Listas Fechadas, as Listas Abertas são todas aquelas que são Não-Circulares. Por fim: da mesma forma que há Listas Circulares Simples, há também Listas Circulares Duplas. Nesse caso, o ponteiro anterior do primeiro elemento aponta para o último elemento e o ponteiro posterior do último elemento aponta para o primeiro elemento.

sem se ter um ponteiro para o nó antecessor. Nos casos em que tais recursos são necessários, a estrutura de dados adequada é uma lista duplamente ligada.

Cada nó numa lista desse tipo contém dois ponteiros, um para seu predecessor e outro para seu sucessor. Na realidade, no contexto de listas duplamente ligadas, os termos *predecessor* e *sucessor* não fazem sentido porque a lista é totalmente simétrica. As listas duplamente ligadas podem ser lineares ou circulares e podem conter ou não um nó de cabeçalho.

Podemos considerar os nós numa lista duplamente ligada como consistindo em três campos: um campo *info* que contém as informações armazenadas no nó, e os campos *left* e *right*, que contêm ponteiros para os nós em ambos os lados. Dado um ponteiro para um elemento, pode-se acessar os elementos adjacentes e, dado um ponteiro para o último elemento, pode-se percorrer a lista em ordem inversa.

Existem cinco operações básicas sobre uma lista encadeada: Criação, em que se cria a lista na memória; Busca, em que se pesquisa nós na lista; Inclusão, em que se insere novos nós na lista em uma determinada posição; Remoção, em que se elimina um elemento da lista; e, por fim, Destruição, em que se destrói a lista junto com todos os seus nós.

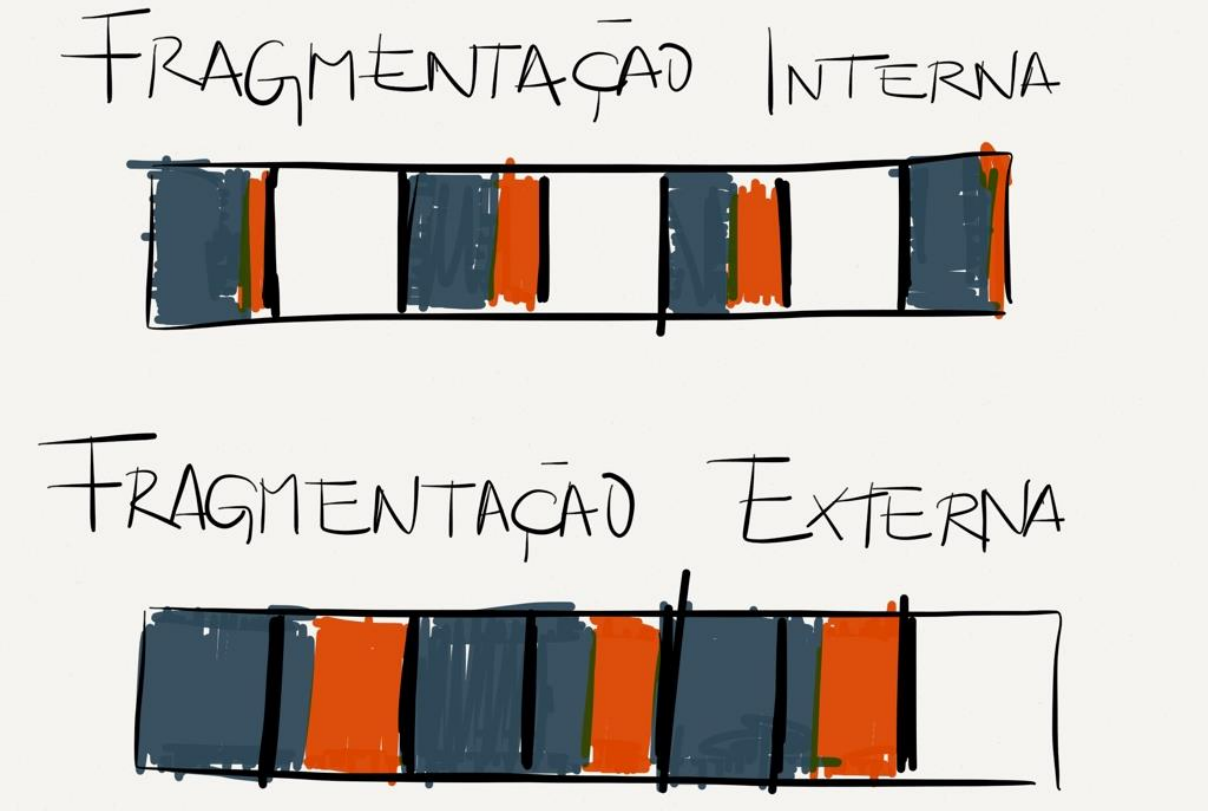
IMPORTANTE

Pilhas e Filas são subespécies de Listas. No entanto, cuidado na hora de responder questões! De maneira genérica, Pilhas e Filas podem ser implementadas como Listas. No entanto, elas possuem características particulares de uma lista genérica. Ok?

Precisamos falar um pouco sobre Fragmentação! *O que é isso, professor?* Galera, falou em fragmentação, lembrem-se de desperdício de espaço disponível de memória. O fenômeno no qual existem vários blocos disponíveis pequenos e não-contíguos é chamado fragmentação externa porque o espaço disponível é desperdiçado fora dos blocos alocados.

Esse fenômeno é o oposto da fragmentação interna, no qual o espaço disponível é desperdiçado dentro dos blocos alocados, como apresenta a imagem abaixo. Sistemas Operacionais possuem uma estrutura de dados que armazena informações sobre áreas ou blocos livres (geralmente uma lista ou tabela). Uma lista encadeada elimina o problema da fragmentação externa. *Por que?*

Porque mantém os arquivos, cada um, como uma lista encadeada de blocos de disco. Dessa forma, uma parte de cada bloco é usada como ponteiro para o próximo bloco e o restante do bloco é usado para dados. Uma vantagem desse tipo de alocação é que o tamanho do arquivo não precisa ser conhecido antes de sua criação, já que cada bloco terá um ponteiro para o próximo bloco.



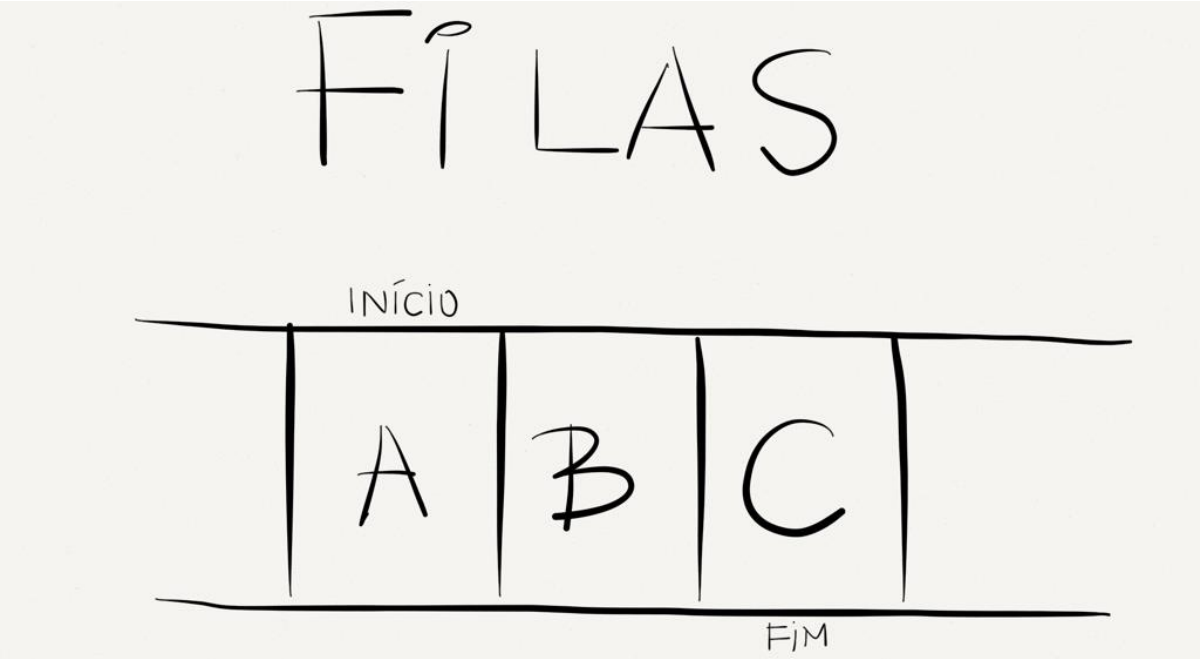
Galera... e o acesso a uma lista? A Lista é uma estrutura de acesso sequencial, i.e., é preciso percorrer nó por nó para acessar um dado específico. Logo, é proporcional ao número de elementos – Acesso O(n). *E os Vetores?* Eles são uma estrutura de acesso direto, i.e., pode-se acessar um elemento diretamente. Portanto, não precisa percorrer elemento por elemento (Acesso O(1))[4].

1.4 ESTRUTURAS DE DADOS: FILAS

Uma fila é um conjunto ordenado de itens a partir do qual podem-se eliminar itens numa extremidade (chamada início da fila) e no qual podem-se inserir itens na outra extremidade (chamada final da fila). Também conhecida como Lista FIFO (First In

[4] No Acesso Sequencial: quanto mais ao fim, maior o tempo para acessar; no Acesso Direto: todos os elementos são acessados no mesmo tempo.

First Out), basta lembrar de uma fila de pessoas esperando para serem atendidas em um banco, i.e., o primeiro a entrar é o primeiro a sair.



Quando um elemento é colocado na fila, ele ocupa seu lugar no fim da fila, como um aluno recém-chegado que ocupa o final da fileira. O elemento retirado da fila é sempre aquele que está no início da fila, como o aluno que se encontra no começo da fileira e que esperou mais tempo. As operações básicas são Enqueue (Enfileirar) e Dequeue (Desenfileirar). As Filas possuem início (ou cabeça) e fim (ou cauda).

É bom salientar outro conceito importante: Deque (Double Ended Queue)! É também conhecida como Filas Duplamente Encadeadas e permite a eliminação e inserção de itens em ambas as extremidades. Ademais, elas permitem algum tipo de priorização, visto que é possível inserir elementos de ambos os lados. Assim sendo, é comum em sistemas distribuídos!

Sistemas distribuídos sempre necessitam que algum tipo de processamento seja mais rápido, por ser mais prioritário naquele momento, deixando outros tipos mais lentos ou em fila de espera, por não requerem tanta pressa. Ele pode ser entendido como uma extensão da estrutura de dados Fila. Agora uma pergunta: *Qual a diferença entre uma lista duplamente encadeada e um deque?*

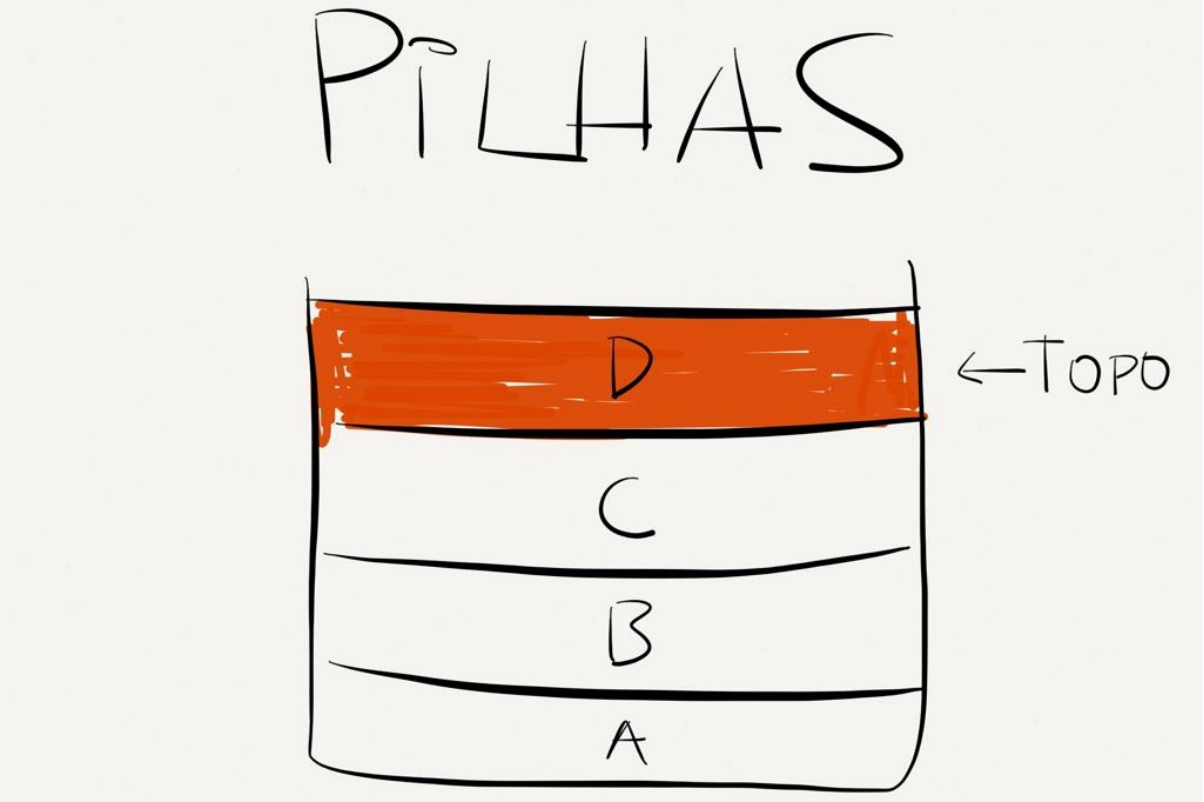
Pessoal, um deque gerencia elementos como um vetor, fornece acesso aleatório e tem quase a mesma interface que um vetor. Ele se diferencia de uma lista duplamente encadeada, entre outras coisas, por essa não fornecer acesso aleatório

aos elementos, i.e., para acessar o quinto elemento, você deve navegar pelos quatro primeiros elementos – logo a lista é mais lenta nesse sentido. *Bacana?*

1.5 ESTRUTURAS DE DADOS: PILHAS

A Pilha é um conjunto ordenado de itens no qual novos itens podem ser inseridos e eliminados em uma extremidade chamada *topo*. Novos itens podem ser colocados no topo da pilha (tornando-se o novo primeiro elemento) ou os itens que estiverem no topo da pilha poderão ser removidos (tornando-se o elemento mais abaixo o novo primeiro elemento).

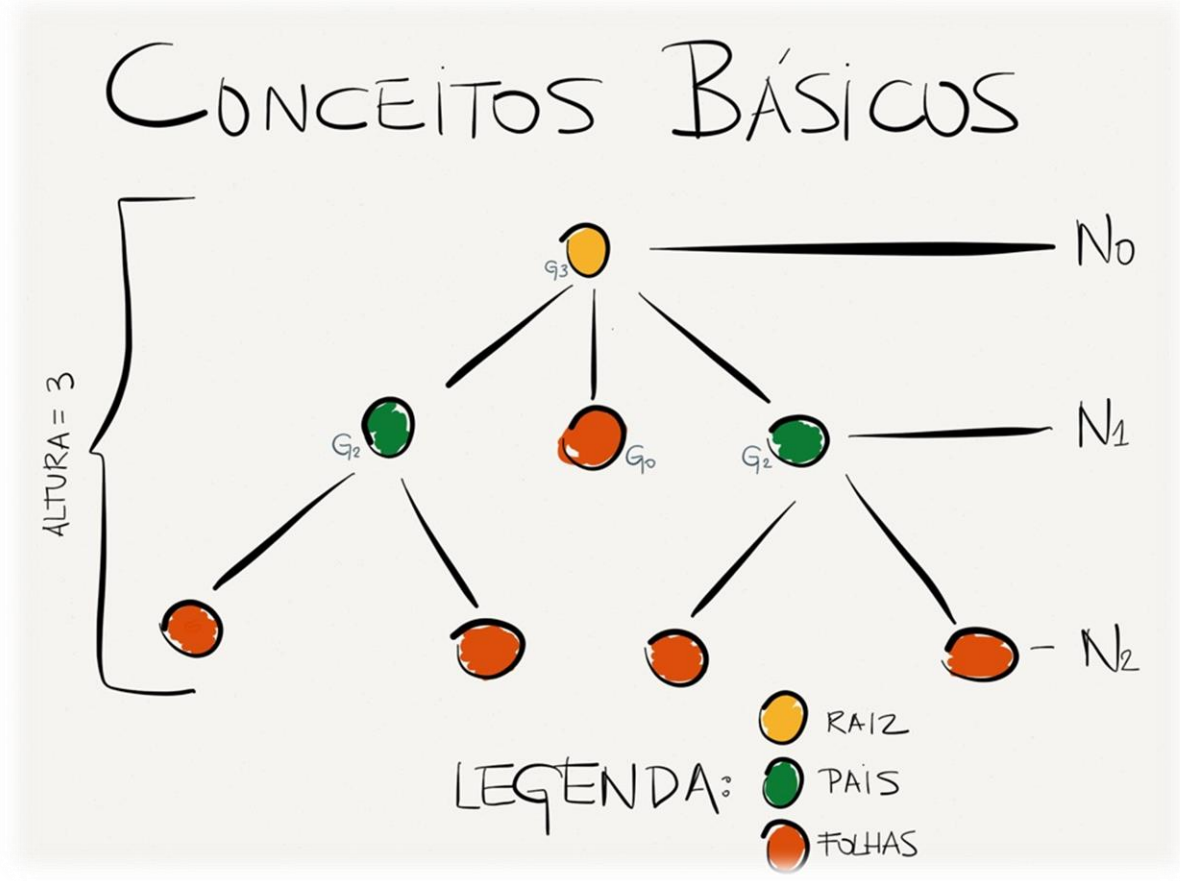
Também conhecida como Lista LIFO (Last In First Out), basta lembrar de uma pilha de pratos esperando para serem lavados, i.e., o último a entrar é o primeiro a sair. A ordem em que os pratos são retirados da pilha é o oposto da ordem em que eles são colocados sobre a pilha e, como consequência, apenas o prato do topo da pilha está acessível.



As Pilhas oferecem três operações básicas: *push*, que insere um novo elemento no topo da pilha; *pop*, que remove um elemento do topo da pilha; e *top* (também conhecida como *check*), que acessa e consulta o elemento do topo da pilha. Pilhas podem ser implementadas por meio de Vetores (Pilha Sequencial - Alocação Estática de Memória) ou Listas (Pilha Encadeada - Alocação Dinâmica de Memória).

1.6 ÁRVORES

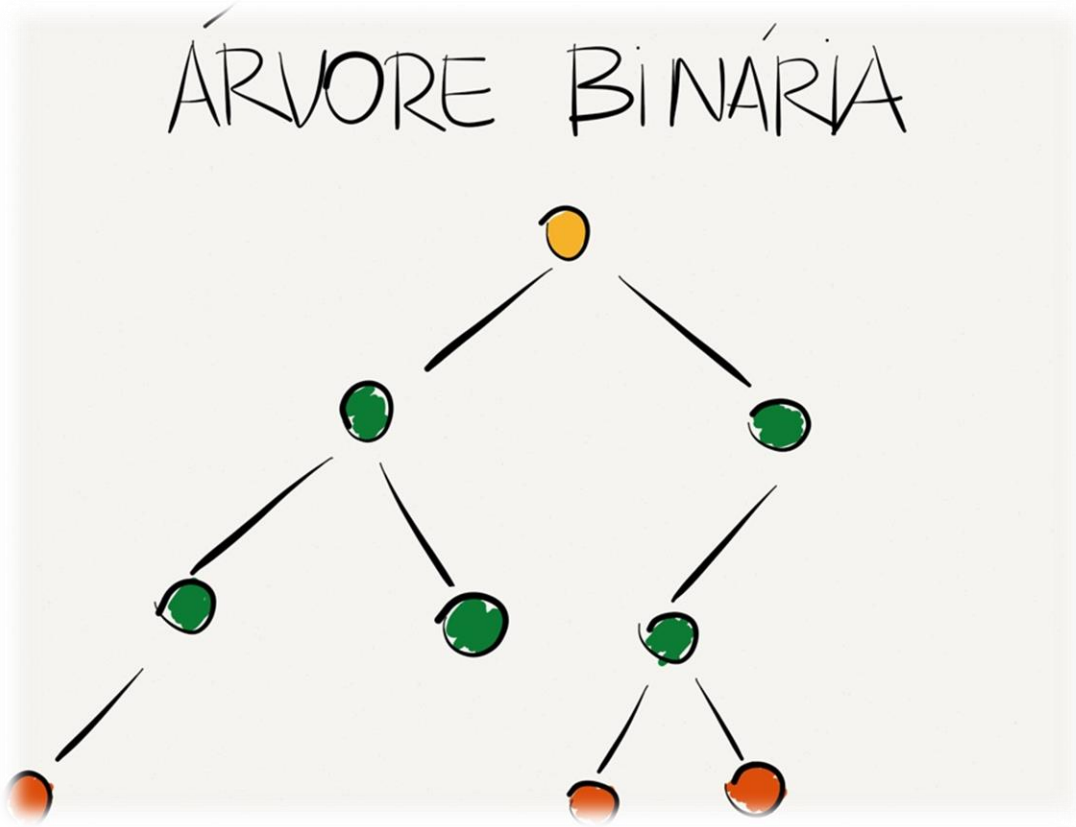
Uma árvore é uma estrutura de dados hierárquica (não-linear) composta por um conjunto finito de elementos com um único elemento raiz, com zero ou mais sub-árvores ligadas a esse elemento raiz. Como mostra a imagem abaixo, há uma única raiz, em amarelo. Há também nós folhas, em vermelho e seus pais, em verde. Observem ainda os conceitos de Altura, Grau e Nível de uma árvore.

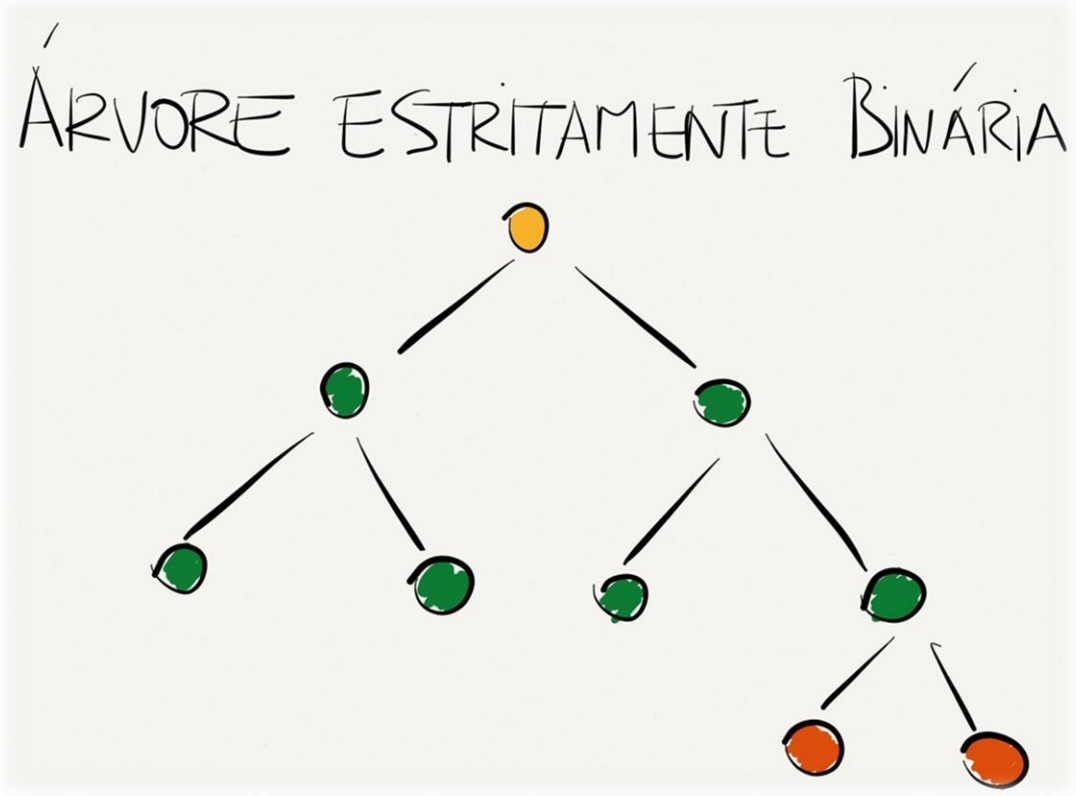


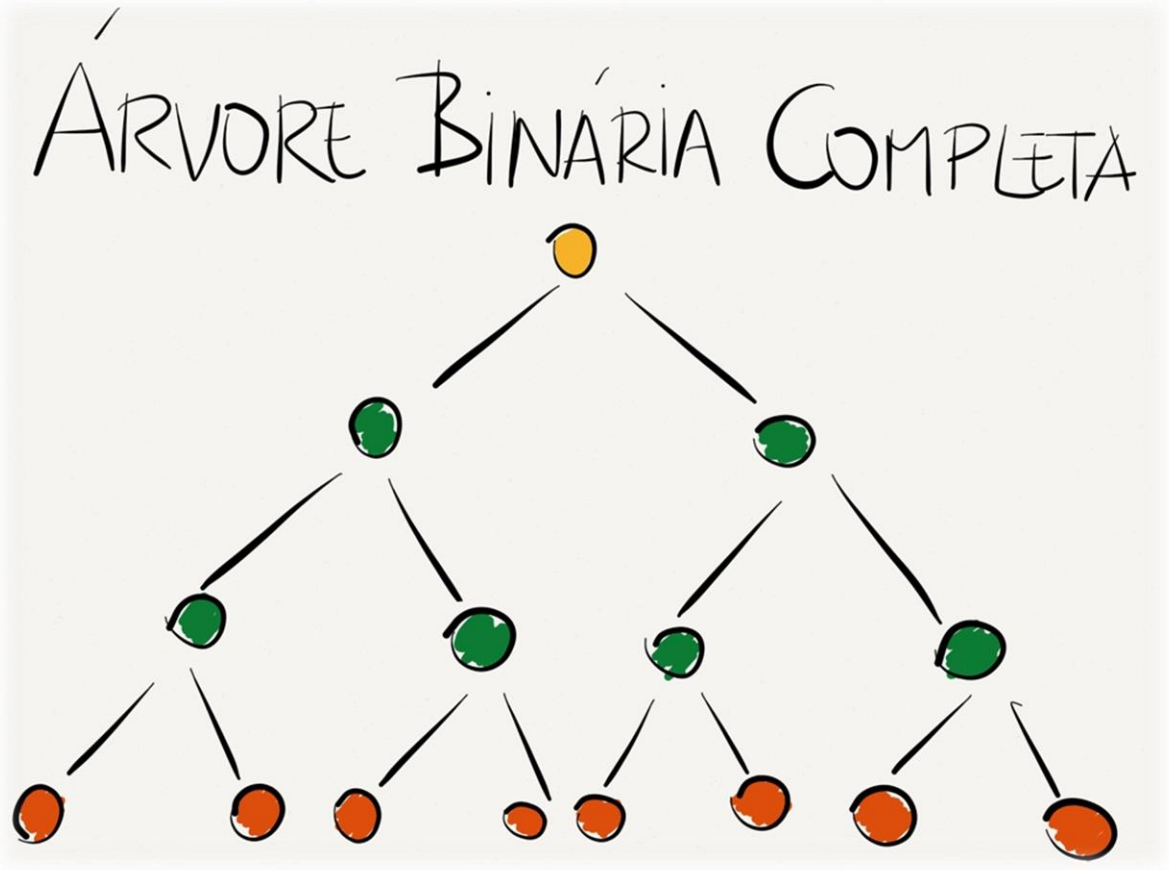
O Grau informa a quantidade de filhos de um determinado nó! A Raiz tem Nível 0 (excepcionalmente, alguns autores consideram que tem Nível 1) e o nível de qualquer outro nó na árvore é um nível a mais que o nível de seu pai. Por fim, a

Altura é a distância entre a raiz e seu descendente mais afastado. Dessas informações, podemos concluir que toda folha tem Grau 0.

Existe um tipo particular de árvore chamado: Árvore Binária! *O que é isso?* É uma estrutura de dados hierárquica em que todos os nós têm grau 0, 1 ou 2. Já uma Árvore Estritamente Binária é aquela em que todos os nós têm grau 0 ou 2. E uma Árvore Binária Completa é aquela em que todas as folhas estão no mesmo nível, como mostram as imagens abaixo.




ÁRVORE ESTRITAMENTE BINÁRIA


ÁRVORE BINÁRIA COMPLETA

Uma Árvore Binária Completa com *x* folhas conterá sempre (2x – 1) nós. Observem a imagem acima e façam as contas: 2*8 – 1 = 15 nós! Uma árvore binária completa de altura *h* e nível *n* contém (2^h-1) ou (2^{n+1}-1) nós e usa-se (2^n), para calcular a quantidade de nós em determinado nível. Na imagem acima, há uma árvore de h = 4 e n = 3; logo, existem 2^{3+1} -1 = 15 nós no total; e no Nível 3, existe 2^3 = 8 nós.

Vamos falar um pouco agora sobre Árvore de Busca Binária! Trata-se de uma Uma estrutura de dados vinculada, baseada em nós, onde cada nó contém uma chave e duas subárvores à esquerda e a direita. Para todos nós, a chave da subárvore esquerda deve ser menor que a chave desse nó, e a chave da subárvore direita deve ser maior. *Beleza?*

Todas estas subárvores devem qualificar-se como árvores binárias de busca. O pior caso de tempo de complexidade para a pesquisa em uma árvore binária de busca é a altura da árvore, isso pode ser tão pequeno como O(log n) para uma árvore com n elementos. Galera, abaixo nós vamos ver como árvores podem ser representadas e o conceito de árvore binária de busca ficará mais clara!

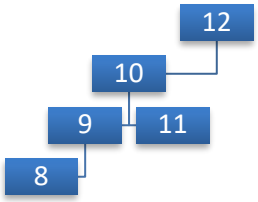
Como representamos árvores? Podemos representar uma árvore como um conjunto de parênteses aninhados. Nessa notação, (P (F_1)(F_2)) significa que P, F_1, F_2 são nós e que F_1, F_2, são filhos do pai P. Ao transcrever isso para o desenho hierárquico de uma árvore, lemos da esquerda para a direita. Agora suponhamos que F_1 tem dois filhos N_1 e N_2. Logo, reescrevemos a subárvore de F_1 como (F_1 (N_1)(N_2)).

Podemos, então, substituir F_1 por (F_1(N_1)(N_2)). Ao final, ficará (P(F_1(N_1)(N_2))(F_2)). E assim por diante. Temos então que o primeiro elemento é a raiz e sempre que tivermos um parêntese, teremos uma nova subárvore. Vamos exemplificar: **(12(10(9(8))(11))(14(13)(15)))**. *Como ficaria, professor?* Sabemos que 12 será a raiz dessa árvore e, a partir daí, criamos a árvore da esquerda para direita.

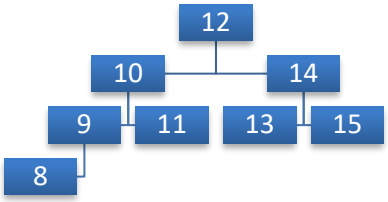
Observem o parêntese após o 12! Notem que ele só é fechado após o 11: **(12(10(9(8))(11))**. Isso significa que tudo que está em vermelho é subárvore da esquerda da raiz 12 – e o restante **(14(13)(15))** é subárvore da direita da raiz 12. Observem o parêntese após o 10! Notem que ele só é fechado após o 8: **10(9(8))**. Isso significa que tudo que está em amarelo é subárvore da esquerda da raiz 10.

E o restante **(11)** é subárvore da direita da raiz 10. Observem o parêntese após o 9! Notem que ele só é fechado após o 8: **9(8)**. Isso significa que tudo que está em verde é subárvore da esquerda da raiz 9 – e o restante... *que restante, professor?*

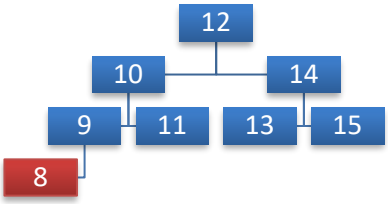
Pois é, não tem restante! Logo, 9 não tem subárvore da direita. Vejam abaixo como ficou e vamos analisar o outro lado.

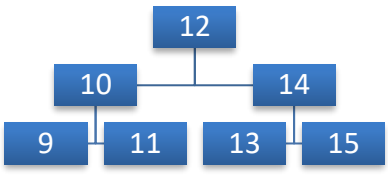


A subárvore da direita da raiz 12 tem raiz 14 e tem dois filhos: na esquerda, 13 e na direita 15. Fim! Galera, eu sei que parece complicado, mas leiam e pratiquem umas três vezes – de preferência no papel - que vocês internalizam tranquilamente esse conteúdo. É chatinho, mas não é difícil! Vejam abaixo como ficou o resultado final e vamos seguir em frente...

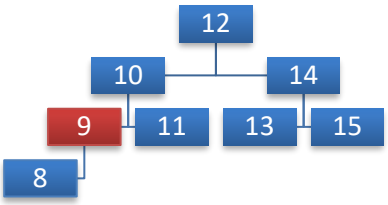


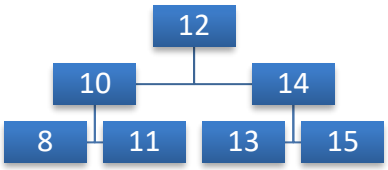
Bem, pessoal! Dito isso, vamos analisar agora como ficaria para remover um nó desta árvore. Existem três possibilidades para realizar essa operação: (1) remover um nó que não tem filhos; (2) remover um nó que tem apenas um filho; (3) e remover um nó que tenha dois filhos. O primeiro caso é muito simples: basta retirar o nó desejado e ponto final. Vejamos como fica:





No segundo caso, basta retirar o nó da árvore e conectar seu único filho (e sua subárvore, se houver) diretamente ao pai do nó removido. Vejamos:

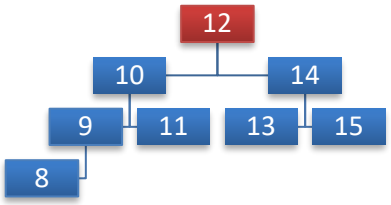


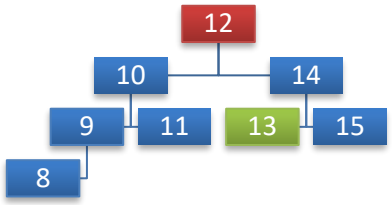


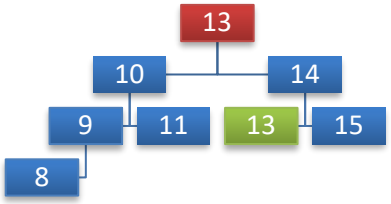
Já o último caso, nós podemos utilizar duas estratégias. Você pode escolher qual deseja utilizar em uma situação específica. Vejamos:

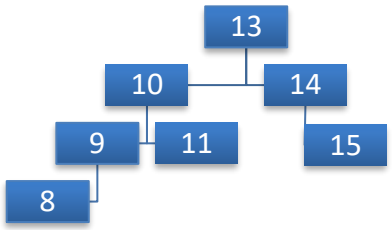
ESTRATÉGIA 1

PASSO 1: IDENTIFIQUE O ELEMENTO QUE VOCÊ DESEJA RETIRAR DA ÁRVORE (EM VERMELHO)
PASSO 2: IDENTIFIQUE O MENOR ELEMENTO DE TODA SUBÁRVORE À DIREITA DO NÓ IDENTIFICADO NO PASSO 1 (EM VERDE)
PASSO 3: COPIE O VALOR DO NÓ IDENTIFICADO NO PASSO 2 PARA O NÓ IDENTIFICADO NO PASSO 1
PASSO 4: REMOVA O ELEMENTO IDENTIFICADO NO PASSO 2.



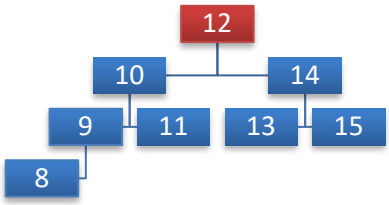


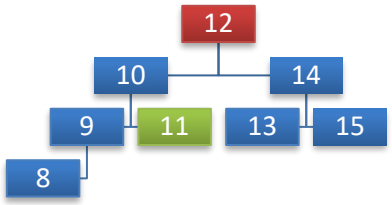


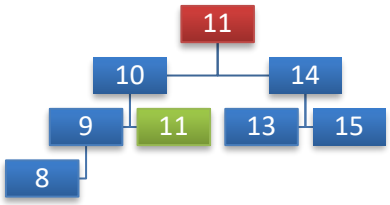


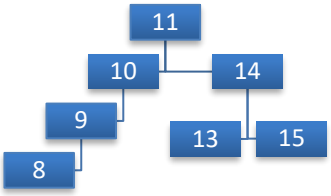
ESTRATÉGIA 2

PASSO 1: IDENTIFIQUE O ELEMENTO QUE VOCÊ DESEJA RETIRAR DA ÁRVORE (EM VERMELHO)
PASSO 2: IDENTIFIQUE O MAIOR ELEMENTO DE TODA SUBÁRVORE À ESQUERDA DO NÓ IDENTIFICADO NO PASSO 1 (EM VERDE)
PASSO 3: COPIE O VALOR DO NÓ IDENTIFICADO NO PASSO 2 PARA O NÓ IDENTIFICADO NO PASSO 1
PASSO 4: REMOVA O ELEMENTO IDENTIFICADO NO PASSO 2.









Por fim, como seria a representação por parênteses aninhados? Na primeira estratégia, temos: (13(10(9(8))(11))(14(15))); na segunda, temos (11(10(9(8)))(14(13)(15))).

Galera, em uma Árvore de Busca Binária, podemos fazer três percursos: pré-ordem, in-ordem e pós-ordem (esses prefixos são em relação a raiz). É interessante notar que, quando se faz um percurso em ordem em uma árvore binária de busca, os valores dos nós aparecem em ordem crescente. A operação "Percorre" tem como objetivo percorrer a árvore numa dada ordem, enumerando os seus nós.

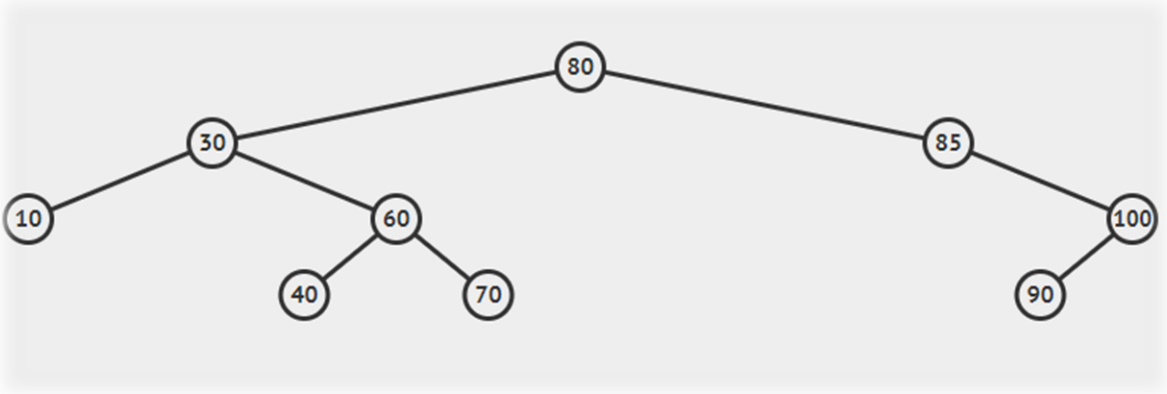
Quando um nó é enumerado, diz-se que ele foi "visitado". Vamos ver agora esses três percursos:

- Pré-Ordem (ou Profundidade): visita a raiz; percorre a subárvore esquerda em pré-ordem; percorre a subárvore direita em pré-ordem.

- In-Ordem (ou Simétrica): percorre a subárvore esquerda em in-ordem; visita a raiz; percorre a subárvore direita em in-ordem.

- Pós-Ordem: percorre a subárvore esquerda em pós-ordem; percorre a subárvore direita em pós-ordem; visita a raiz.

Vamos ver um exemplo:



Como ler essa árvore em Pré-Ordem? Vamos tentar...

//Percorrendo a Árvore em Pré-Ordem:

1. Visite a Raiz: {**80**};
2. Percorra a subárvore esquerda em pré-ordem:
    1. Visite a Raiz: {**30**};
    2. Percorra a subárvore esquerda em pré-ordem:
        1. Visite a Raiz: {**10**};
        2. Percorra a subárvore esquerda em pré-ordem: {Vazio}
        3. Percorra a subárvore direita em pré-ordem: {Vazio}
    3. Percorra a subárvore direita em pré-ordem:
        1. Visite a Raiz: {**60**};
        2. Percorra a subárvore esquerda em pré-ordem:
            1. Visite a Raiz: {**40**}
            2. Percorra a subárvore esquerda em pré-ordem: {Vazio}
            3. Percorra a subárvore direita em pré-ordem: {Vazio}
        3. Percorra a subárvore direita em pré-ordem:
            1. Visite a Raiz: {**70**}
            2. Percorra a subárvore esquerda em pré-ordem: {Vazio}
            3. Percorra a subárvore direita em pré-ordem: {Vazio}
3. Percorra a subárvore direita em pré-ordem:
    1. Visite a Raiz: {**85**};
    2. Percorra a subárvore esquerda em pré-ordem: {Vazio}
    3. Percorra a subárvore direita em pré-ordem: {Vazio}
        1. Visite a Raiz: {**100**};
        2. Percorra a subárvore esquerda em pré-ordem:
            1. Visite a Raiz: {**90**};

2. Percorra a subárvore esquerda em pré-ordem: {Vazio}
3. Percorra a subárvore direita em pré-ordem: {Vazio}

3. Percorra a subárvore direita em pré-ordem: {Vazio}

RESULTADO: 80, 30, 10, 60, 40, 70, 85, 100, 90

//Percorrendo a Árvore em In-Ordem:

1. Percorra a subárvore esquerda em in-ordem:
    1. Percorra a subárvore esquerda em in-ordem:
        1. Percorra a subárvore esquerda em in-ordem: {Vazio}
        2. Visite a Raiz: {**10**}
        3. Percorra a subárvore direita em in-ordem: {Vazio}
    2. Visite a Raiz: {**30**}
    3. Percorra a subárvore direita em in-ordem:
        1. Percorra a subárvore esquerda em in-ordem:
            1. Percorra a subárvore esquerda em in-ordem: {Vazio}
            2. Visite a Raiz: {**40**}
            3. Percorra a subárvore direita em in-ordem: {Vazio}
        2. Visite a Raiz: {**60**}
        3. Percorra a subárvore direita em in-ordem:
            1. Percorra a subárvore esquerda em in-ordem: {Vazio}
            2. Visite a Raiz: {**70**}
            3. Percorra a subárvore direita em in-ordem: {Vazio}
2. Visite a Raiz: {**80**}
3. Percorra a subárvore direita em in-ordem:
    1. Percorra a subárvore esquerda em in-ordem: {Vazio}
    2. Visite a Raiz: {**85**}
    3. Percorra a subárvore direita em in-ordem:
        1. Percorra a subárvore esquerda em in-ordem:
            1. Percorra a subárvore esquerda em in-ordem: {Vazio}
            2. Visite a Raiz: {**90**}
                3. Percorra a subárvore direita em in-ordem: {Vazio}
        2. Visite a Raiz: {**100**}
            3. Percorra a subárvore direita em in-ordem: {Vazio}

RESULTADO: 10, 30, 40, 60, 70, 80, 85, 90, 100.

//Percorrendo a Árvore em Pós-Ordem:

1. Percorra a subárvore esquerda em pós-ordem:
    1. Percorra a subárvore esquerda em pós-ordem:
        1. Percorra a subárvore esquerda em pós-ordem: {Vazio}
        2. Percorra a subárvore direita em pós-ordem: {Vazio}
        3. Visite a Raiz: {**10**}
    2. Percorra a subárvore direita em pós-ordem:
        1. Percorra a subárvore esquerda em pós-ordem: {Vazio}
            1. Percorra a subárvore esquerda em pós-ordem: {Vazio}
            2. Percorra a subárvore direita em pós-ordem: {Vazio}
            3. Visite a Raiz: {**40**}
        2. Percorra a subárvore direita em pós-ordem:
            1. Percorra a subárvore esquerda em pós-ordem: {Vazio}
            2. Percorra a subárvore direita em pós-ordem: {Vazio}
            3. Visite a Raiz: {**70**}
        3. Visite a Raiz: {**60**}
    3. Visite a Raiz: {**30**}
2. Percorra a subárvore direita em pós-ordem:
    1. Percorra a subárvore esquerda em pós-ordem: {Vazio}
    2. Percorra a subárvore direita em pós-ordem:
        1. Percorra a subárvore esquerda em pós-ordem:
            1. Percorra a subárvore esquerda em pós-ordem: {Vazio}
            2. Percorra a subárvore direita em pós-ordem: {Vazio}
            3. Visite a Raiz: {**90**}
        2. Percorra a subárvore direita em pós-ordem: {Vazio}
        3. Visite a Raiz: {**100**}
    3. Visite a Raiz: {**85**}
3. Visite a Raiz: {**80**}

RESULTADO: 10, 40, 70, 60, 30, 90, 100, 85, 80.

Agora vamos falar um pouquinho sobre três tipos especiais e árvores: Árvore B, Árvore B+ e Árvore AVL. E aí, eu preciso bastante da atenção de vocês agora! Eu coloco esse assunto porque os editais pedem apenas Árvore e não especificam a profundidade do assunto. Com exceção da CESGRANRIO, é raro outras bancas cobrarem esse assunto na profundidade que veremos agora.

É um assunto mais difícil e que cai pouco em prova, portanto só recomendo seguir caso queiram realmente cercar todas as possibilidades. Galera, época de faculdade, segundo semestre, disciplina de Estrutura de Dados! O trabalho final da disciplina era construir um compactador! Isso mesmo! Uma espécie de WinZip, WinRar, etc. E a estrutura usada para compactar arquivos de índices era uma Árvore B.

Uma árvore B é uma maneira de armazenar grandes quantidades de dados de tal forma que você pode procurá-los e recuperá-los muito rapidamente. As árvores B são a base da maioria dos bancos de dados modernos. Como eles funcionam é um bocado complicado, mas eu vou contar uma historinha que talvez os ajude a entender melhor! Vamo lá...

Imagine que você está procurando um par de novos fones de ouvido. Você tem algumas abordagens. *Certo?* Você poderia ir a todas as lojas do mundo até encontrar o produto que você procura. Como você pode imaginar, esta seria uma maneira horrível de fazer compras. Em vez disso, você poderia ir em uma FNAC, porque você sabe que eles vendem eletrônicos.

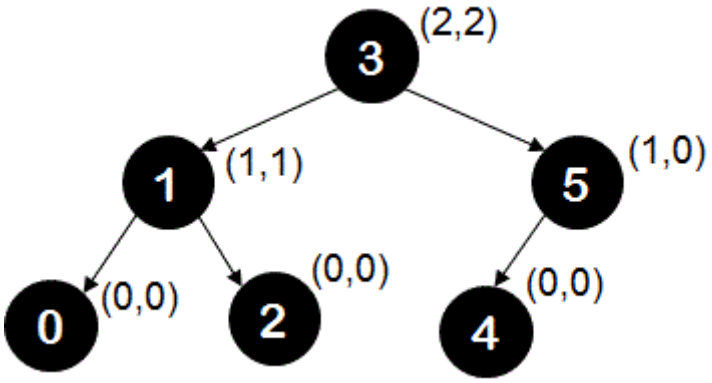
Uma vez que chegou à FNAC, você pode observar cada uma das descrições de corredor para ver onde os fones de ouvido são armazenados. Depois de encontrar o corredor correto, você pode escolher os fones de ouvido que você deseja. Ponto final! Observe como o objetivo desse processo é restringir o foco de uma pesquisa cada vez mais...

É assim que funcionam as Árvores B! Ao organizar os dados de forma específica, podemos aproveitá-las para que não desperdicemos nosso tempo buscando dados que tenham chance zero de armazenar os dados que estamos procurando. E a árvore já é construída de maneira a organizar os dados da melhor forma possível. *Bacana, pessoal?*

E qual a diferença de uma Árvore B para uma Árvore B+? Galera, a principal diferença é que, em uma Árvore B, as chaves e os dados podem ser armazenados tanto nos nós internos da árvore quanto nas folhas da árvore, enquanto que em uma Árvore B+ as chaves podem ser armazenadas em qualquer nó, mas os dados só podem ser armazenados nas folhas.

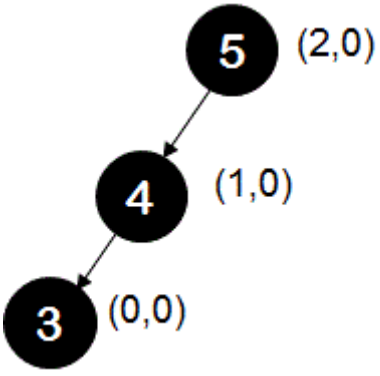
Por fim, vamos falar um pouco sobre Árvores AVL! Uma Árvore AVL (Adelson-Vesky e Landis) é uma Árvore Binária de Busca em que, para qualquer nó, a altura das subárvores da esquerda e da direita não podem ter uma diferença maior do que 1,

portanto uma Árvore AVL é uma Árvore Binária de Busca autobalanceável. Calma que nós vamos ver isso em mais detalhes...

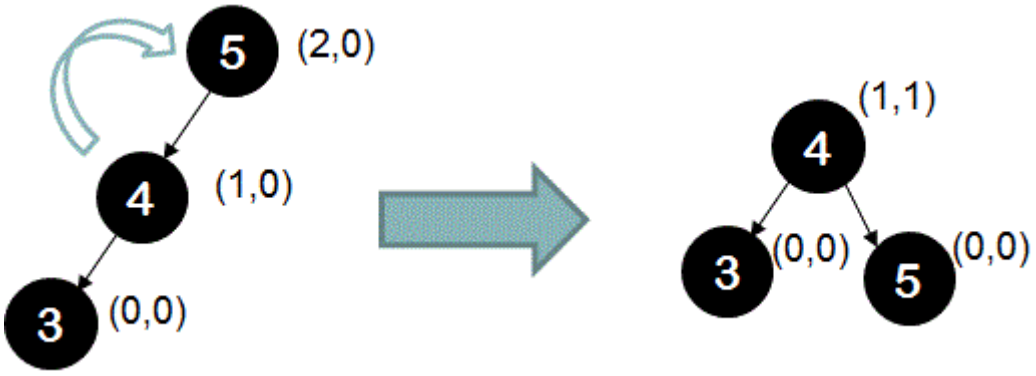


Observem o Nó 3: a altura da subárvore da esquerda é 2 e da subárvore da direita também é 2. Vejamos agora o Nó 1: a altura da subárvore da esquerda é 1 e da subárvore da direita também é 1. E o Nó 5: a altura da subárvore da esquerda é 1 e da subárvore da direita é 0 (visto que ela não existe). Vocês podem ver todos os nós e não encontrarão subárvore da esquerda e direita com diferença maior que 1.

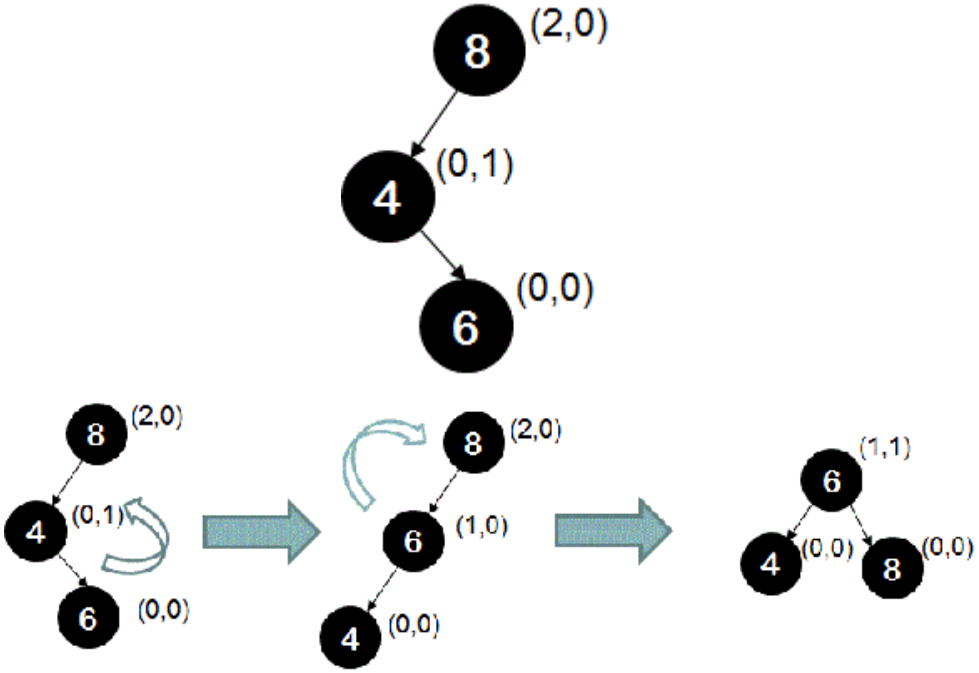
Uma Árvore AVL mantém seu equilíbrio de altura executando operações de rotação se algum dos nós violar a propriedade da diferença maior que 1. Exemplo 1: para a seguinte árvore, a propriedade da árvore AVL é violada no Nó 5, porque a subárvore esquerda possui altura 2, mas a subárvore da direita tem altura 0, então eles diferem em 2. *Entenderam?*



Essa diferença de 2 é construída nos dois ramos esquerdos - Ramo 5-4 e Ramo 4-3. *Concordam?* Se a diferença de 2 for construída por dois ramos esquerdos, chamamos esse caso de um Caso Esquerdo-Esquerdo (*Genial, né?*). Neste caso, realizamos uma rotação à direita no Ramo 5-4 como mostrado na imagem abaixo de forma a rebalancear a árvore.



Já na imagem abaixo, a Árvore AVL tem sua propriedade violada no Nó 8. A altura da subárvore esquerda é maior do que a altura da subárvore direita em 2. Essa diferença de 2 é construída por um ramo esquerdo (Ramo 8-4) e um ramo direito (Ramo 4-6), logo é um Caso Esquerdo-Direito. Para restaurar o equilíbrio de altura, executamos uma rotação esquerda seguida de uma rotação direita. Vejam...



Então, galera, caso a árvore não esteja balanceada, é necessário seu balanceamento através de rotações. No Caso Esquerda-Esquerda, basta fazer uma rotação simples para direita no nó desbalanceado. No caso Esquerda-Direita, temos que fazer uma rotação dupla, i.e., faz-se uma rotação para esquerda no nó filho e uma rotação para direita no nó desbalanceado.

No caso Direita-Direita, basta fazer uma rotação simples para a esquerda no nó desbalanceado. No caso Direita-Esquerda, temos que fazer uma rotação dupla, i.e.,

faz-se uma rotação para direita no nó filho, seguida de uma rotação para esquerda no nó desbalanceado. *Bacana?* Vocês entenderão isso melhor nos exercícios. Vamos ver agora a complexidade logarítmica dessas estruturas.

ÁRVORE BINÁRIA DE BUSCA		
ALGORITMO	CASO MÉDIO	PIOR CASO
Espaço	O(n)	O(n)
Busca	O(log n)	O(n)
Inserção	O(log n)	O(n)
Remoção	O(log n)	O(n)

ÁRVORE B / ÁRVORE AVL		
ALGORITMO	CASO MÉDIO	PIOR CASO
Espaço	O(n)	O(n)
Busca	O(log n)	O(log n)
Inserção	O(log n)	O(log n)
Remoção	O(log n)	O(log n)

Quem quiser brincar de Árvore Binária de Busca
https://www.cs.usfca.edu/~galles/visualization/BST.html
Quem quiser brincar de Árvore AVL
https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

2 COMPLEXIDADE DE ALGORITMOS

2.1 INTRODUÇÃO

Galera, por que estudamos a complexidade de algoritmos? Para determinar o custo computacional (tempo, espaço, etc) para execução de algoritmos. Em outras palavras**, ela classifica problemas computacionais de acordo com sua dificuldade inerente**. É importante entender isso para posteriormente estudarmos a complexidade de métodos de ordenação e métodos de pesquisa.

Nosso estudo aqui será bastante superficial por duas razões: primeiro, concursos cobram pouco e, quando cobram, querem saber as complexidades dos métodos de ordenação mais conhecidos; segundo, essa é uma disciplina absurdamente complexa que envolve Análise Assintótica, Cálculo Diferencial, Análise Polinomial (Linear, Exponencial, etc). **Logo, vamos nos ater ao que cai em concurso público**!

Só uma pausa: passei dias sem dormir na minha graduação por conta dessa disciplina! Na UnB, ela era ministrada pelo Instituto de Matemática e era considerada a disciplina mais difícil do curso :-(. Continuando: lá em cima eu falei sobre custo computacional! **Ora, para que eu escolha um algoritmo, eu preciso definir algum parâmetro**.

Podemos começar com Tempo! *Um algoritmo que realiza uma tarefa em 20 minutos é melhor do que um algoritmo que realiza uma tarefa em 20 dias?* **Não é uma boa estratégia, porque depende do computador que eu estou utilizando** (e todo o hardware correspondente), depende das otimizações realizadas pelo compilador, entre outras variáveis.

Vamos analisar, então, Espaço! *Um algoritmo que utiliza 20Mb de RAM é melhor do que um algoritmo que utiliza 20Gb?* Seguem os mesmos argumentos utilizados para o Tempo, ou seja, não é uma boa opção! *E agora, o que faremos?* Galera, eu tenho uma sugestão: **investigar a quantidade de vezes que operações são executadas na execução do algoritmo**!

Essa estratégia independe do computador (e hardware associado), do compilador, da linguagem de programação, das condições de implementação, entre outros fatores – ela depende apenas da qualidade inerente do algoritmo[5] implementado. **Utilizam-se algumas simplificações matemáticas para se ter uma ideia do comportamento do algoritmo**. Prosseguindo...

Dada uma entrada de dados de tamanho N, podemos calcular o custo computacional de um algoritmo em seu pior caso, médio caso e melhor caso! *Como assim, professor?* Para entender isso, vamos utilizar a metáfora de um jogo de baralho! Imaginem que eu estou jogando contra vocês. Vocês embaralham e me entregam 5 cartas, eu embaralho novamente e lhes entrego 5 cartas.

Quem joga baralho sabe que uma boa alternativa para grande parte dos jogos é ordenar as cartas em ordem crescente de modo a encontrar mais facilmente a melhor carta para jogar. Agora observem... vocês receberam as seguintes cartas (nessa ordem): 4, 5, 6, 7, 8. Já eu recebi as seguintes cartas

[5] Pessoal, é claro que nossa visão sobre a complexidade dos algoritmos é teórica. Na prática, depende de diversos outros fatores, mas nosso foco é na visão analítica e, não, empírica.

(também nessa ordem): 8, 7, 6, 5, 4 – **nós queremos analisar a complexidade de ordenação dessas cartas**.

Ora, convenhamos que vocês possuem o melhor caso, porque vocês deram a sorte de as cartas recebidas já estarem ordenadas. Já eu peguei o pior caso, porque as cartas estão ordenadas na ordem inversa. Por fim, o caso médio ocorre caso as cartas recebidas estejam em uma ordem aleatória. Com isso, espero que vocês tenham entendido o sentido de pior, médio e melhor casos.

Vamos partir agora para o estudo da Notação Big-O (ou Notação Assintótica)! Isso é simplesmente uma forma de representar o comportamento assintótico de uma função. No nosso contexto, ela busca expressar a quantidade de operações primitivas executadas como função do tamanho da entrada de dados. Vamos ver isso melhor!

A Notação Big-O é a representação relativa da complexidade de um algoritmo. É relativa porque só se pode comparar maçãs com maçãs, isto é, você não pode comparar um algoritmo de multiplicação aritmética com um algoritmo de ordenação de inteiros. **É uma representação porque reduz a comparação entre algoritmos a uma simples variável por meio de observações e suposições**.

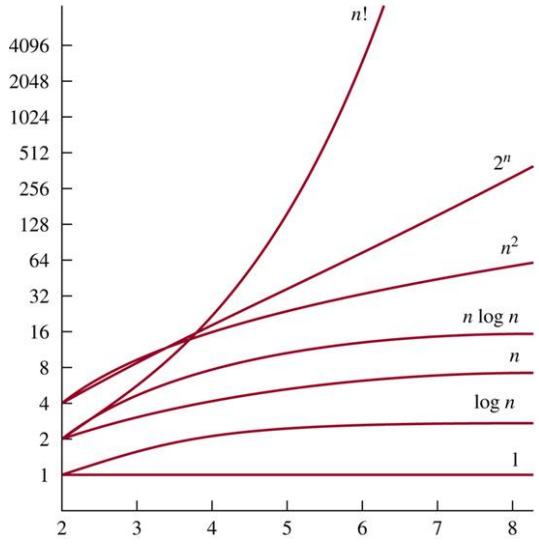
E trata da complexidade porque se é necessário 1 segundo para ordenar 10.000 elementos, quanto tempo levará para ordenar 1.000.000? **A complexidade, nesse exemplo particular, é a medida relativa para alguma coisa.** Vamos ver isso por meio de um exemplo: soma de dois inteiros! A soma é uma operação ou um problema, e o método para resolver esse problema é chamado algoritmo!

Transporte →	1	1	1	1	0
Parcela 1 →		1	1	0	1
Parcela 2 →	+	1	0	1	1
Soma →	1	1	0	0	0

Vamos supor que no algoritmo de somar (mostrado acima), a operação mais cara seja a adição. Observem que se somarmos dois números de 100 dígitos, teremos que fazer 100 adições. Se somarmos dois números de 10.000 dígitos, teremos que fazer 10.000 adições. *Perceberam o padrão?* **A complexidade (aqui, número de operações) é diretamente proporcional ao número *n* de dígitos, i.e., O(n)**.

Quando dizemos que um algoritmo é $O(n^2)$, estamos querendo dizer que esse algoritmo é da ordem de grandeza quadrática! Ele basicamente serve para te dizer quão rápido uma função cresce, por exemplo: um algoritmo $O(n)$ é melhor do que um algoritmo $O(n^2)$, porque ela cresce mais lentamente! **Abaixo vemos uma lista das classes de funções mais comuns em ordem crescente de crescimento**:

Notação	Nome
$O(1)$	Constante
$O(\log n)$	Logarítmica
$O[(\log n)^c]$	Polilogarítmica
$O(n)$	Linear
$O(n \log n)$	–
$O(n^2)$	Quadrática
$O(n^3)$	Cúbica
$O(n^c)$	Polinomial
$O(c^n)$	Exponencial
$O(n!)$	Fatorial



Quando dizemos que o Shellsort é um algoritmo $O(n^2)$, estamos querendo dizer que a complexidade (nesse caso, o número de operações) para ordenar um conjunto de n dados com o Algoritmo Shellsort é proporcional ao quadrado do número de elementos no conjunto! **Grosso modo, para ordenar 20 números, é necessário realizar 400 operações** (sem entrar em detalhes sobre a operação em si, nem sobre as simplificações matemáticas que são realizadas).

Entender como se chega a esses valores para cada método de ordenação e pesquisa é extremamente complexo! Galera, apesar de eu nunca ter visto isso em prova, é bom que vocês saibam que existem outras notações! Utiliza-se Notação Big-O (O) para pior caso; Notação Big-Ômega para melhor caso (Ω); e Notação Big-Theta (Θ) para caso médio.

Como na prática utiliza-se Big-O para tudo, o que eu recomendo (infelizmente, porque eu sei que vocês têm zilhões de coisas para decorar) é memorizar o pior caso dos principais métodos. Dessa forma, é possível responder a maioria das questões de prova sobre esse tema. Eventualmente, as questões pedem também caso médio e melhor caso, mas é menos comum. *Bacana?* :-)

Por último, uma pergunta muito frequente: Professor, já vi questões cobrando Logaritmo na Base 10, Logaritmo na Base 2, Logaritmo Neperiano, etc... *isso não está errado?* Galera, suponha que um algoritmo levou um tempo Θ(log10 n). Você também poderia dizer que levou um tempo Θ(lg n) (ou seja, Θ(log2 n)). **Você pode utilizar qualquer base.**

Sempre que a base do logaritmo é uma constante, não importa a base que usamos na notação assintótica. *Por que não?* Porque, para a notação assintótica, isso é completamente irrelevante.

Beleza? Então, não se prendam a base do logaritmo, qualquer uma pode ser utilizada na representação de complexidade assintótica de algoritmos. Bacana? Exercícios...

2.2 EXERCÍCIOS COMENTADOS: COMPLEXIDADE DE ALGORITMOS

1. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações - WEB Mobile / Análise de Sistemas)

Para projetar algoritmos eficientes um desenvolvedor deve estar preocupado com a complexidade deste algoritmo, desde sua concepção.

Considere a seguinte função T(n) que mede os recursos (ex. tempo de execução) que um algoritmo necessita no pior caso para processar uma entrada qualquer de tamanho n:

T(n) = O(log(n))

Sabendo que O(log(n)) é a ordem da complexidade de tempo do algoritmo seguindo a notação "big O", é correto afirmar que este algoritmo tem complexidade de ordem:
a) constante;
b) sublinear;
c) linear;
d) polinomial;
e) exponencial.

Gabarito: **Letra B**.

a) constante;
ERRADO - A complexidade é constante se T(n) = **O(1)**.

b) sublinear;
CERTO - O termo sublinear está sendo utilizado aqui com "menor que a linear". A função y = log(n) tem crescimento menor que y = n. O nome exato dessa complexidade é **Logarítmica**, mas não deixa de ser uma complexidade abaixo da linear, portanto, sublinear.

c) linear;
ERRADO - A complexidade é linear se T(n) = **O(n)**.

d) polinomial;
ERRADO - A complexidade é polinomial se T(n) = **$O(n^c)$**.

e) exponencial.
ERRADO - A complexidade é exponencial se T(n) = **$O(c^n)$**.

2. (FCC / Analista Judiciário (TRF 5ª Região) / 2017 / Informática - Desenvolvimento / Apoio Especializado)

Considere o algoritmo abaixo.

```
static int fibonacci(int n) {
  if (n <= 1) {
     return n;
  }
  return fibonacci(n - 2) + fibonacci(n - 1);
}
```

A complexidade deste algoritmo, na notação Big O, é
a) $O(2^n)$.
b) $O(n^2)$.
c) $O(n)$.
d) $O(\log(n))$.
e) $O(n^4)$.

Gabarito: **Letra A**.

A chamada da função fibonacci gera, potencialmente, 2 chamadas para a mesma função. Essas 2 chamadas geram, cada uma 2 chamadas, tornando-se, potencialmente, 4. Essas 4 geram 2 cada uma, tornando-se potencialmente 8. A cada iteração, dobramos.

Logo, após n iterações, faremos 2^n chamadas, o que significa que a complexidade é $O(2^n)$, **Letra A**.

3. (FGV / Analista Legislativo (ALERO) / 2018 / Infraestrutura de Redes e Comunicação / Tecnologia da Informação)

Considere a Sequência de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, ...), onde os dois primeiros termos valem 0 e 1 respectivamente, e cada termo seguinte é a soma de seus dois predecessores.

O pseudocódigo a seguir apresenta um algoritmo simples para o cálculo do N-ésimo termo dessa sequência.

```
function fibo (N)
if n = 1 then
     return 0
elif n = 2 then
     return 1
else
       penultimo := 0
       ultimo := 1
       for i := 3 until N do
            atual := penultimo + ultimo
```

```
            penultimo := ultimo
            ultimo := atual
        end for
        return atual
end if
```

Assinale a opção que mostra a complexidade desse algoritmo.
a) O(n/2)
b) O(n)
c) O(n2)
d) O(log n)
e) O(2n)

Gabarito: **Letra B**.

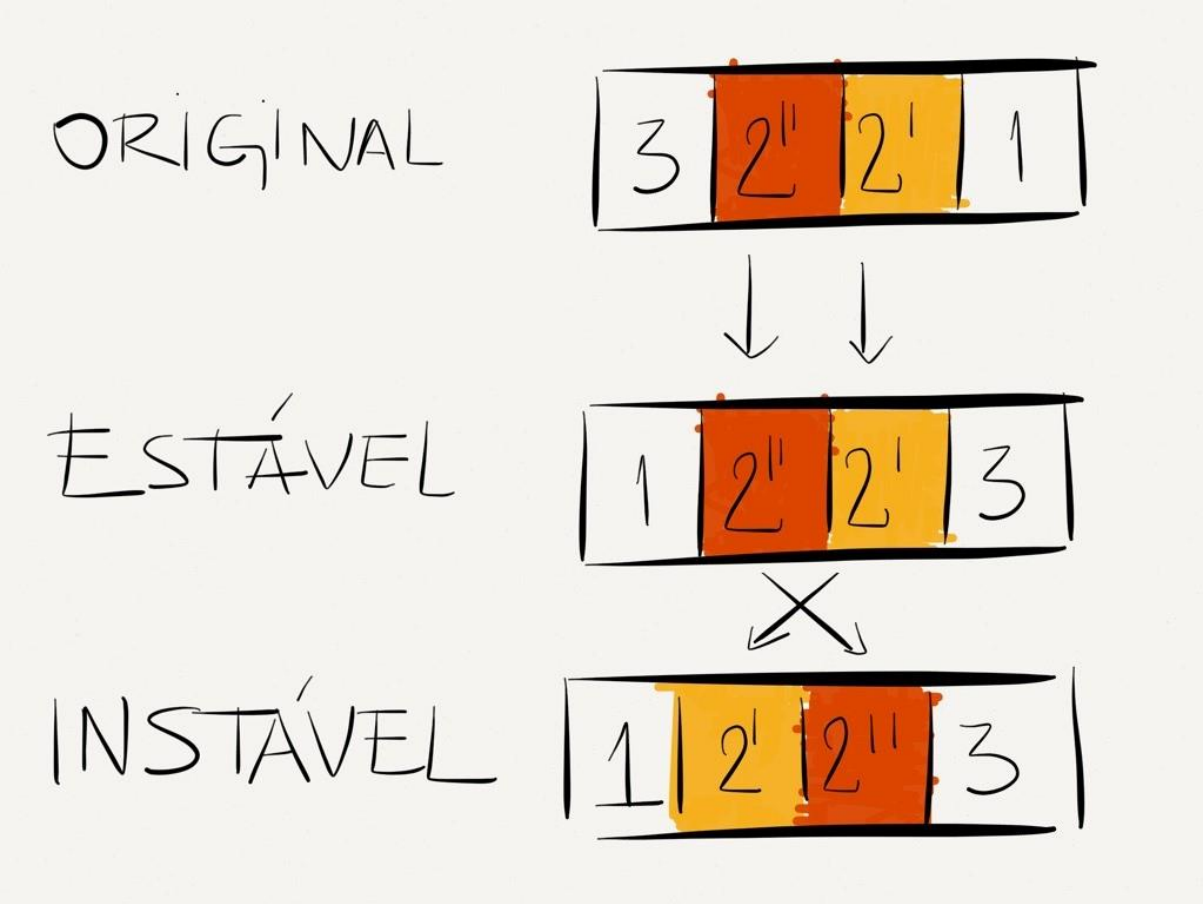
O algoritmo não usa recursividade, e possui apenas um loop. Esse loop itera de 3 até N. Portanto, o número de operações será proporcional a N, gerando uma complexidade O(N) - **Letra B**.

3 MÉTODOS DE ORDENAÇÃO

3.1 INTRODUÇÃO

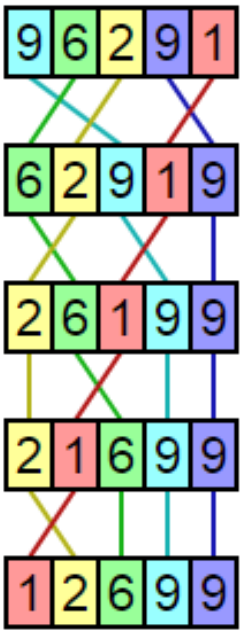
Métodos de Ordenação são algoritmos que têm o objetivo principal de posicionar os elementos de uma estrutura de dados em uma determinada ordem. *Para que, professor?* Ora, isso possibilita o acesso mais rápido e eficiente aos dados. Existem dezenas de métodos, todavia nessa aula veremos apenas os mais importantes: BubbleSort, QuickSort, InsertionSort, SelectionSort, MergeSort, ShellSort e HeapSort.

Antes de iniciar, vamos falar sobre o conceito de Estabilidade! Um método estável é aquele em que os itens com chaves iguais se mantêm com a posição inalterada durante o processo de ordenação, i.e., preserva-se a ordem relativa dos itens com chaves duplicadas ou iguais. **Métodos Estáveis: Bubble, Insertion e Merge; Métodos Instáveis: Selection, Quick, Heap e Shell**. Vejamos um exemplo:



Na imagem acima, foi colocado um sinal de aspas simples e duplas apenas para diferenciá-los, mas trata-se do mesmo número. Um algoritmo estável ordena todo o restante e não perde tempo trocando as posições de elementos que possuam chaves idênticas. Já um algoritmo instável ordena todos os elementos, inclusive aqueles que possuem chaves idênticas (sob algum outro critério).

3.2 BUBBLE SORT (2 A 2)



Esse algoritmo é o primeiro método de ordenação aprendido na faculdade, porque ele é bastante simples e intuitivo. Nesse método, os elementos da lista são movidos para as posições adequadas de forma contínua. Se um elemento está inicialmente em uma posição *i* e, para que a lista fique ordenada, ele deve ocupar a posição *j*, então ele terá que passar por todas as posições entre *i* e *j*.

Em cada iteração do método, percorremos a lista a partir de seu início comparando cada elemento com seu sucessor, trocando-se de posição se houver necessidade. É possível mostrar que, se a lista tiver *n* elementos, após no máximo *(n-1)* iterações, a lista estará em ordem (crescente ou decrescente). Observem abaixo o código para a Ordenação em Bolha:

```
ALGORITMO BOLHA
  ENTRADA: UM VETOR L COM N POSIÇÕES
  SAÍDA: O VETOR L EM ORDEM CRESCENTE

  PARA i = 1 até n - 1
    PARA j = 0 até n - 1 - i
      SE  L[j] >  L[j+1]
        AUX = L[j]          // SWAP
        L[j] = L[j+1]
        L[j+1] = AUX
FIM {BOLHA}
```

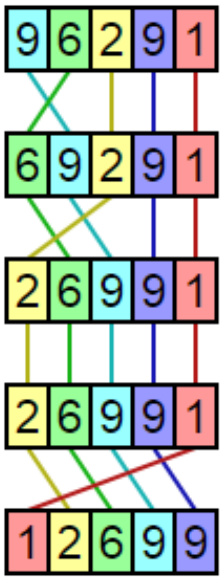
Melhor Caso	Caso Médio	Pior Caso
$\Omega(n)$	$\Omega(n^2)$	$\Omega(n^2)$

3.3 INSERTION SORT (INSIRA)

Esse algoritmo, também conhecido como Inserção Direta, é bastante simples **e apresenta um desempenho significativamente melhor que o BubbleSort, em termos absolutos**. Além disso, ele é extremamente eficiente para listas que já estejam substancialmente ordenadas e listas com pequeno número de elementos. Observem abaixo o código para a Ordenação de Inserção:

```
ALGORITMO INSERÇÃO
  ENTRADA: UM VETOR L COM N POSIÇÕES
  SAÍDA: O VETOR L EM ORDEM CRESCENTE

  PARA i = 1 até n - 1
    PIVO = L[i]
    j = i - 1
    ENQUANTO j ≥ 0 e L[j] > PIVO
      L[j+1] = L[j]
      j = j - 1
    L[j+1] = PIVO
FIM {INSERÇÃO}
```

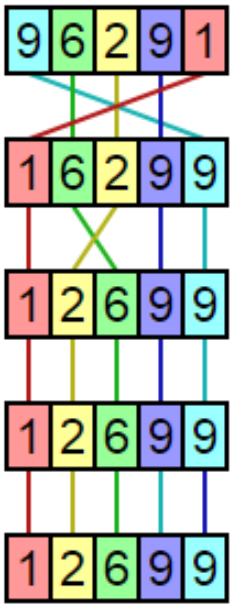


Nesse método, consideramos que a lista está dividida em parte esquerda, já ordenada, e parte direita, em possível desordem. Inicialmente, a parte esquerda contém apenas o primeiro elemento da lista. **Cada iteração consiste em inserir o primeiro elemento da parte direta (pivô) na posição adequada da parte esquerda, de modo que a parte esquerda continue ordenada**.

É fácil perceber que se a lista possui *n* elementos, após (n-1) inserções, ela estará ordenada. Para inserir o pivô, percorremos a parte esquerda, da direita para a esquerda, deslocando os elementos estritamente maiores que o pivô uma posição para direita. **O pivô deve ser colocado imediatamente à esquerda do último elemento movido**.

Melhor Caso	Caso Médio	Pior Caso
$\Omega(n)$	$\Omega(n^2)$	$\Omega(n^2)$

3.4 SELECTION SORT (SELECIONE)



Esse algoritmo consiste em selecionar o menor[6] elemento de um vetor e trocá-lo *(swap)* pelo item que estiver na primeira posição, i.e., inseri-lo no início do vetor. Essas duas operações são repetidas com os itens restantes até o último elemento. Tem como ponto forte o fato de que realiza poucas operações de *swap*. Seu desempenho costuma ser superior ao BubbleSort e inferior ao InsertionSort.

Assim como no InsertionSort, considera-se que a lista está dividida em parte esquerda, já ordenada, e parte direita, em possível desordem. Ademais, os elementos da parte esquerda são todos menores ou iguais aos elementos da parte direita. **Cada iteração consiste em selecionar o menor elemento da parte direita (pivô) e trocá-lo com o primeiro elemento da parte direita.**

Assim, a parte esquerda aumenta, visto que passa a incluir o pivô, e a parte direita diminui. Note que o pivô é maior que todos os demais elementos da parte esquerda, portanto a parte esquerda continua ordenada. Ademais, o pivô era o menor elemento da direita, logo todos os elementos da esquerda continuam sendo menores ou iguais aos elementos da direita. Inicialmente, a parte esquerda é vazia.

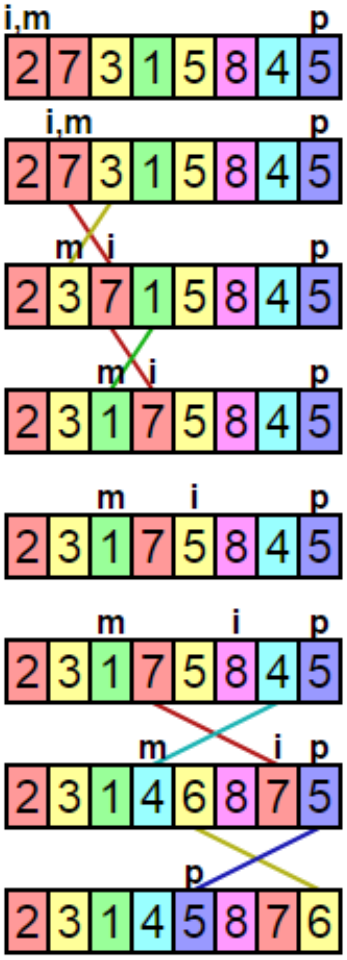
```
ALGORITMO SELEÇÃO
  ENTRADA: UM VETOR L COM N POSIÇÕES
  SAÍDA: O VETOR L EM ORDEM CRESCENTE

  PARA i = 0 ate n - 2
    MIN = i
    PARA j = i + 1 até n - 1
      SE L[j] < L[MIN]
        MIN= j
    TROCA(L[i], L[MIN])
FIM {SELEÇÃO}
```

Melhor Caso	Caso Médio	Pior Caso
$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$

[6] A definição formal afirma que é o maior valor; a maioria das implementações utiliza o menor valor. As questões de prova cobram algumas vezes o maior, outras vezes o menor.

3.5 QUICKSORT (PIVÔ)



Esse algoritmo divide um conjunto de itens em conjuntos menores, que são ordenados de forma independente, e depois os resultados são combinados para produzir a solução de ordenação do conjunto maior. **Trata-se, portanto, de um algoritmo do tipo *Divisão-e-Conquista*, i.e., repartindo os dados em subgrupos, dependendo de um elemento chamado pivô**.

Talvez seja o método de ordenação mais utilizado! Isso ocorre porque quase sempre ele é significativamente mais rápido do que todos os demais métodos de ordenação baseados em comparação. **Ademais, suas características fazem com que ele, assim como o MergeSort, possa ser facilmente paralelizado.** Ele também pode ser adaptado para realizar ordenação externa (QuickSort Externo).

Neste método, a lista é dividida em parte esquerda e parte direita, sendo que os elementos da parte esquerda são todos menores que os elementos da parte direita. Essa fase do processo é chamada de partição. **Em seguida, as duas partes são ordenadas recursivamente (usando o próprio QuickSort). A lista está, portanto, ordenada corretamente**!

Uma estratégia para fazer a partição é escolher um valor como pivô e então colocar na parte esquerda os elementos menores ou iguais ao pivô e na parte direita os elementos maiores que o pivô – galera, a escolha do pivô é crítica! **Em geral, utiliza-se como pivô o primeiro elemento da lista, a despeito de existirem maneiras de escolher um "melhor" pivô**.

Esse algoritmo é um dos métodos mais rápidos de ordenação, apesar de às vezes partições desequilibradas poderem conduzir a uma ordenação lenta. A eficácia do método depende da escolha do pivô mais adequado ao conjunto de dados que se deseja ordenar. Alguns, por exemplo, utilizam a mediana de três elementos para otimizar o algoritmo.

Alguns autores consideram a divisão em três subconjuntos, sendo o terceiro contendo valores iguais ao pivô. O Melhor Caso ocorre quando o conjunto é dividido em subconjuntos de mesmo tamanho; o Pior Caso ocorre quando o pivô corresponde a um dos extremos (menor ou maior valor). Alguns o consideram um algoritmo frágil e não-estável, com baixa tolerância a erros.

```
PROCEDIMENTO PARTIÇÃO
  ENTRADA: UM VETOR L E AS POSIÇÕES INICIO E FIM
  SAÍDA:j, tal que L[INICIO]..L[j-1] ≤ L[j] e
        L[j+1]..L[FIM] > L[j]
     // j é a posição onde será colocado o pivô, como
     // ilustrado na figura abaixo
       PIVO = L[INICIO], i = INICIO + 1, j = FIM
        ENQUANTO i ≤ j
          ENQUANTO i ≤ j e L[i] ≤ PIVO
            i = i + 1
          ENQUANTO L[j] > PIVO
            j = j - 1
          SE i ≤ j
            TROCA(L[i],L[j])
            i = i + 1, j = j - 1
        TROCA(L[INICIO], L[j])
        DEVOLVA j
      FIM {PARTIÇÃO}
```

Melhor Caso	Caso Médio	Pior Caso
$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$

3.6 SHELLSORT (GAP)

Esse algoritmo é uma extensão ou refinamento do algoritmo do InsertionSort, contornando o problema que ocorre quando o menor item de um vetor está na posição mais à direita. Ademais, difere desse último pelo fato de, em vez de considerar o vetor a ser ordenado como um único segmento, **ele considera vários segmentos e aplica o InsertionSort em cada um deles.**

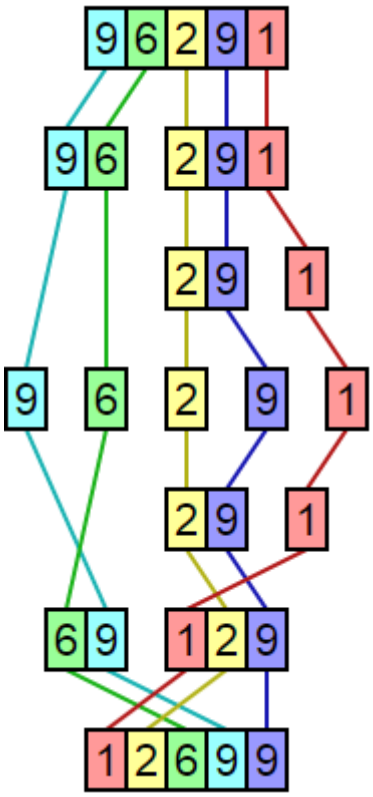
É o algoritmo mais eficiente dentre os de ordem quadrática. Nesse método, as comparações e as trocas são feitas conforme determinada distância *(gap)* entre dois elementos, de modo que, se gap = 6, há comparação entre o 1º e 7º elementos ou entre o 2º e 8º elementos e assim sucessivamente, repetindo até que as últimas comparações e trocas tenham sido efetuadas e o gap tenha chegado a 1.

```
ALGORITMO SHELLSORT
  ENTRADA: UM VETOR L COM N POSIÇÕES
  SAÍDA: O VETOR L EM ORDEM CRESCENTE

  H = 1
  ENQUANTO H < n FAÇA H = 3 * H + 1
  FAÇA
    H = H / 3            // divisão inteira
    PARA i = H até n - 1 // Inserção adaptado para h-listas
      PIVO = L[i]
      j = i - H
      ENQUANTO j ≥ 0 e L[j] > PIVO
        L[j + H] = L[j]
        j= j - H
      L[j + H] = PIVO
  ENQUANTO H > 1
     FIM {SHELLSORT}
```

Melhor Caso	Caso Médio	Pior Caso
O(n log n)	Depende do *gap*	$O(n^2)$

3.7 MERGESORT (MESCLA)



Esse algoritmo é baseado na estratégia de resolução de problemas conhecida como *divisão-e-conquista*. Essa técnica consiste basicamente em decompor a instância a ser resolvida em instâncias menores do mesmo tipo de problema, resolver tais instâncias (em geral, recursivamente) e por fim utilizar as soluções parciais para obter uma solução da instância original.

Naturalmente, nem todo problema pode ser resolvido através de divisão e conquista. Para que seja viável aplicar essa técnica a um problema, ele deve possuir duas propriedades estruturais. **O problema deve ser decomponível**, i.e., deve ser possível decompor qualquer instância não trivial do problema em instâncias menores do mesmo tipo de problema.

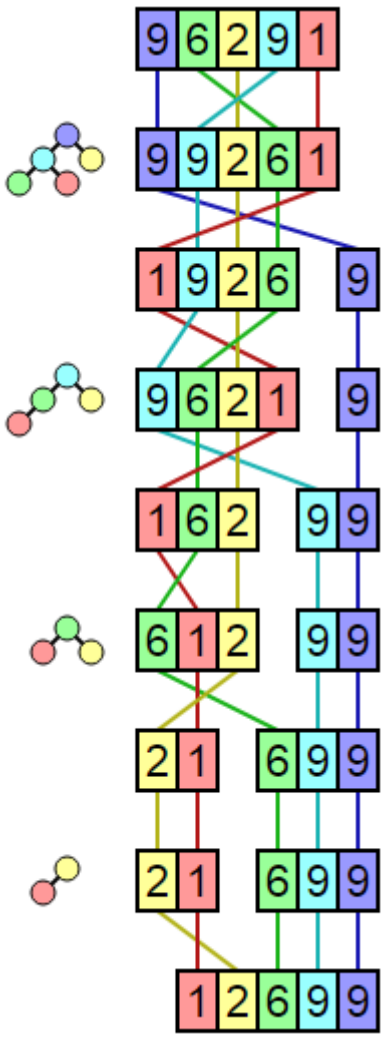
Além disso, deve ser sempre possível utilizar as soluções obtidas com a resolução das instâncias menores para chegar a uma solução da instância original. No MergeSort, divide-se a lista em duas metades. Essas metades são ordenadas recursivamente (usando o próprio MergeSort) e depois são intercaladas. Abaixo segue uma possível solução:

```
ALGORITMO MERGESORT
  ENTRADA: UM VETOR L E AS POSIÇÕES INICIO E FIM
  SAÍDA: O VETOR L EM ORDEM CRESCENTE DA POSIÇÃO INICIO ATÉ
         A POSIÇÃO FIM

  SE inicio < fim
    meio = (inicio + fim)/2          // divisão inteira
    SE inicio < meio
      MERGESORT(L, inicio, meio)
    SE meio + 1 < fim
      MERGESORT(L, meio + 1, fim)
    MERGE(L, inicio, meio, fim)
FIM {MERGESORT}
```

Melhor Caso	Caso Médio	Pior Caso
O(n log n)	O(n log n)	O(n log n)

3.8 HEAPSORT (MAX HEAP)



Esse algoritmo utiliza uma estrutura de dados chamada heap, para ordenar os elementos à medida que os insere na estrutura.

Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada.

Essa estrutura pode ser representada como uma árvore ou como um vetor. *Entenderam?* **Inicialmente, insere-se os elementos da lista em um heap.**

Em seguida, fazemos sucessivas remoções do menor elemento do heap, colocando os elementos removidos do heap de volta na lista – a lista estará então em ordem crescente.

O heapsort é um algoritmo de ordenação em que a sua estrutura auxiliar de armazenamento fora do arranjo de entrada é constante durante toda a sua execução.

```
ALGORITMO HEAP SORT
  ENTRADA: UM VETOR L COM N POSIÇÕES
  SAÍDA: O VETOR L EM ORDEM CRESCENTE

inicialize um HBC H com n posições
PARA i = 0 até n - 1
  INSERE_HBC(H, L[i])
PARA i = 0 até n - 1
  L[i] = REMOVE_MENOR(H)
FIM {HEAP SORT}
```

Melhor Caso	Caso Médio	Pior Caso
O(n log n)	O(n log n)	O(n log n)

3.9 RESUMÃO DOS TERMOS CHAVES

- **Bubble Sort**: Compare **um elemento com o seguinte**;
- **Insertion Sort**: Tome da parte desordenada, e **insira na posição certa** da ordenada
- **Selection Sort**: **Selecione o maior** da desordenada e coloque no começo da ordenada (ou **selecione o menor** da desordenada e coloque no final da ordenada)
- **Quick Sort**: Escolha um **pivô** e divida os elementos em 2 conjuntos – maiores e menores que o pivô. Repita para os conjuntos.
- **Shell Sort**: Escolha um **GAP**, compare os elementos separados pelo **GAP**, e vá diminuindo o **GAP**.
- **Merge Sort:** Vá separando os conjuntos na metade. Quando chegar em 2 elementos, **volte fazendo a mescla** de forma ordenada.
- **Heap Sort:** Forme um HEAP, aplique o MAX HEAP e remova a RAIZ.

3.10 RESUMÃO DAS COMPLEXIDADES

ALGORITMO	MELHOR CASO	CASO MÉDIO	PIOR CASO
BubbleSort	$O(n)$	$O(n^2)$	$O(n^2)$
InsertionSort	$O(n)$	$O(n^2)$	$O(n^2)$
SelectionSort	$O(n^2)$	$O(n^2)$	$O(n^2)$
QuickSort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
ShellSort	$O(n \log n)$	Depende do *gap*	$O(n^2)$
MergeSort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
HeapSort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

3.11 EXERCÍCIOS COMENTADOS: ORDENAÇÃO

1. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações / Análise de Sistemas)

O algoritmo de ordenação baseado em vários percursos sobre o *array*, realizando, quando necessárias, trocas entre pares de elementos consecutivos denomina-se método:
a) das trocas (*exchange sort*);
b) da inserção (*insertion sort*);
c) da bolha (*bubble sort*);
d) da seleção (*selection sort*);
e) da permuta (*permutation sort*).

Gabarito: Letra C.

a) das trocas (*exchange sort*);
ERRADO. É um primo do bubble-sort, só muda a forma de comparação. O bubble compara uma posição com a seguinte, e o exchange compara a mesma posição com todas as seguintes.

b) da inserção (*insertion sort*);

ERRADO. O insertion-sort separa o array entre parte ordenada e parte desordenada (inicialmente a parte ordenada é apenas o 1o. elemento, e todos os outros são a parte desordenada). Depois toma item por item da parte desordenada, e posicionando-o corretamente na parte ordenada da coleção.

c) da bolha (*bubble sort*);
CERTO. Compara um item com o posterior, trocando-os se estiverem na ordem errada.

d) da seleção (*selection sort*);
ERRADO. O selection-sort separa o array entre parte ordenada e parte desordenada (inicialmente a parte ordenada está vazia, e todos os elementos estão na parte desordenada). Depois toma o maior item da parte desordenada, e posicionando-o no início na parte ordenada.

e) da permuta (*permutation sort*).
ERRADO. Esse é o método mais "louco" de ordenação... não deve nem ser levado a sério. Consiste em permutar os elementos, e testar para ver se estão em ordem. Se não tiver, faz outra permutação, e testa, e fica repetindo essa tentativa e erro até que a coleção esteja ordenada.

Portanto, a **Letra C** é a resposta correta.

2. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações - WEB Mobile / Análise de Sistemas)

Considere o seguinte algoritmo, responsável por realizar a ordenação de um array de dados.

```
public int[] mySortingAlgorithm (int[] date){
  int size = data.length;
  int tmp = 0;
  for(int i = 0;i<size;i++){
   for(int j = (size-1);j>=(i+1);j--){
     if(data[j]<data[j-1]){
        tmp = data[j];
        data[j]=data[j-1];
        data[j-1]=tmp;
      }
    }
  }
  return data;
}
```

Podemos afirmar que o método de ordenação utilizado pelo algoritmo é o:
a) quickSort;
b) insertionSort;
c) mergeSort;
d) shellSort;
e) bubbleSort.

Gabarito: **Letra E**.

Analisando o algoritmo, vemos que ele navega de trás para frente do array, comparando sempre um número com o anterior, e trocando-os de posição caso estejam na ordem errada (vide o trecho abaixo):

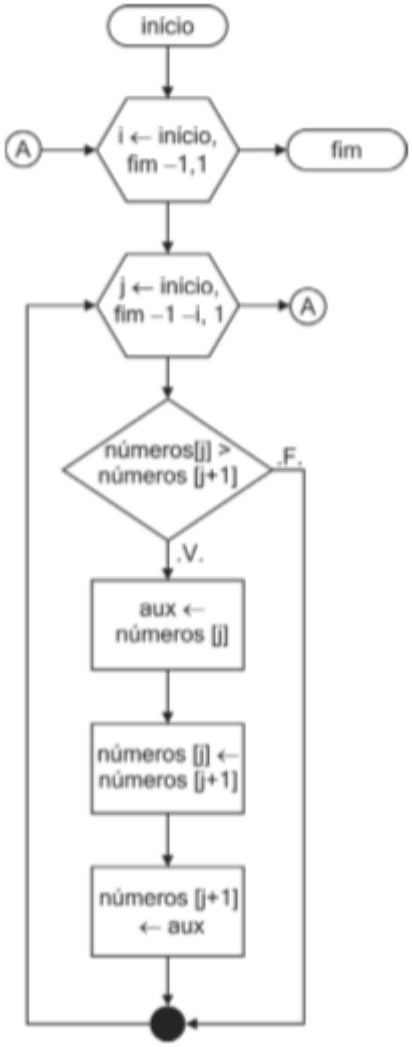
```
if(data[j]<data[j-1]){ //se o item na posição j
                       //for maior que na posição j-1
```

```
    tmp = data[j];
    data[j]=data[j-1];  // troca os itens de lugar
    data[j-1]=tmp;
}
```

Essa é uma variação bem comum do algoritmo Bubble Sort - Compara um item com o seguinte, e troca-os se necessário.

Portanto, a **Letra E** está correta.

3. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)



Se, no fluxograma precedente, *início* indica o primeiro elemento do vetor e *fim*, o último elemento, então, para o vetor [11,6,2,7,8,3,5], o resultado final é

a) [2,7,3,5].
b) [11,7,3,5].
c) [11,6,2,7,8,3,5].
d) [2,3,5,6,7,8,11].
e) [6,2,8,3,5].

Gabarito: **Letra D**.

O algoritmo itera da primeira posição até a penúltima, sempre comparando a posição em evidência com a próxima. Se o elemento atual for maior que o seu sucessor, troca os dois de lugar.

Pessoal, esse é o algoritmo de ordenação conhecido como **Bubble Sort**, ou **Método da Bolha**. Ele ordena uma estrutura de dados, sem remover nenhum elemento.

Portanto, tendo como entrada o vetor [11,6,2,7,8,3,5], o resultado será o mesmo vetor com os elementos ordenados: **[2,3,5,6,7,8,11], Letra D**.

DICA: Sempre que você identificar em um algoritmo a comparação var[j] > var[j+1], desconfie que trata-se do Bubble Sort. Para confirmar, verifique se o bloco que segue esse teste troca os elementos de posição. Esse é o algoritmo mais cobrado em provas!

4. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Para responder à questão a seguir, considere a estratégia de ordenação apresentada em Java abaixo.

```
 private static void ordena(int[] vetor, int inicio, int fim) {
     if (inicio < fim) {
             int posicaoP = separar(vetor, inicio, fim);
             ordena(vetor, inicio, posicaoP - 1);
             ordena(vetor, posicaoP + 1, fim);
     }
}

private static int separar(int[] vetor, int inicio, int fim) {
     int P = vetor[inicio];
     int i = inicio + 1, f = fim;
     while (i <= f) {
             if (vetor[i] <= P)
                     i++;
             else if (P < vetor[f])
                     f--;
             else { int troca = vetor[i];
                     vetor[i] = vetor[f];
                     vetor[f] = troca;
                     i++;
                     f--;
             }
     }
     vetor[inicio] = vetor[f];
     vetor[f] = P;
     return f;
}
```

A estratégia apresentada em Java é o método de ordenação

a) Bubblesort.

b) Mergesort.
c) Insertion Sort.
d) Quicksort.
e) Heapsort.

Gabarito: **Letra D**

A estratégia está baseada na escolha de um pivô, conforme acontece na declaração: `int P = vetor[inicio];`

Portanto, podemos eliminar:

a) ~~Bubblesort~~. Compara um número com o seguinte, sem pivô.
b) ~~Mergesort~~. Separa em sub-vetores, ordenando-os. Volta comparando e fazendo uma fusão dos vetores ordenados. Não usa pivô.
e) ~~Heapsort~~. Ordena recursivamente a partir de um Max Heap. Não está sendo utilizado nenhum heap na solução. Também não utiliza pivô.

Sobram a letra C e D.

O Insertion Sort normalmente faz a análise somente em um sentido, e o código em análise olha o início e o fim da estrutura, caminhando incrementando o início (i++) e decrementando o fim (f--) para o posicionamento correto do pivot. Esse já é motivo suficiente para eliminar o Insertion Sort, e, portanto, marcar **letra D - Quicksort**.

Para quem quer saber exatamente o nome desse algoritmo, temos aqui um Quicksort com a Partição de Hoare.

5. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Para responder à questão a seguir, considere a estratégia de ordenação apresentada em Java abaixo.

```
 private static void ordena(int[] vetor, int inicio, int fim) {
     if (inicio < fim) {
             int posicaoP = separar(vetor, inicio, fim);
             ordena(vetor, inicio, posicaoP - 1);
             ordena(vetor, posicaoP + 1, fim);
     }
}

private static int separar(int[] vetor, int inicio, int fim) {
     int P = vetor[inicio];
     int i = inicio + 1, f = fim;
     while (i <= f) {
             if (vetor[i] <= P)
                     i++;
             else if (P < vetor[f])
                     f--;
             else { int troca = vetor[i];
                     vetor[i] = vetor[f];
                     vetor[f] = troca;
```

```
                    i++;
                    f--;
                }
        }
        vetor[inicio] = vetor[f];
        vetor[f] = P;
        return f;
}
```

Considerando que N é número de elementos do vetor a ser ordenado, a estratégia de ordenação apresentada em Java

a) também é conhecida como método de ordenação por intercalação e possui uma versão para unir dois vetores já ordenados.

b) tem complexidade $O(N^2)$ no pior caso e no caso médio, mas apresenta complexidade O(N) no melhor caso.

c) faz um número fixo de comparações dado por $\log_2 N$, independente dos valores do vetor original. Isso é garantido pelas chamadas recursivas ao método ordena().

d) utiliza o método separar() para dividir o vetor original em 2 sublistas de igual tamanho. Isso garante que mesmo no pior caso o método realize Nlog2N comparações.

e) utiliza o método separar() para fazer a partição do vetor, por meio da seleção de um elemento chamado pivô. A escolha do pivô é crucial para o bom desempenho do método, já que a fase de partição é a parte crítica do algoritmo.

Gabarito: **Letra E**.

a) também é conhecida como método de ~~ordenação por intercalação~~ e ~~possui uma versão para unir dois vetores já ordenados~~.
ERRADO. A definição de ordenação por intercalação (merge sort) está correta, mas o método utilizado é o **Quicksort**.

b) tem complexidade $O(N^2)$ no pior caso e no caso médio, mas apresenta complexidade ~~O(N)~~ no melhor caso.
ERRADO. Para o **Quicksort**, o pior caso está correto $O(N^2)$, mas o melhor caso e o caso médio possuem complexidade O(N log N).

c) faz um ~~número fixo de comparações dado por $\log_2 N$~~, independente dos valores do vetor original. Isso é garantido pelas chamadas recursivas ao método ordena().
ERRADO. O número de comparações do Quicksort será sempre maior que $\log_2 N$.

d) utiliza o método separar() para dividir o vetor original em 2 sublistas ~~de igual tamanho~~. Isso garante que mesmo no pior caso o método realize Nlog2N comparações.
ERRADO. A divisão em 2 arrays não é igualitária, e vai depender da posição correta do pivô no array ordenado. Se a posição correta do pivô for no início, por exemplo, o sub-array da esquerda nem vai existir.

e) utiliza o método separar() para fazer a partição do vetor, por meio da seleção de um elemento chamado pivô. A escolha do pivô é crucial para o bom desempenho do método, já que a fase de partição é a parte crítica do algoritmo.
CORRETO. No Quicksort a escolha do pivô pode ser um algoritmo por si só. Existem diversas possibilidades de partição como, por exemplo, a Hoares e Lomuto. A escolha de pivô e do algoritmo de partição podem mudar completamente a performance do método.

6. (CESGRANRIO / Analista (PETROBRAS) / 2018 / / Sistema Júnior)

Dada a sequência numérica (15,11,16,18,23,5,10,22,21,12) para ordenar pelo algoritmo Selection Sort, qual é a sequência parcialmente ordenada depois de completada a quinta passagem do algoritmo?

a) [15, 11, 16, 18, 12, 5, 10, 21, 22, 23]
b) [15, 11, 5, 10, 12, 16, 18, 21, 22, 23]
c) [15, 11, 16, 10, 12, 5, 18, 21, 22, 23]
d) [10, 11, 5, 12, 15, 16, 18, 21, 22, 23]
e) [12, 11, 5, 10, 15, 16, 18, 21, 22, 23]

Gabarito: **Letra B**

O método **selection sort** separa o array em **parte ordenada** (normalmente à direita) e **parte desordenada** (normalmente à esquerda). O método consiste em pegar o maior número da parte desordenada e colocar no início parte ordenada. Pode-se começar a partir do penúltimo valor.

Sendo assim:

Inicial: [15,11,16,18,23,5,10,22,21][12]
1a. passagem: [15,11,16,18,~~**23**~~,5,10,22,21][12,**23**]
2a. passagem: [15,11,16,18,5,10,~~**22**~~,21][12,**22**,23]
3a. passagem: [15,11,16,18,5,10,~~**21**~~][12,**21**,22,23]
4a. passagem: [15,11,16,~~**18**~~,5,10][12,**18**,21,22,23]
5a. passagem: [15,11,~~**16**~~,5,10][12,**16**,18,21,22,23]

Portanto, na 5a. passagem o vetor estará com a sequencia [15,11,5,10,12,16,18,21,22,23], **Letra B**.

7. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

O método de ordenação conhecido como quick sort utiliza o maior elemento, o qual é sempre colocado ao final do vetor, para garantir que a ordenação seja realizada em ordem decrescente.

Gabarito: **ERRADO**.

O método *quick sort* utiliza um pivô. Passa pelo array, organizando elementos menores que o pivô à esquerda, e maiores que o pivô à direita. No final de uma iteração, o pivô está na posição correta. Ele repete então o algoritmo na coleção à esquerda e à direita.

O método que passa selecionando o maior elemento e colocando-o em ordem é o ***Selection Sort***. Ess algoritmo separa o vetor em parte desordenada (à esquerda) e parte ordenada (à direita). A cada iteração, o posicionamento do maior elemento da parte desordenada ocorre no início da parte ordenada (ou final da parte desordenada, que dá no mesmo).

Reescrevendo o item:

O método de ordenação conhecido como ~~*quick sort*~~ ***selection sort*** utiliza o maior elemento **do sub-vetor desordenado**, o qual é sempre colocado ~~ao final do vetor~~ **no início do sub-vetor ordenado** para garantir que a ordenação seja realizada em ordem decrescente.

8. (AOCP / Técnico em Gestão de Infraestrutura (SUSIPE) / 2018 / / Gestão de Informática)

Assinale a alternativa correta a respeito dos principais algoritmos de ordenação.
a) O algoritmo de ordenação QuickSort é um algoritmo eficiente, porém deve ser implementado visando a uma boa escolha do elemento pivô.
b) O algoritmo de ordenação por inserção tem uma implementação cara, porém com um desempenho estável.
c) O algoritmo de ordenação HeapSort é um algoritmo eficiente em relação ao tempo, porém ineficiente em relação à memória.
d) O algoritmo de ordenação ShellSort tem sua principal desvantagem quando os dados a serem ordenados estão parcialmente ordenados.
e) O algoritmo de ordenação por seleção tem uma implementação cara, porém com desempenho estável.

Gabarito: **Letra A**.

a) O algoritmo de ordenação QuickSort é um algoritmo eficiente, porém deve ser implementado visando a uma boa escolha do elemento pivô.
CORRETO. Essa é uma característica marcante do método QuickSort. Ele trabalha ordenando a partir da escolha de um pivô, colocando os itens menores à esquerda, e maiores à direita. Dessa forma, a escolha do pivô tem um papel central no método.

b) O algoritmo de ordenação por inserção tem uma ~~implementação cara~~, porém com um desempenho estável.
INCORRETO. O *insertion sort* têm implementação muito simples: toma o próximo valor desordenado, e insere-o em posição na coleção ordenada.

c) O algoritmo de ordenação HeapSort é um algoritmo eficiente em relação ao tempo, porém ineficiente em relação à memória.
INCORRETO. O *heap sort* é bem eficiente com relação à memória, uma vez que a estrutura de *Heap* utilizada pelo método pode ser feita sobre o próprio vetor de entrada, sem a necessidade de alocação extra de memória.

d) O algoritmo de ordenação ShellSort tem sua principal desvantagem quando os dados a serem ordenados estão parcialmente ordenados.
INCORRETO. O *shell sort* faz a ordenação a partir da comparação de 2 elementos no array de distâncias (GAPs) variáveis. O pior caso depende do posicionamento dos elementos com relação aos GAPs. O método que tem seu pior caso em arrays ordenados (ou quase ordenados) é o *quick sort*.

e) O algoritmo de ordenação por seleção tem uma implementação cara, porém com desempenho estável.
INCORRETO. O *selection sort* têm implementação muito simples: toma o maior valor desordenado, e insere-o no início da coleção ordenada.

9. (CESGRANRIO / Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Processos de Negócio)

Analise o algoritmo de ordenação que se segue.

```
def ordenar(dado):
    for passnum in range(len(dado)-1,0,-1):
      for i in range(passnum):
          if dado[i]>dado[i+1]:
             temp = dado[i]
             dado[i] = dado[i+1]
             dado[i+1] = temp
dado = [16,18,15,37,13]
ordenar(dado)
print(dado)
```

Com o uso desse algoritmo, qual é a quantidade de trocas realizadas para ordenar a sequência **dado**?
a) 4
b) 5
c) 6
d) 7
e) 8

Gabarito: **Letra C**.

Analisando o algoritmo, temos que ele percorre a coleção, comparando o de trás para frente e, se o item anterior for maior que o atual, troca os dois de posição. É uma variação do Bubble Sort, analisando de trás para frente. As trocas são feitas entre elementos consecutivos, até que a coleção esteja ordenada. Portanto:

INICIAL: [16,18,15,37,13]
1: TROCA 37,13 - [16,18,15,13,37]
2: TROCA 15,13 - [16,18,13,15,37]
3: TROCA 18,13 - [16,13,18,15,37]
4: TROCA 13,16 - [13,16,18,15,37]
5: TROCA 18,15 - [13,16,15,18,37]
6: TROCA 16,15 - **[13,15,16,18,37]**

Pronto! A coleção está organizada depois de 6 trocas. Portanto, **Letra C**.

Curiosidade: A linguagem utilizada na questão é Python.

10. (CESGRANRIO / Escriturário (BB) / 2018)

O programa a seguir, em Python, implementa o algoritmo do método de bolha, imprimindo o resultado de cada passo.

```
def bolha(lista):
    for passo in range(len(lista)-1,0,-1):
         for i in range(passo):
              if lista[i]>lista[i+1]:
                   lista[i],lista[i+1]=lista[i+1],lista[i]
         print(lista)
```

Qual será a quarta linha impressa para a chamada **bolha([4, 3, 1, 9, 8, 7, 2, 5])** ?

a) [3, 1, 4, 8, 7, 2, 5, 9]
b) [1, 3, 4, 7, 2, 5, 8, 9]
c) [1, 2, 3, 4, 5, 7, 8, 9]
d) [1, 3, 2, 4, 5, 7, 8, 9]
e) [1, 3, 4, 2, 5, 7, 8, 9]

Gabarito: **Letra D.**

Analisando iteração por iteração da função bolha (a cada iteração, mais uma posição ao final do array está correta, e não precisa mais ser testada):

```
# Iteração 1
  -  Começa com:  [4, 3, 1, 9, 8, 7, 2, 5]
   [4, 3, 1, 9, 8, 7, 2, 5]  - Troca  4  <->  3
   [3, 4, 1, 9, 8, 7, 2, 5]  - Troca  4  <->  1
   [3, 1, 4, 9, 8, 7, 2, 5]  - Mantem 4   -   9
   [3, 1, 4, 9, 8, 7, 2, 5]  - Troca  9  <->  8
   [3, 1, 4, 8, 9, 7, 2, 5]  - Troca  9  <->  7
   [3, 1, 4, 8, 7, 9, 2, 5]  - Troca  9  <->  2
   [3, 1, 4, 8, 7, 2, 9, 5]  - Troca  9  <->  5
  - Termina com:  [3, 1, 4, 8, 7, 2, 5, 9]
# Iteração 2
  -  Começa com:  [3, 1, 4, 8, 7, 2, 5, 9]
   [3, 1, 4, 8, 7, 2, 5, 9]  - Troca  3  <->  1
   [1, 3, 4, 8, 7, 2, 5, 9]  - Mantem 3   -   4
   [1, 3, 4, 8, 7, 2, 5, 9]  - Mantem 4   -   8
   [1, 3, 4, 8, 7, 2, 5, 9]  - Troca  8  <->  7
   [1, 3, 4, 7, 8, 2, 5, 9]  - Troca  8  <->  2
   [1, 3, 4, 7, 2, 8, 5, 9]  - Troca  8  <->  5
  - Termina com:  [1, 3, 4, 7, 2, 5, 8, 9]
# Iteração 3
  -  Começa com:  [1, 3, 4, 7, 2, 5, 8, 9]
   [1, 3, 4, 7, 2, 5, 8, 9]  - Mantem 1   -   3
   [1, 3, 4, 7, 2, 5, 8, 9]  - Mantem 3   -   4
   [1, 3, 4, 7, 2, 5, 8, 9]  - Mantem 4   -   7
   [1, 3, 4, 7, 2, 5, 8, 9]  - Troca  7  <->  2
   [1, 3, 4, 2, 7, 5, 8, 9]  - Troca  7  <->  5
  - Termina com:  [1, 3, 4, 2, 5, 7, 8, 9]
# Iteração 4
  -  Começa com:  [1, 3, 4, 2, 5, 7, 8, 9]
   [1, 3, 4, 2, 5, 7, 8, 9]  - Mantem 1   -   3
   [1, 3, 4, 2, 5, 7, 8, 9]  - Mantem 3   -   4
   [1, 3, 4, 2, 5, 7, 8, 9]  - Troca  4  <->  2
   [1, 3, 2, 4, 5, 7, 8, 9]  - Mantem 4   -   5
  - Termina com:  [1, 3, 2, 4, 5, 7, 8, 9] - Letra D.
```

11. (FCC / Analista Judiciário (TRF 5ª Região) / 2017 / Informática - Desenvolvimento / Apoio Especializado)

O algoritmo QuickSort usa uma técnica conhecida por divisão e conquista, onde problemas complexos são reduzidos em problemas menores para se tentar chegar a uma solução.

A complexidade média deste algoritmo em sua implementação padrão e a complexidade de pior caso são, respectivamente,

a) $O(n-1)$ e $O(n^3)$.
b) $O(n^2)$ e $O(n \log n^2)$.
c) $O(n^2)$ e $O(n^3)$.
d) $O(n)$ e $O(n^2)$.
e) $O(n \log n)$ e $O(n^2)$.

Gabarito: **LETRA E.**

ALGORITMO	MELHOR CASO	CASO MÉDIO	PIOR CASO
BubbleSort	$O(n)$	$O(n^2)$	$O(n^2)$
InsertionSort	$O(n)$	$O(n^2)$	$O(n^2)$
SelectionSort	$O(n^2)$	$O(n^2)$	$O(n^2)$
QuickSort	**$O(n \log n)$**	**$O(n \log n)$**	**$O(n^2)$**
ShellSort	$O(n \log n)$	Depende do *gap*	$O(n^2)$
MergeSort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
HeapSort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

12. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Para ordenar um vetor com N elementos, o método de ordenação Seleção (Selection Sort) faz o seguinte número de comparações:
a) $(N^2 - N)/2$, sendo muito lento e inadequado para valores grandes de N.
b) $\log_2(N^2 + N)$ no melhor caso.
c) $(N^2 + N - 1)/2$ no caso médio, ficando lento para valores grandes de N.
d) $(N - 1)$ quando o vetor já está originalmente ordenado.
e) $(N^2 + N)/4$ no pior caso, sendo melhor que o pior caso do Bolha (Bubble Sort) pois faz menos trocas.

Gabarito: **Letra A**

O Selection Sort percorre a parte desordenada do vetor selecionando o maior elemento (daí o nome *selection*), e posicionando no início da parte ordenada. Ele precisa fazer um número de comparações que depende do tamanho da parte desordenada do vetor.

Na primeira passagem, precisa fazer N-1 comparações. Na segunda, N-2. Na terceira, N-3, e assim sucessivamente até só fazer 1 comparação. Portanto, temos:

Comparações = (N-1) + (N-2) + ... + 3 + 2 + 1

É uma progressão aritmética de (N-1) elementos, portanto, a soma será o número de elementos (N-1), vezes média do primeiro elemento (N-1) com o último elemento (1):

Comparações = (N-1)*((N-1) + (1))/2 = (N-1)*(N)/2 = (N^2 – N)/2.

É uma complexidade quadrática e, portanto, inadequada para valores grandes de N.

13. (CESGRANRIO - 2010 – BACEN – Analista de Sistemas)

Uma fábrica de software foi contratada para desenvolver um produto de análise de riscos. Em determinada funcionalidade desse software, é necessário realizar a ordenação de um conjunto formado por muitos números inteiros. Que algoritmo de ordenação oferece melhor complexidade de tempo (Big O notation) no pior caso?

a) Merge sort
b) Insertion sort
c) Bubble sort
d) Quick sort
e) Selection sort

Gabarito: **Letra A**

Algoritmo	Melhor Caso	Caso Médio	Pior Caso
BubbleSort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
InsertionSort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
SelectionSort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
QuickSort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$
ShellSort	$\Omega(n \log n)$	Depende do *gap*	$O(n^2)$
MergeSort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$
HeapSort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$

14. (FGV - 2013 – MPE/MS – Analista de Sistemas)

Assinale a alternativa que indica o algoritmo de ordenação capaz de funcionar em tempo O(n) para alguns conjuntos de entrada.

a) Selectionsort (seleção)
b) Insertionsort (inserção)
c) Merge sort

d) Quicksort
e) Heapsort

Gabarito: **Letra B**.

Algoritmo	Melhor Caso	Caso Médio	Pior Caso
BubbleSort	O(n)	O(n²)	O(n²)
InsertionSort	O(n)	O(n²)	O(n²)
SelectionSort	O(n²)	O(n²)	O(n²)
QuickSort	O(n log n)	O(n log n)	O(n²)
ShellSort	O(n log n)	Depende do *gap*	O(n²)
MergeSort	O(n log n)	O(n log n)	O(n log n)
HeapSort	O(n log n)	O(n log n)	O(n log n)

15. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – III)

O tempo médio do QuickSort é de ordem igual ao tempo médio do MergeSort.

Gabarito: **CERTO**.

Algoritmo	Melhor Caso	Caso Médio	Pior Caso
BubbleSort	O(n)	O(n²)	O(n²)
InsertionSort	O(n)	O(n²)	O(n²)
SelectionSort	O(n²)	O(n²)	O(n²)
QuickSort	O(n log n)	O(n log n)	O(n²)
ShellSort	O(n log n)	Depende do *gap*	O(n²)
MergeSort	O(n log n)	O(n log n)	O(n log n)
HeapSort	O(n log n)	O(n log n)	O(n log n)

4 PESQUISA DE DADOS (BUSCA DE DADOS)

4.1 INTRODUÇÃO

Uma das tarefas de maior importância na computação é a pesquisa de informações contidas em coleções de dados. **Em geral, desejamos que essa tarefa seja executada sem que haja a necessidade de inspecionar toda a coleção de dados.** Existem algumas maneiras de realizar pesquisas, tais como: Pesquisa Sequencial, Pesquisa Binária, Tabelas de Dispersão (Hashing), Árvores AVL, Árvores B e Árvores B+.

4.2 BUSCA SEQUENCIAL

Galera, imaginem que eu estou à procura de um valor X em um vetor L[ ]! Para tal, posso inspecionar as posições sequenciais de L[ ] a partir da primeira posição: se eu encontrar X, minha busca tem êxito; se eu alcanço a última posição e não encontro X, concluímos que esse valor não ocorre no vetor L[ ]. *Como é chamada essa busca em que eu inspeciono uma estrutura posição por posição?* **Sequencial ou Linear.**

Considerando que o vetor L[ ] contém N elementos, ordenados ou não, é fácil verificar que a busca sequencial requer tempo linearmente proporcional ao tamanho do vetor, i.e., da ordem $O(n)$. Por conta disso, é comum dizer que a busca sequencial é uma Busca Linear. *Entenderam?* **Quanto maior o vetor, maior o tempo em média para buscar um elemento! Quanto mais ao final, mais demorado**.

A Busca Sequencial é muito lenta para grandes quantidades de dados, mas aceitável para listas pequenas e que mudam constantemente. Observa-se que no Melhor Caso, X está na primeira posição, logo necessita apenas de uma comparação; no Pior Caso, X está na última posição, logo necessita de N comparações; e no Caso Médio, X é encontrado após $(n+1)/2$ comparações.

A seguir, encontram-se dois algoritmos para realização de uma Busca Sequencial: o primeiro ocorre de forma simples e o segundo ocorre de forma recursiva. Galera, para vetores de médio ou grande porte, o tempo de busca sequencial é considerado completamente inaceitável, dado que existem técnicas mais eficientes! *Professor, me dá um exemplo?* Claro, veremos adiante: Busca Binária!

```
PROCEDIMENTO BUSCA_SEQUENCIAL ( L , X , POS )

   ENTRADA: UM VETOR L e UM VALOR X
   SAÍDA: POS = i, SE X OCORRE NA POSIÇÃO i DE L // SUCESSO
          POS = 0, CASO CONTRÁRIO.

   POS = 0

   PARA i = 1 até N
       SE L[i] = X
          POS = i
          ESCAPE
   // fim para
   DEVOLVA POS
FIM {BUSCA_SEQUENCIAL}
```

```
PROCEDIMENTO BUSCA_SEQUENCIAL_REC ( L , N, X )

   ENTRADA: UM VETOR L DE TAMANHO N e UM VALOR X
   SAÍDA: i, SE X OCORRE NA POSIÇÃO i DE L    // SUCESSO
          0, CASO CONTRÁRIO.

 SE N = 1
       SE L[1] = X
            DEVOLVA 1
       SENÃO
            DEVOLVA 0
         SENÃO
       SE L[N] = X
          DEVOLVA N
 SENÃO
       BUSCA_SEQUENCIAL_REC ( L , N-1, X )

FIM {BUSCA_SEQUENCIAL_REC}
```

4.3 BUSCA BINÁRIA

A Busca Binária é um algoritmo de busca em vetores que segue o paradigma de divisão-e-conquista. Parte-se do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca, comparando o elemento chave com o elemento do meio do vetor. Possui complexidade da ordem de $O(\log_2 n)$, em que N é o tamanho do vetor de busca.

Quando o Vetor L[ ] estiver em ordem crescente, podemos determinar se X ocorre em L[ ] de forma mais rápida da seguinte forma: inspeciona-se a posição central do vetor! Se essa posição já contiver X, a busca para! *Por que, professor?* **Porque nós já encontramos X! Se X for menor que esse elemento central, passamos a procurar X, recursivamente, no intervalo de L[ ] que se encontra à esquerda da posição central.**

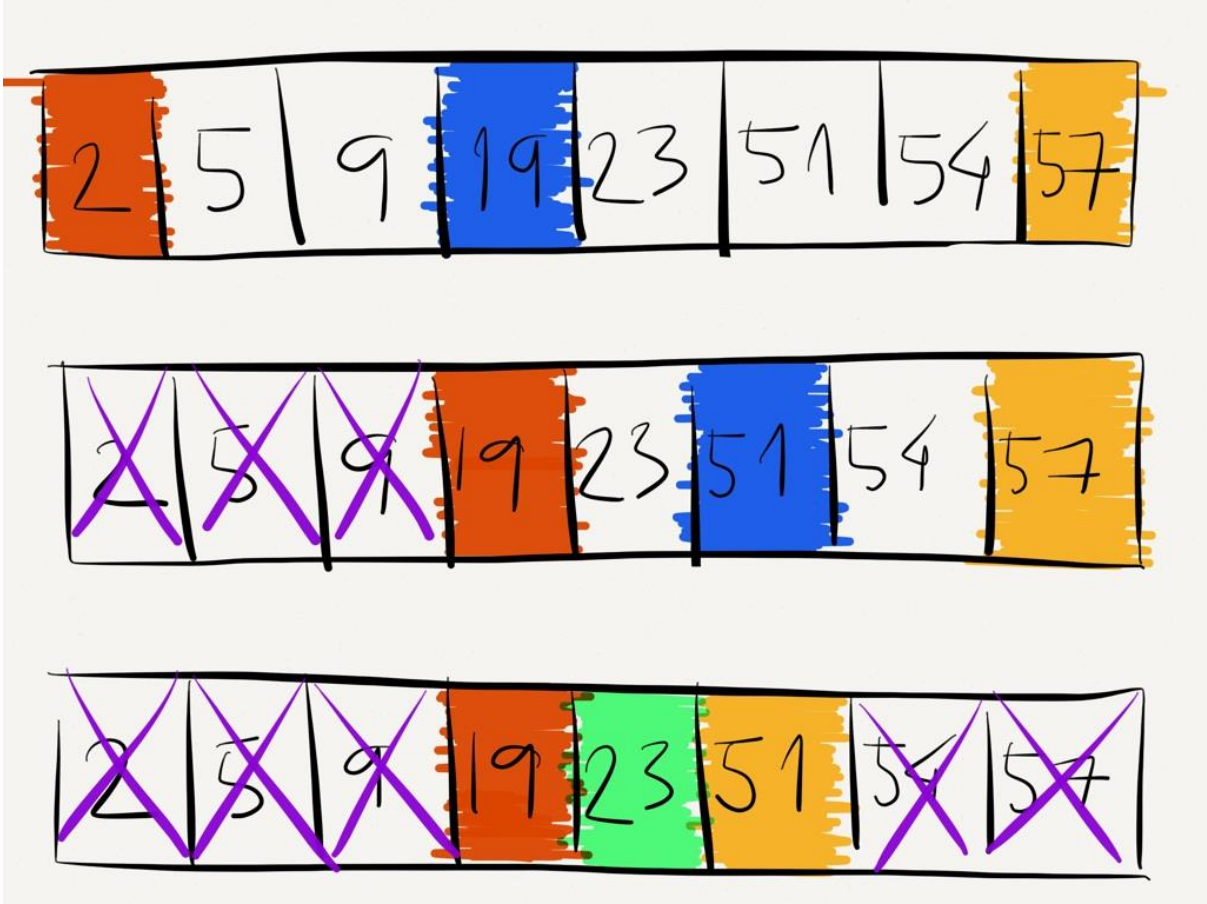
Se X for maior do que o elemento central, continuamos a procurar X, recursivamente, no intervalo de L que está à direita da posição central. **Se o intervalo se tornar vazio, a busca para, tendo sido malsucedida.** Esse procedimento é conhecido como Busca Binária e, facilmente, pode-se adaptar a busca em ordem decrescente. Segue abaixo um possível algoritmo:

```
PROCEDIMENTO BUSCA_BINARIA
  ENTRADA: UM VETOR L EM ORDEM CRESCENTE, UM VALOR X, E AS
           POSIÇÕES INICIO E FIM.
  SAÍDA: SIM, SE X OCORRE ENTRE AS POSIÇÕES INICIO E FIM DE
         L; NÃO, CASO CONTRÁRIO.

  SE INICIO > FIM
    DEVOLVA NÃO E PARE
  MEIO = (INICIO+FIM)/2                // DIVISÃO INTEIRA
  SE X = L[MEIO]
    DEVOLVA SIM E PARE
  SE X < L[MEIO]
    DEVOLVA BUSCA_BINARIA(L, X, INICIO, MEIO - 1)
  SENAO
    DEVOLVA BUSCA_BINARIA(L, X, MEIO + 1, FIM)
FIM {BUSCA_BINARIA}
```

Na imagem abaixo, estamos à procura do valor 23! Em vermelho, encontra-se o elemento inicial L[0] = 2 e, em amarelo, encontra-se o elemento final L[N-1] = 57. Procuramos, então, o elemento central! *Como?* Ele é o elemento de índice [0 + (N-1)]/2 = 7/2 = 3,5 = 3 (Arredonda-se para baixo). Ora, L[3] = 19! *Encontramos?* Não, 23>19! Sendo assim, L[0] = 19 e L[4] = 57.

Procuramos, então, o elemento central! *Como?* Ele é o elemento de índice [0 + (N-1)]/2 = 4/2 = 2. Ora, L[2] = 51! *Encontramos?* Não, 23<51! Sendo assim, L[0] = 19 e L[2] = 51. Procuramos, então, o elemento central! *Como?* Ele é o elemento de índice [0+ (N-1)]/2 = 2/2 = 1. Ora, L[1] = 23! *Encontramos?* Sim! Então, nossa busca obteve êxito e encontramos o que buscávamos.



4.4 EXERCÍCIOS COMENTADOS: PESQUISA

1. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações / Análise de Sistemas)

Para poder ser aplicado, o algoritmo de pesquisa binária exige que os elementos do *array*:
a) sejam números;
b) estejam ordenados;
c) estejam representados em base múltipla de 2;
d) ocupem somente as posições pares;
e) não sejam repetidos.

Gabarito: **Letra B**.

O método de busca binária consiste em acessar o elemento no meio do array, e testar se é o elemento procurado. Se não for o elemento procurado, repete a operação no lado com elementos menores (se o elemento procurado for menor que o elemento no meio), ou maiores (caso seja maior).

Ora, para que exista um lado com elementos menores e maiores, é preciso que o array esteja ordenado. Portanto, **a busca binária só pode ser feita sobre vetores ordenados**!

Por isso, resposta correta, **Letra B**. Todas as outras condições não são limitações do método.

2. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

Considere que um algoritmo de pesquisa, em um arquivo previamente ordenado, é caracterizado por realizar comparação de chaves e sucessivas divisões no espaço de busca até encontrar o termo pesquisado ou até haver um único registro. Trata-se de um algoritmo de

a) pesquisa por interpolação.
b) pesquisa binária.
c) pesquisa sequencial.
d) árvore de busca binária.

Gabarito: **Letra B**.

a) pesquisa por interpolação.
ERRADO. A pesquisa por interpolação é uma melhoria feita no método de busca binária, quando a coleção, além de ordenada, tem os elementos com separados por uma diferença constante (ou quase constante) - por exemplo, um elemento é sempre 10 unidades maior que o anterior. Nesse caso, após 2 consultas é possível "adivinhar" a posição do elemento procurado.

b) pesquisa binária.
CERTO. A busca binária é feita sobre uma coleção ordenada, buscando o elemento no meio (pivô), e testando contra o elemento procurado. Se o elemento não for igual, procura na sub-coleção com elementos menores que o pivô, se for maior, procura na sub-coleção com elementos maiores.

c) pesquisa sequencial.
ERRADO. A pesquisa sequencial testa o valor procurado elemento por elemento, em ordem sequencial. A única vantagem sobre a busca binária é que funciona sobre coleções não-ordenadas.

d) árvore de busca binária.
ERRADO. A árvore de busca binária é uma estrutura de dados, não um algoritmo de pesquisa.

Portanto, a **Letra B** é a alternativa correta.

3. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Considere que na Defensoria há uma lista ordenada com o nome de 1000 cidadãos amazonenses. Utilizando o método de pesquisa binária para localizar o nome de um destes cidadãos, serão necessárias, no máximo,
a) 1.000 comparações.
b) 10 comparações.
c) 500 comparações.
d) 200 comparações.
e) 5 comparações.

Gabarito: **Letra B**

A busca binária busca no meio da lista ordenada. Se for menor, repete na sub-lista à esquerda, se for maior, na sub-lista à direita. Como a lista é sempre repartida pela metade, é efetuada, no pior caso, o primeiro número inteiro maior ou igual a log2(n). Como log2(1.000) = 9,966, teremos, no máximo, 10 comparações.

Eu acho mais fácil pensar na potência de 2 que é ligeiramente maior que o número de elementos. No caso, 2^10 = 1024 é a primeira potência de 2 maior que 1.000, portanto, são necessárias, no máximo, **10 comparações**.

4. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

Na pesquisa do tipo sequencial, há aumento do desempenho se a tabela estiver ordenada pelo valor da chave.

Gabarito: **CERTO**, mas MUITO polêmico...

Uma pesquisa sequencial, por padrão, independe da ordenação da coleção (no caso, da tabela). O algoritmo passa item por item sequencialmente testando a igualdade. Ela pode ser realizada em coleções desordenada, ou ordenadas. Portanto, **por essa análise, não há aumento do desempenho pela ordenação**.

Contudo, **é POSSÍVEL optimizar a busca caso a coleção estiver ordenada**: uma vez que eu encontre um valor maior do que o item que eu procuro, posso assumir que o item procurado não está presente na coleção.

Na minha opinião, descuido do examinador e da banca.

Reescrevendo de forma melhor:

Na pesquisa do tipo sequencial, ~~**há**~~ **é possível o** aumento do desempenho se a tabela estiver ordenada pelo valor da chave.

5. (CESGRANRIO / Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Infraestrutura)

Um método que implementa um algoritmo de busca binária recebe como parâmetros um vetor de inteiros ordenados descendentemente, o comprimento desse vetor e um número inteiro que se deseja localizar no vetor. O cabeçalho desse método é o seguinte:

public int buscaBin(int vet[], int n, int val)

Admitindo-se que o vetor passado como parâmetro tenha 750 elementos, qual será o número máximo de iterações que o algoritmo irá realizar até que o valor (val) seja localizado ou que seja detectado que esse valor não se encontra no vetor?

a) 8
b) 9
c) 10
d) 11
e) 12

Gabarito: **Letra C**.

O jeito "decorado" é saber que o número de iterações para buscar com a busca binária é $\log_2$ (N), arredondado para cima. Como N é 750, o resultado será 9,xxx. Arredondado para cima, temos 10.

Mas é fácil resolver mesmo sem fórmula!

A busca binária só ocorre sobre listas ordenadas. Esse método busca o elemento procurado no meio da estrutura e, caso seja menor, repete a busca na sub-lista com elementos menores, e caso seja maior, repete na sub-lista com elementos maiores. Em ambos os casos, o universo de elementos pesquisados sempre cai pela metade depois de um teste.

Sendo assim, no pior cenário, faríamos o 1o. teste, dividindo a lista em 374 elementos à esquerda, 1 testado, e 375 elementos à direita. Como é o pior cenário, não teríamos encontrado, e teríamos de procurar do lado maior com 375 elementos.

Novamente, procuraríamos entre 375, sendo 187 a esquerda, 1 testado, e 187 à direita. Não encontraríamos e teríamos que procurar em 187...

Em cada passo dividimos o universo na metade. Resumindo:
1a. iteração - sobram 375
2a. iteração - sobram 187
3a. iteração - sobram 93
4a. iteração - sobram 46
5a. iteração - sobram 23
6a. iteração - sobram 11
7a. iteração - sobram 5
8a. iteração - sobram 2
9a. iteração - sobram 1
10a. iteração - sobra 0

Após 10 iterações sabemos se o elemento está ou não na lista, portanto **Letra C**.

6. (CESGRANRIO / Escriturário (BB) / 2018)

Deseja-se realizar uma busca sobre um vetor não ordenado de inteiros. Para tal, deve-se criar um método Java que receba como parâmetros o vetor em questão e um número inteiro (elemento) que se deseja procurar no vetor, além de outros parâmetros que se julgarem necessários. Essa função deve retornar

• o índice do elemento no vetor, caso ele seja encontrado;
• o inteiro -1, caso o elemento não seja encontrado.

Assumindo-se que todos os pacotes necessários foram devidamente importados, qual método Java irá realizar corretamente essa busca?

a)

```
static int busca(int[] vet, int elem, int ini, int fim) {
        if (ini > fim)
                return -1;

        int m = (ini + fim) / 2;
        if (elem == vet[m])
                return m;
        else
                if (elem > vet[m])
                        return busca(vet, elem, m + 1, fim);
                else
                        return busca(vet, elem, ini, m - 1);
```

---

b)

```
public static int busca(int vet[], int elem) {
        for(int i=0; i < vet.length; i++)
                if(elem == vet[i])
                        return i;
                else
                        if(elem < vet[i])
                                return -1;

        return -1;
```

---

c)

```
public static int busca(int[] vet, int elem) {
        int ini = 0, fim = vet.length-1;

        while(ini <= fim) {
                int m = (ini + fim) / 2;
                if (elem == vet[m])
                        return m;
                else
                        if (elem > vet[m])
                                ini = m + 1;
                        else
                                fim = m - 1;
        }

        return -1;
```

---

d)

```
public static int busca(int vet[],int elem) {
        if(vet.length==0)
                return -1;

        if(elem==vet[vet.length-1])
                return vet.length-1;

        return busca(Arrays.copyOf(vet,vet.length-1),elem);
```

---

e)

```
public static int busca(int vet[], int elem) {
    if(vet.length == 0)
            return -1;

    if(elem == vet[vet.length-1])
            return vet.length-1;
    else
            if(elem > vet[vet.length-1])
                    return -1;

    return busca(Arrays.copyOf(vet, vet.length-1), elem);
```

Gabarito: **Letra D**.

Analisando item por item:

Letra A) **ERRADO**. O algoritmo faz uma busca binária recursiva - Compara o elemento procurado com o elemento no meio. Se for menor, chama novamente o método para a sub-coleção na esquerda, se for maior, na direita.

Letra B) **ERRADO**. O algoritmo faz uma busca sequencial, mas considera a coleção ordenada, e para caso o elemento procurado seja menor que o elemento comparado.

Letra C) **ERRADO**. O algoritmo faz uma busca binária ajustando os intervalos ini e fim.

Letra D) **CERTO**. O algoritmo faz uma busca sequencial recursiva, de trás para frente, comparando o elemento procurado com o último elemento do array. Caso não encontre, chama novamente o método, passando um novo array sem a última posição (o que vai fazer com que se compare com o penúltimo, e assim sucessivamente).

Letra E) **ERRADO**. É semelhante ao algoritmo da Letra D, mas considera a coleção ordenada, pois se o elemento procurado for maior que o último, assume que a coleção não contém o elemento.

7. (CESGRANRIO / Engenheiro (PETROBRAS) / 2018 / Eletrônica / Equipamentos Júnior)

A função a seguir implementa um algoritmo de busca binária sobre um vetor de inteiros ordenado de modo ascendente.

```
int busca(int vet[], int elem, int ini, int fim) {
int m;
if(fim < ini)
  return -1;

m=(ini + fim) / 2;
```

```
System.out.println(vet[m]);

if(vet[m] == elem)
      return m;

if(vet[m] > elem)
        return busca(vet, elem, ini, m-1);

return busca(vet, elem, m+1, fim);
}
```

Essa função recebe como parâmetros um vetor (vet), o elemento que se deseja procurar no vetor (elem), o índice do primeiro elemento do vetor (ini) e o índice do último elemento do vetor (fim).

O comando System.out.println(vet[m]) exibe no console o valor do elemento de índice **m** do vetor **vet**.

Seja o seguinte vetor (vt) de inteiros:

20	25	27	38	51	57	60	65	73	74	78	80	83	88	90
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Suponha que a função busca seja chamada por meio do seguinte comando:

busca(vt, 39, 0, 14);

Qual será o 3o valor exibido no console?
a) 65
b) 51
c) 57
d) 38
e) 27

Gabarito: **Letra C**.

A função sempre imprime o valor no meio da sequência procurada, portanto basta verificar o elemento no meio na 3a. iteração. Assim:

Inicial: [20,25,27,38,51,57,60,65,73,74,78,80,83,88,90] - 15 posições

Iteração 1:
meio: 65
Como 39 é menor que 65, procura nos menores:
menores: [20,25,27,38,51,57,60]

Iteração 2:
meio: 38
Como 39 é maior que 38, procura nos maiores:

menores: [51,57,60]
Iteração 3:
meio: 57

Pronto! O elemento no meio, na 3a. iteração será o 57, **Letra C**.

8. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

Julgue o item a seguir, relativo aos conceitos de construção de algoritmos.

O algoritmo a seguir apresenta um exemplo de busca sequencial.

```
var a:vetor[1..5] de inteiro;
    temp:inteiro;
    i,k:inteiro;
início
    a[1]:=57;
    a[2]:=58;
    a[3]:=60;
    a[4]:=56;
    a[5]:=59;
    para i:=1 até 4 faça início
        para k:=i+1 até 5 faça início
            se a[i] > a[k] então início
               temp:=a[i];
               a[i]:=a[k];
               a[k]:=temp;
            fim se;
        fim para;
    fim para;
    escreva('Resultado: ');
    para i:=1 até 5 faça início
        escreva(a[i],',');
    fim para;
fim;
```

Gabarito: **ERRADO**.

Existem diversas evidências que contrariam a declaração de que esse seja um algoritmo de busca sequencial:

- Só são necessárias 2 variáveis para a busca sequencial: o vetor, e o elemento procurado. O algoritmo declara o vetor, e mais 3 variáveis de controle.
- Só é necessário percorrer o vetor 1 vez na busca sequencial (complexidade O(n)), e existem 2 loops aninhados no algoritmo ($O(n^2)$).
- Só é necessário apresentar 1 valor como resposta - seja o elemento, ou o índice do elemento procurado. O algoritmo imprime o vetor inteiro como resposta.

Por isso, rapidamente é possível dizer que o item está **ERRADO**.

Se quiser ir mais fundo...
Perceba que o que está acontecendo no algoritmo é o teste do valor em cada posição do array com as posições seguintes do array e, se estiverem fora da ordem crescente, trocam a posição. Trata-se de um **algoritmo de ordenação** muito semelhante ao Bubble Sort, conhecido como **Exchange Sort.**

9. (CESPE / Agente Controlador de Arrecadação (AL) / 2002)

Os dados que são tratados por um sistema de computação podem possuir diferentes tipos de organização natural. Para lidar com essa diversidade de organizações naturais, diferentes tipos de estruturas de dados podem ser utilizados.

Acerca dos principais tipos de estruturas de dados, julgue o item subseqüente.

Operações de busca em uma árvore binária envolvem, no mínimo, N/2 operações de comparação, em que N é o número de elementos contidos na árvore binária.

Gabarito: **Errado**

A quantidade de comparações máxima em uma árvore binária $\log_2 N$, desde que ela esteja balanceada. Contudo, não há quantidade mínima! Se você der a sorte de procurar exatamente o valor que está no nó raiz, só acontecerá 1 teste!

RELEMBRANDO CONCEITOS DE ÁRVORES BINÁRIAS!

A árvore é uma estrutura recursiva, formada por um nó, uma sub-árvore esquerda, e outra sub-árvore à direita. A árvore binária tem todos os elementos menores que o nó na sub-árvore esquerda, e os maiores na sub-árvore à direita. A árvore binária é dita balanceada se, para qualquer nó da árvore, a sub-árvore à direita não tenha a altura diferente da sub-árvore à esquerda em mais de 1 unidade.

10. (FCC / Analista (DPE RS) / 2017 / Desenvolvimento de Sistemas / Tecnologia da Informação)

Um Analista, estudando a complexidade de algoritmos de busca linear (ou sequencial), concluiu corretamente que no pior caso, considerando um vetor de n elementos, este tipo de algoritmo tem complexidade

a) O(n).
b) O(log_2n-1).
c) O(√n).
d) O(log_2n).
e) O(log_2n^2).

Gabarito: **Letra A**.

O pior caso de uma busca linear em um vetor de *n* elementos é quando o elemento procurado é o último. Nesse caso, serão realizados N testes e, portanto, temos a complexidade O(n). Portanto, **Letra A** está correta.

Lembrando que no melhor caso da busca linear, o primeiro elemento testado já é o procurado, e só é realizado 1 teste. No caso médio, são realizados *n/2* testes (a complexidade permanece O(n)).

11. (FGV / Analista do Ministério Público (MPE AL) / 2018 / / Desenvolvimento de Sistemas)

A complexidade do algoritmo de busca binária, sobre uma lista indexada ordenada pela chave de busca, é

a) $O(N)$
b) $O(\log_2 N)$
c) $O(2 \log_2 N)$
d) $O(N \log_2 N)$
e) $O(N^2)$

Gabarito: **Letra B.**

Pessoal, cada iteração da busca binária reparte a lista na metade. No final de x iterações, teremos testado 2^x elementos. Se $n = 2^x$, $x = \log_2 n$. Mas é mais fácil decorar:

- complexidade da busca binária: O(log N)
- complexidade da busca sequencial: O(N)

5 EXERCÍCIOS COMENTADOS

5.1 EXERCÍCIOS COMENTADOS: INTRODUÇÃO

1. (FGV – 2015 – DPE/MT – Analista de Sistemas)

No desenvolvimento de sistemas, a escolha de estruturas de dados em memória é especialmente relevante. Dentre outras classificações, é possível agrupar essas estruturas em lineares e não lineares, conforme a quantidade de sucessores e antecessores que os elementos da estrutura possam ter. Assinale a opção que apresenta, respectivamente, estruturas de dados lineares e não lineares.

a) Tabela de dispersão e fila.
b) Estrutura de seleção e pilha.
c) Pilha e estrutura de seleção.
d) Pilha e árvore binária de busca.
e) Fila e pilha.

Gabarito: D

Além dessa classificação, existe outra também importante: Estruturas Lineares e Estruturas Não-Lineares. As Estruturas Lineares são aquelas em que cada elemento pode ter um único predecessor (exceto o primeiro elemento) e um único sucessor (exceto o último elemento). Como exemplo, podemos citar Listas, Pilhas, Filas, Arranjos, entre outros.

Já as Estruturas Não-Lineares são aquelas em que cada elemento pode ter mais de um predecessor e/ou mais de um sucessor. Como exemplo, podemos citar Árvores, Grafos e Tabelas de Dispersão. Essa é uma classificação muito importante e muito simples de entender. Pessoal, vocês perceberão que esse assunto é cobrado de maneira superficial na maioria das questões, mas algumas são nível doutorado!

Conforme vimos em aula, trata-se de pilha e árvore respectivamente.

2. (CESPE – 2010 – DETRAN/ES – Analista de Sistemas)

Um tipo abstrato de dados apresenta uma parte destinada à implementação e outra à especificação. Na primeira, são descritas, em forma sintática e semântica, as operações que podem ser realizadas; na segunda, os objetos e as operações são representados por meio de representação, operação e inicialização.

Comentários:

Gabarito: ERRADO

A Especificação Sintática define o nome do tipo, suas operações e o tipo dos argumentos das operações, definindo a assinatura do TAD. A Especificação Semântica descreve propriedades e efeitos das operações de forma independente de uma implementação específica. E a Especificação de Restrições estabelece as condições que devem ser satisfeitas antes e depois da aplicação das operações.

Conforme vimos em aula, temos duas partes: Especificação e Implementação. Sendo que a especificação se divide em Especificação Sintática e Semântica e Implementação se divide em Representação e Algoritmos. Logo, a questão está invertida: na segunda, são descritas, em forma sintática e semântica, as operações que podem ser realizadas; na primeira, os objetos e as operações são representados por meio de representação, operação e inicialização.

3. (CESPE – 2010 – TRT/RN – Analista de Sistemas)

O tipo abstrato de dados consiste em um modelo matemático (v,o), em que v é um conjunto de valores e o é um conjunto de operações que podem ser realizadas sobre valores.

Gabarito: CERTO

Por fim, vamos falar sobre Tipos Abstratos de Dados (TAD). Podemos defini-lo como um modelo matemático (v,o), em que v é um conjunto de valores e o é um conjunto de operações que podem ser realizadas sobre valores. Eu sei, essa definição é horrível de entender! Para compreender esse conceito, basta

lembrar de abstração, i.e., apresentar interfaces e esconder detalhes.

Conforme vimos em aula, a questão está conforme nós descrevemos em nossa aula. Essa é a definição formal de Tipos Abstratos de Dados.

4. (CESPE – 2010 – BASA – Analista de Sistemas)

A escolha de estruturas internas de dados utilizados por um programa pode ser organizada a partir de TADs que definem classes de objetos com características distintas.

Gabarito: ERRADO

Em utras palavras, o nível semântico trata do comportamento de um tipo abstrato de dados; e o nível sintático trata da apresentação de um tipo abstrato de dados. Podemos dizer, então, que o TAD encapsula uma estrutura de dados com características semelhantes – podendo ser formado por outros TADs –, e esconde a efetiva implementação dessa estrutura de quem a manipula.

Conforme vimos em aula, definem classes de objetos com características semelhantes.

5. (CESPE – 2010 – BASA – Analista de Sistemas)

A descrição dos parâmetros das operações e os efeitos da ativação das operações representam, respectivamente, os níveis sintático e semântico em que ocorre a especificação dos TDAs.

Gabarito: CERTO

A Especificação Sintática define o nome do tipo, suas operações e o tipo dos argumentos das operações, definindo a assinatura do TAD. A Especificação Semântica descreve propriedades e efeitos

das operações de forma independente de uma implementação específica. E a Especificação de Restrições estabelece as condições que devem ser satisfeitas antes e depois da aplicação das operações.

Conforme vimos em aula, realmente se trata respectivamente do nível sintático e semântico.

6. (FCC – 2010 – TRE/AM – Analista de Sistemas)
Em relação aos tipos abstratos de dados - TAD, é correto afirmar:

a) O TAD não encapsula a estrutura de dados para permitir que os usuários possam ter acesso a todas as operações sobre esses dados.

b) Na transferência de dados de uma pilha para outra, não é necessário saber como a pilha é efetivamente implementada.

c) Alterações na implementação de um TAD implicam em alterações em seu uso.

d) Um programador pode alterar os dados armazenados, mesmo que não tenha conhecimento de sua implementação.

e) TAD é um tipo de dados que esconde a sua implementação de quem o manipula.

Gabarito: E

Em outras palavras, o nível semântico trata do comportamento de um tipo abstrato de dados; e o nível sintático trata da apresentação de um tipo abstrato de dados. Podemos dizer, então, que o TAD encapsula uma estrutura de dados com características semelhantes – podendo ser formado por outros TADs –, e esconde a efetiva implementação dessa estrutura de quem a manipula.

(a) Errado, ele encapsula a estrutura de dados; (b) Correto, não é necessário saber a implementação, porém a FCC marcou o gabarito como errado; (c) Errado, a implementação pode mudar livremente; (d) Errado, sem conhecimento da implementação, ele não poderá alterar dados armazenados; (e) Correto, esse item também está correto (Lembrem-se: Na FCC, muitas vezes tempos que marcar a mais correta ou a menos errada – infelizmente!).

7. (FCC – 2009 – TRE/PI – Analista de Sistemas)
Em relação a tipos abstratos de dados, é correto afirmar que:

a) o TAD não encapsula a estrutura de dados para permitir que os usuários possam ter acesso a todas as operações disponibilizadas sobre esses dados.

b) algumas pilhas admitem serem declaradas como tipos abstratos de dados.

c) filas não permitem declaração como tipos abstratos de dados.

d) os tipos abstratos de dados podem ser formados pela união de tipos de dados primitivos, mas não por outros tipos abstratos de dados.

e) são tipos de dados que escondem a sua implementação de quem o manipula; de maneira geral as operações sobre estes dados são executadas sem que se saiba como isso é feito.

Gabarito: E

Em outras palavras, o nível semântico trata do comportamento de um tipo abstrato de dados; e o nível sintático trata da apresentação de um tipo abstrato de dados. Podemos dizer, então, que o TAD encapsula uma estrutura de dados com características semelhantes – podendo ser formado por outros TADs –, e esconde a efetiva implementação dessa estrutura de quem a manipula.

(a) Errado. Pelo contrário, ele encapsula a estrutura de dados de modo que o usuário não tem acesso a implementação das operações; (b) Correto. Todas elas admitem, porém a FCC marcou o gabarito como errado; (c) Errado. Elas permitem a declaração como tipos abstratos de dados; (d) Errado. Um TAD pode ser formado por outros TADs; (e) Correto. Escondem a implementação de quem os manipula.

ACERTEI	ERREI

5.2 EXERCÍCIOS COMENTADOS: VETORES E MATRIZES

1. (FCC - 2009 - TJ-PA - Analista Judiciário - Tecnologia da Informação)
Considere uma estrutura de dados do tipo vetor. Com respeito a tal estrutura, é correto que seus componentes são, caracteristicamente,

a) heterogêneos e com acesso FIFO.
b) heterogêneos e com acesso LIFO.
c) heterogêneos e com acesso indexado-sequencial.
d) homogêneos e acesso não indexado.
e) homogêneos e de acesso aleatório por intermédio de índices.

Gabarito: E

Vetores possuem componentes homogêneos, i.e., todos os dados são de apenas um único tipo básico de dados. Ademais, seu acesso é aleatório por meio de índices! Bem, seria mais correto dizer que seu acesso é direto.

2. (CETAP - 2010 - AL-RR - Analista de Sistemas)

Matrizes são estruturas de dados de n-dimensões. Por simplicidade, chamaremos de matrizes as matrizes bidimensionais numéricas (que armazenam números inteiros). Sendo assim, marque a alternativa INCORRETA.

a) Uma matriz de m linhas e n colunas contêm (m * n) dados.

b) Uma matriz pode ser representada utilizando listas ligadas.

c) A soma dos elementos de uma matriz pode ser calculada fazendo dois laços aninhados, um sobre as linhas e o outro sobre as colunas.

d) A soma de duas matrizes de m linhas e n colunas resulta em uma matriz de 2*m linhas e 2*n colunas.

e) O produto de duas matrizes de n linhas e n colunas resulta em uma matriz de n linhas e n colunas.

Gabarito: D

(a) Perfeito, são M x N colunas; (b) Perfeito, podem ser utilizadas listas ligadas; (c) Perfeito, um laço para as linhas e outro para as colunas; (d) Não, a soma de uma Matriz 3x5 com outra Matriz 3x5 resulta em uma Matriz 3x5; (e) Perfeito, uma Matriz 2x2 multiplicada por outra Matriz 2x2 resulta em uma Matriz 2x2.

3. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Arquitetura de Tecnologia)

Os dados armazenados em uma estrutura do tipo matriz não podem ser acessados de maneira aleatória. Portanto, usa-se normalmente uma matriz quando o volume de inserção e remoção de dados é maior que o volume de leitura dos elementos armazenados.

Gabarito: E

Podem, sim, ser acessados de maneira aleatória ou direta, por meio de seus índices. Ademais, usa-se normalmente uma matriz quando o volume de leitura de elementos armazenados é maior que o volume de inserção e remoção de dados. Ora, é possível fazer acesso direto, logo é eficiente mesmo com alto volume de leitura.

4. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação)

Entre alguns tipos de estrutura de dados, podem ser citados os vetores, as pilhas e as filas.

Gabarito: CERTO

Questão extremamente simples – realmente são exemplos de estruturas de dados.

5. (CESPE - 2011 - EBC - Analista - Engenharia de Software)

Vetores são utilizados quando estruturas indexadas necessitam de mais que um índice para identificar um de seus elementos.

Gabarito: ERRADO

Não! Se são necessários mais de um índice, utilizam-se Matrizes! Vetores necessitam apenas de um índice.

6. (CESPE - 2010 - TRE-BA - Analista Judiciário - Análise de Sistemas)

Vetores podem ser considerados como listas de informações armazenadas em posição contígua na memória.

Gabarito: CERTO

Perfeito! Vetores podem ser considerados como listas de informações? Sim! As informações (dados) são armazenadas em posição contígua na memória? Sim, em geral são armazenados de forma contígua.

7. (CESPE - 2010 - TRE-BA - Analista Judiciário - Análise de Sistemas)

Uma posição específica de um vetor pode ser acessada diretamente por meio de seu índice.

Gabarito: CERTO

Perfeito! Observem que vetores são diferentes de listas, nesse sentido. Eu posso acessar qualquer elemento diretamente por meio de seu índice.

8. (FCC - 2016 - Copergás - PE - Analista Tecnologia da Informação)

Considere o algoritmo a seguir, na forma de pseudocódigo:

```
Var n, i, j, k, x: inteiro
Var v: vetor[0..7] inteiro
Início
 v[0] ← 12
 v[1] ← 145
 v[2] ← 1
 v[3] ← 3
 v[4] ← 67
 v[5] ← 9
 v[6] ← 45
 n ← 8
 k ← 3
 x ← 0

 Para j ← n-1 até k passo -1 faça
```

```
  v[j] ← v[j - 1];
 Fim_para

 v[k] ← x;
Fim
```

Este pseudocódigo

a) exclui o valor contido na posição x do vetor v.
b) insere o valor de x entre v[k-1] e v[k] no vetor v.
c) exclui o valor contido na posição k do vetor v.
d) tentará, em algum momento, acessar uma posição que não existe no vetor.
e) insere o valor de k entre v[x] e v[x+1] no vetor v.

Gabarito: B

O algoritmo, na estrutra de repetição "`Para j ← n-1 até k passo -1 faça`" percorre o vetor de trás para frente, "empurrando" os valores para o final do vetor. De fato, a estrutura vai da posição 7 até a 3 no vetor passando os valores da posição anterior para a posterior. Desse modo v[6] vai para v[7], v[5] para v[6], até que o valor de v[3] fica igual ao valor de v[4]. Na última instrução v[3] recebe outro valor. Ou seja, o algoritmo "afasta" os valores, a partir da posição três, para incluir um novo valor, entre v[2] e v[3] (v[k-1] e v[k]).

ACERTEI	ERREI

5.3 EXERCÍCIOS COMENTADOS: LISTAS

1. (CESPE –Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação)

Considere uma estrutura de dados em que cada elemento armazenado apresenta ligações de apontamento com seu sucessor e com o seu predecessor, o que possibilita que ela seja percorrida em qualquer sentido. Trata-se de
a) uma fila.
b) um grafo.
c) uma lista duplamente encadeada.
d) uma pilha.

Gabarito: **Letra C**.
a) uma fila.
ERRADO. Filas só precisam manter uma referência para o primeiro elemento, e que cada elemento conheça o sucessor. Não é necessário que um elemento conheça seu predecessor.

b) um grafo.
ERRADO. Grafos não têm conceito predecessor e sucessor, pois não são elementos lineares. Um grafo tem nós, que podem ser ligar uns aos outros, sem restrição quantidade de ligações.

c) uma lista duplamente encadeada.
CERTO. Listas são estruturas lineares. Listas duplamente encadeadas mantêm referencias em ambos sentidos.

d) uma pilha.

ERRADO. Pilhas são estruturas lineares, mas também só precisam conhecer seu topo, e que cada elemento conheça o elemento de baixo. Não é necessária a navegação nos dois sentidos.

Portanto, **Letra C** está correta.

2. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

O tempo de busca de um elemento em uma lista duplamente encadeada é igual à metade do tempo da busca de um elemento em uma lista simplesmente encadeada.

Gabarito: ERRADO

Não! Apesar de permitir que se percorra a lista em ambas as direções, em média ambas possuem o mesmo tempo de busca de um elemento.

3. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

Em algumas implementações, uma lista vazia pode ter um único nó, chamado de sentinela, nó cabeça ou header. Entre suas possíveis funções, inclui-se simplificar a implementação de algumas operações realizadas sobre a lista, como inserir novos dados, recuperar o tamanho da lista, entre outras.

Gabarito: CERTO

Perfeito! Ele simplifica a implementação de algumas operações porque se guarda o endereço do primeiro e do último elemento de uma estrutura de dados de modo que o programador não precisa conhecer a estrutura de implementação da lista para realizar suas operações.

4. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

Estruturas ligadas como listas encadeadas superam a limitação das matrizes que não podem alterar seu tamanho inicial.

Gabarito: C

Perfeito! Listas Encadeadas admitem alocação dinâmica, em contraste com as matrizes.

5. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

As listas duplamente encadeadas diferenciam-se das listas simplesmente encadeadas pelo fato de, na primeira, os nós da lista formarem um anel com o último elemento ligado ao primeiro da lista.

Gabarito: ERRADO

Não, a diferença é que Listas Duplamente Encadeadas possuem dois ponteiros, que apontam para o nó sucessor e para o nó predecessor e Listas Simplesmente Encadeadas possuem apenas um ponteiro, que aponta para o nó sucessor.

6. (FCC - 2012 - TRE-SP - Analista Judiciário - Análise de Sistemas - E)

Numa lista singularmente encadeada, para acessar o último nodo é necessário partir do primeiro e ir seguindo os campos de ligação até chegar ao final da lista.

Gabarito: CERTO

Perfeito! Se é uma lista singularmente encadeada, é necessário percorrer cada elemento um-a-um até chegar ao final da lista.

7. (CESPE - 2011 - EBC - Analista - Engenharia de Software)

Uma lista é uma coleção de elementos do mesmo tipo dispostos linearmente, que podem ou não seguir determinada organização. As listas podem ser dos seguintes tipos: de encadeamento simples, duplamente encadeadas e ordenadas.

Gabarito: CERTO

Uma lista é, por natureza, heterogênea, i.e., seus elementos são compostos por tipos de dados primitivos diferentes. A questão afirmou que a lista é uma coleção de elementos do mesmo tipo. E ela está certa, veja só o peguinha da questão:

Eu crio um tipo composto por dois tipos de dados diferentes: tipoEstrategia = {curso: caractere; duracao: inteiro}. Observe que o tipo tipoEstrategia é composto por tipos de dados primitivos diferentes (caractere e inteiro). Agora eu crio uma lista ListaEstrategia: tipoEstrategia. Veja que todos os elementos dessa lista terão o mesmo tipo (tipoEstrategia). Em outras palavras: a Lista Heterogênea é composta por elementos do mesmo tipo que, em geral, são compostos por mais de um tipo primitivo.

Além disso, as listas podem ser simplesmente encadeadas, duplamente encadeadas e ordenadas. E ainda podem ser circulares. Observem que alguns autores consideram Listas Ordenadas como um tipo de lista! *Como, professor?* Ela é uma lista em que seus elementos são ordenados (crescente ou decrescente).

8. (CESPE - 2009 - ANAC - Técnico Administrativo - Informática)

Em uma lista circular duplamente encadeada, cada nó aponta para dois outros nós da lista, um anterior e um posterior.

Gabarito: CERTO

Perfeito! Há dois ponteiros: uma para o nó anterior e um para o nó posterior.

9. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação)

A principal característica de uma lista encadeada é o fato de o último elemento da lista apontar para o elemento imediatamente anterior.

Gabarito: ERRADO

Não, o último elemento da lista não aponta para nenhum outro nó em uma lista não-circular.

10. (CESPE - 2009 - TCE-AC - Analista de Controle Externo - Processamentos de Dados)

Uma lista encadeada é uma coleção de nodos que, juntos, formam uma ordem linear. Se é possível os nodosse deslocarem em ambas as direções na lista, diz-se que se trata de uma lista simplesmente encadeada.

Gabarito: ERRADO

Se é possível os nodos se deslocarem em ambas as direções na lista, diz-se que se trata de uma lista **duplamente** encadeada.

11. (CESPE - 2008 – HEMOBRÁS – Técnico de Informática)

Uma estrutura do tipo lista, em que é desejável percorrer o seu conteúdo nas duas direções indiferentemente, é denominado lista duplamente encadeada.

Gabarito: CERTO

Perfeito, é exatamente isso!

12. (CESPE - 2010 – TRE/MT – Analista de Sistemas – C)

Uma lista duplamente encadeada é uma lista em que o seu último elemento referencia o primeiro.

Gabarito: ERRADO

Não, isso se trata de uma Lista Circular!

13. (CESPE - 2006 – SGA/AC – Analista de Sistemas)

O principal problema da alocação por lista encadeada é a fragmentação.

Gabarito: ERRADO

Não! Em geral, a alocação por lista encadeada elimina a fragmentação.

14. (CESPE - 2008 – MCT – Analista de Sistemas)

O armazenamento de arquivos em disco pode ser feito por meio de uma lista encadeada, em que os blocos de disco são ligados por ponteiros. A utilização de lista encadeada elimina completamente o problema de fragmentação interna.

Gabarito: ERRADO

Não, ela elimina a fragmentação externa!

15. (CESPE - 2009 – FINEP – Analista de Sistemas)

Uma lista encadeada é uma representação de objetos na memória do computador que consiste de uma sequência de células em que:

a) cada célula contém apenas o endereço da célula seguinte.

b) cada célula contém um objeto e o tipo de dados da célula seguinte.

c) o último elemento da sequência aponta para o próximo objeto que normalmente possui o endereço físico como not null.

d) cada célula contém um objeto de algum tipo e o endereço da célula seguinte.

e) a primeira célula contém o endereço da última célula.

Gabarito: D

Cada célula contém um objeto de algum tipo e o endereço da célula seguinte!

16. (CESPE - 2010 – BASA – Analista de Sistemas)

Em uma lista encadeada, o tempo de acesso a qualquer um de seus elementos é constante e independente do tamanho da estrutura de dados.

Gabarito: ERRADO

Claro que não! Em uma busca sequencial, o tempo de acesso é proporcional ao tamanho da estrutura de dados, i.e., quanto mais ao final da lista, maior o tempo de acesso! *Por que, professor?* Porque a lista é acessada sequencialmente (ou seja, é preciso percorrer elemento por elemento) e, não, diretamente (ou seja, pode-se acessar de modo direto). Um vetor tem acesso direto, portanto seu tempo de acesso é igual independentemente do tamanho da estrutura.

17. (CESPE - 2010 – INMETRO – Analista de Sistemas – C)

Considere que Roberto tenha feito uso de uma lista encadeada simples para programar o armazenamento e o posterior acesso aos dados acerca dos equipamentos instalados em sua empresa. Considere, ainda, que, após realizar uma consulta acerca do equipamento X, Roberto precisou acessar outro equipamento Y que se encontrava, nessa lista, em posição anterior ao equipamento X. Nessa situação, pela forma como os ponteiros são implementados em uma lista encadeada simples, o algoritmo usado por Roberto realizou a consulta ao equipamento Y sem reiniciar a pesquisa do começo da lista.

Gabarito: ERRADO

Não! Infelizmente, ele teve que reiniciar a pesquisa a partir do primeiro elemento da lista, na medida em que ele não pode voltar. *Por que, professor?* Porque se trata de uma lista encadeada simples e, não, dupla. Portanto, a lista só é percorrida em uma única direção.

18. (FCC - 2003 – TRE/AM – Analista de Sistemas)

Os dados contidos em uma lista encadeada estão:

a) ordenados seqüencialmente.
b) sem ordem lógica ou física alguma.
c) em ordem física e não, necessariamente, em ordem lógica.
d) em ordem lógica e, necessariamente, em ordem física.
e) em ordem lógica e não, necessariamente, em ordem física.

Gabarito: E

A Ordem Física é sua disposição na memória do computador e a Ordem Lógica é como ela pode ser lida e entendida. Ora, a ordem em que ela se encontra na memória pouco importa, visto que cada sistema operacional e cada sistema de arquivos tem sua maneira. Portanto, trata-se da ordem lógica e, não, necessariamente física.

19. (FCC - 2010 – DPE/SP – Analista de Sistemas)

Uma estrutura de dados que possui três campos: dois ponteiros e campo de informação denomina-se:

a) lista encadeada dupla.
b) Lista encadeada simples.
c) pilha.
d) fila.
e) vetor.

Gabarito: A

Trata-se da Lista Encadeada Dupla: dois ponteiros (Ant e Prox) e um campo de informação.

20. (CESPE - 2010 – TRE/MT – Analista de Sistemas)

O algoritmo para inclusão de elementos em uma pilha é usado sem nenhuma alteração para incluir elementos em uma lista.

Gabarito: C

Galera... uma pilha pode ser implementada por meio de uma lista! Ademais, o algoritmo para inclusão de elementos de ambas necessita do primeiro elemento (ou topo). Portanto, questão correta!

ACERTEI	ERREI

5.4 EXERCÍCIOS COMENTADOS: FILAS

1. (CESPE – 2017 - Analista Judiciário (TRT 7ª Região))

A lógica FIFO (first-in first-out) é utilizada na estrutura de dados do tipo
a) pointer ou ponteiros.
b) queue ou filas.
c) stack ou pilhas.
d) array ou matrizes.

Gabarito: **Letra B**.

a) pointer ou ponteiros.
ERRADO. São um tipo de dados primitivo, não uma estrutura de dados...

b) queue ou filas.
CERTO. Filas (ou Queue em inglês) utilizam a lógica FIFO - First-In First-Out (Primeiro a Entrar, Primeiro a Sair). Os elementos saem na ordem que entraram.

c) stack ou pilhas.
ERRADO. Pilhas (ou Stacks em inglês) utilizam a lógica FILO - First-In Last-Out (Primeiro a Entrar, Último a Sair). Os elementos saem na ordem inversa que entraram.

d) array ou matrizes.
ERRADO. São um tipo de estruturado que se caracterizam por tamanho fixo, tipo de dados homogêneo, e acesso direto pelo índice. Não obedece a regra FIFO.

Portanto, **Letra B** está correta.

2. (IADES - Profissional de Suporte Técnico (CFM) / 2018 / / Assistente de Tecnologia da Informação)

A sigla FIFO refere-se a estruturas de dados do tipo fila. Como é o funcionamento em uma FIFO?
a) O primeiro objeto inserido na fila é o último a ser removido.
b) O primeiro objeto inserido na fila é também o primeiro a ser removido.
c) O último objeto inserido na fila é o primeiro a ser removido.
d) O programador irá definir a ordem de entrada e de saída dos objetos em uma FIFO.
e) Uma FIFO e uma LIFO possuem as mesmas características de entrada e de saída dos objetos.

Gabarito: **Letra B**

FIFO é o acronismo formado pelas primeiras letras de **F**irst **I**nt, **F**irst **O**ut (primeiro a entrar, primeiro a sair), ou seja, uma **Fila**.

a) O primeiro objeto inserido na fila é o último a ser removido.
ERRADO - primeiro a entrar é o primeiro a ser removido.

b) O primeiro objeto inserido na fila é também o primeiro a ser removido.
CERTO - é o primeiro a ser removido.

c) O último objeto inserido na fila é o primeiro a ser removido.
ERRADO - ultimo a entrar é o último a ser removido.

d) O programador irá definir a ordem de entrada e de saída dos objetos em uma FIFO.
ERRADO - a ordem é a da fila.

e) Uma FIFO e uma LIFO possuem as mesmas características de entrada e de saída dos objetos.
ERRADO - LIFO é **L**ast **I**n, **F**irst **O**ut (último a entrar é o primeiro a sair), que é o conceito de **PILHA**.

3. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Análise de Sistemas)

Em um programa existe a necessidade de guardar todas as alterações feitas em determinado dado para que seja possível desfazer alterações feitas ao longo de toda a sua existência. Nessa situação, a estrutura de dados mais adequada para o armazenamento de todas as alterações citadas seria uma fila.

Gabarito: E

Não! Pensem comigo: eu faço uma atividade, depois outra, depois mais uma e, por fim, mais outra. Se eu desejo desfazer a última atividade realizada para retornar a um estado anterior, eu preciso de uma pilha. Dessa forma, resgata-se o último estado válido e, não, o primeiro.

4. (CESPE - 2012 – TST – Analista de Sistemas – A)

As pilhas e as filas são estruturas de dados essenciais para os sistemas computacionais. É correto afirmar que a fila é conhecida como lista LIFO - Last In First Out.

Gabarito: ERRADO

Não, Fila é FIFO!

5. (CESPE - 2012 - TRE-RJ - Técnico Judiciário - Programação de Sistemas)

As filas são estruturas com base no princípio LIFO (last in, first out), no qual os dados que forem inseridos primeiro na fila serão os últimos a serem removidos. Existem duas funções que se aplicam a todas as filas: PUSH, que insere um dado no topo da fila, e POP, que remove o item no topo da fila.

Gabarito: ERRADO

Não, isso é uma Pilha (LIFO).

6. (FCC - 2012 - MPE-AP – Analista de Sistemas - A)

Nas estruturas de dados, devido às características das operações da fila, o primeiro elemento a ser inserido será o último a ser retirado. Estruturas desse tipo são conhecidas como LIFO.

Gabarito: ERRADO

Não, será o primeiro a ser retirado – são do tipo FIFO!

7. (FCC - 2012 - MPE-AP – Analista de Sistemas - C)

Nas estruturas de dados, a fila é uma lista linear na qual as operações de inserção e retirada ocorrem apenas no início da lista.

Gabarito: ERRADO

Não, isso é a definição de Pilha!

8. (FCC - 2012 - TRE-SP - Analista Judiciário - Análise de Sistemas – D)

Pela definição de fila, se os elementos são inseridos por um extremo da lista linear, eles só podem ser removidos pelo outro.

Gabarito: CERTO

Exato! Essa é a definição de fila: insere-se por um extremo e remove-se por outro.

9. (FCC - 2011 - TRT - 19ª Região (AL) - Analista Judiciário - Tecnologia da Informação)

FIFO refere-se a estruturas de dados do tipo:

a) fila.
b) árvore binária.
c) pilha.
d) matriz quadrada.
e) cubo.

Gabarito: A

Trata-se da Fila!

10. (ESAF - 2010 - CVM - Analista de Sistemas - prova 2)

Uma fila é um tipo de lista linear em que:

a) as inserções são realizadas em um extremo e as remoções no outro extremo.
b) as inserções e remoções são realizadas em um mesmo extremo.
c) podem ser realizadas apenas inserções.
d) a inserção de um elemento requer a remoção de outro elemento.
e) a ordem de saída não corresponde à ordem de entrada dos elementos.

Gabarito: A

As inserções são realizadas em um extremo e as remoções são realizadas no outro extremo, por isso é FIFO!

11. (CESPE - 2010 - DETRAN-ES - Analista de Sistemas)

No armazenamento de dados pelo método FIFO (first in - first out), a estrutura de dados é representada por uma fila, em cuja posição final ocorrem inserções e, na inicial, retiradas.

Gabarito: CERTO

Perfeito! Basta lembrar de uma fila: o primeiro a entrar é o primeiro a sair.

12. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação)

Entre alguns tipos de estrutura de dados, podem ser citados os vetores, as pilhas e as filas.

Gabarito: CERTO

Perfeito, são todos exemplos de estruturas de dados!

13. (CESPE - 2004 – SES/PA – Analista de Sistemas)

Uma estrutura mais geral que as pilhas e filas é o deque, em que as inserções, retiradas e acessos são permitidos em ambas as extremidades.

Gabarito: CERTO

Perfeito, deques permitem todas essas operações!

14. (CESPE - 2009 – TCE/AC – Analista de Sistemas – D)

Um deque (double ended queue) requer inserção e remoção no topo de uma lista e permite a implementação de filas com algum tipo de prioridade. A implementação de um deque, geralmente é realizada com a utilização de uma lista simplesmente encadeada.

Gabarito: ERRADO

Não, pode ser do início ou fim da lista! De fato, permite a implementação de filas com algum tipo de prioridade, mas geralmente é realizada com a utilização de filas duplamente encadeadas.

15. (FCC - 2007 – TRT/23 – Analista de Sistemas)

Uma estrutura de dados com vocação de FIFO de duplo fim e que admite a rápida inserção e remoção em ambos os extremos é:

a) uma pilha.
b) uma splay tree.
c) um deque.
d) uma lista linear.
e) uma árvore AVL.

Gabarito: C

Trata-se de um Deque – fila duplamente encadeada!

16. (CESPE - 2004 – PBV/RR - Analista de Sistemas)

As filas com prioridade são listas lineares nas quais os elementos são pares da forma (q_i, p_i), em que q é o elemento do tipo base e p é uma prioridade. Elas possuem uma política de fila do tipo FIFO (first in first out) entre os elementos de mesma prioridade.

Gabarito: CERTO

Perfeito! É assim que funciona a prioridade em conjunto com filas.

17. (CESPE - 2004 – STJ – Analista de Sistemas)

A seguir, está representada corretamente uma operação de retirada em uma fila de nome f.

se f.começo = nil então
erro {fila vazia}
senão j ← f.começo.info

Gabarito: ERRADO

Não, o correto seria:

se f.começo = nil então
erro {fila vazia}
senão f.começo ← f.começo.anterior

Por que, professor? As duas primeiras linhas estão apenas dizendo que se o primeiro elemento da fila for Null (ou Nil), vai dar erro porque a fila está vazia, logo não há como retirar elementos de uma fila vazia. Agora vejam a última linha: ele atribui a uma variável j o valor f.começo.info. Na verdade, ele simplesmente está colocando em j os dados do primeiro elemento da fila, mas a questão pede o código para retirar um elemento da lista e não para capturar os dados do primeiro elemento. Na resposta, eu coloco que f.começo, i.e., o primeiro elemento da lista vai ser f.começo.anterior, ou seja, temos um novo primeiro elemento e eu retirei aquele elemento anterior.

18. (FCC - 2016 - Prefeitura de Teresina - PI - Analista Tecnológico - Analista de Suporte Técnico)

Considerando uma estrutura de dados do tipo fila, e a seguinte sequência de comandos sobre essa fila (sendo que o comando Push representa uma inserção de elemento e o comando Pop representa uma exclusão de elemento) e considerando também que a fila estava inicialmente vazia:

Push 3, Push 5, Pop 3, Push 7, Pop 5, Push 9, Push 8

Após a execução dessa sequência de comandos, o conjunto de elementos que resulta na fila é:

a) 3 – 5 – 7 – 9 – 8.
b) 7 – 9 – 8 – 3 – 5.
c) 3 – 3 – 5 – 5 – 7 – 9 – 8.
d) 7 – 9 – 8.
e) 3 – 5 – 3 – 7 – 5 – 9 – 8.

Gabarito: D

Como a questão nem pediu a ordem, ficou bem fácil. Push inclui e pop retira. Se há dois pop's, os elementos 3 e 5 são removidos da fila, sobrando 7, 9 e 8.

19. (FCC - 2016 – TRT - 23ª REGIÃO (MT) - Técnico Judiciário - Tecnologia da Informação)

Estruturas de dados básicas, como as pilhas e filas, são usadas em uma gama variada de aplicações. As filas, por exemplo, suportam alguns métodos essenciais, como o:

a) enqueue(x), que insere o elemento x no fim da fila, sobrepondo o último elemento.

b) dequeue(), que remove e retorna o elemento do começo da fila; um erro ocorrerá se a fila estiver vazia.

c) push(x), que insere o elemento x no topo da fila, sem sobrepor nenhum elemento.

d) pop(), que remove o elemento do início da fila e o retorna, ou seja, devolve o último elemento inserido.

e) top(), que retorna o elemento do fim da fila sem removê-lo; um erro ocorrerá se a fila estiver vazia.

Gabarito: B

Queue = fila! Stack = pilha! Sendo assim, o que seria "enqueue"? Para quem tem idade (assim como eu!! ☹) deve se lembrar do famoso Winamp (ah, minha época de ouvir mp3!). Quanto clicávamos no botão direito de uma música, sempre tinha a opção "enqueue in Winamp", ou seja, incluir na fila de reprodução. Enqueue, portanto, inclui na fila, enquanto dequeue remove!

20. (FCC - 2017 – TRE/BA - Analista de Sistemas)

A estrutura que, além de ser similar à fila, é apropriada para ampliar as características desta, permitindo inserir e retirar elementos tanto do início quanto do fim da fila, é o(a):

a) árvore.
b) lista duplamente encadeada.
c) deque.
d) fila circular.
e) pilha

Gabarito: C

Trata-se de um Deque (Double Ended Queue). Deque é uma estrutura de dados similar à fila e que permite que elementos possam ser adicionados ou removidos da frente (cabeça) ou de trás (cauda). *Qual a diferença entre uma lista duplamente encadeada e um deque?* Um deque gerencia elementos como um vetor, fornece acesso aleatório e tem quase a mesma interface que um vetor.

Uma lista duplamente encadeada se difere de um deque por não fornecer acesso aleatório aos elementos, i.e., para acessar o quinto elemento, você deve necessariamente navegar pelos quatro primeiros elementos – logo a lista é mais lenta nesse sentido. Existem outras diferenças, mas essa é a diferença fundamental entre essas duas estruturas. *Bacana?*

ACERTEI	ERREI

5.5 EXERCÍCIOS COMENTADOS: PILHAS

1. (CESPE - Oficial Técnico de Inteligência / 2018 / / Área 8)

Pilha é uma estrutura de dados em que o último elemento a ser inserido será o primeiro a ser retirado.

Gabarito: **CERTO**.

Pilha é uma estrutura do tipo **LIFO** - **L**ast **I**n, **F**irst **O**ut - Último a Entrar, Primeiro a Sair.
Fila é uma estrutura do tipo **FIFO** - **F**irst **I**n, **F**irst **O**ut - Primeiro a Entrar, Primeiro a Sair.

2. (FUNDATEC - Analista Legislativo (ALERS) / 2018 / / Analista de Tecnologia da Informação e Comunicação)

Suponha que, após a criação de uma pilha vazia de números inteiros, a sequência de operações abaixo tenha sido executada. Quantos elementos esta pilha terá ao término da execução desta sequência?

PUSH(7); PUSH(5); PUSH(3); PUSH(3); POP(); CONSULTA(); PUSH(2); PUSH(1); POP(); POP(); PUSH(17); PUSH(33); POP(); CONSULTA(); POP(); POP(); CONSULTA(); POP(); PUSH(22); PUSH(80); POP(); CONSULTA(); POP(); POP(); PUSH(4);

a) 0.
b) 1.
c) 2.
d) 3.
e) 4.

Gabarito: **Letra B**

O comando PUSH empurra um elemento no topo da pilha (aumenta em 1 o número de elementos), e o comando POP remove o elemento do topo (diminui em 1 o número de elementos). O comando CONSULTA só consulta o elemento no topo, sem modificar o número de elementos da pilha. São 11 PUSHs e 10 POPs. Portanto, o **tamanho final da pilha é Letra B) 1.**

O tamanho da pilha após cada comando será:

```
PUSH(7);   1
PUSH(5);   2
PUSH(3);   3
PUSH(3);   4
POP();   3
CONSULTA();
PUSH(2);   4
PUSH(1);   5
POP();   4
POP();   3
PUSH(17);   4
PUSH(33);   5
POP();   4
CONSULTA();
POP();   3
POP();   2
CONSULTA();
POP();   1
PUSH(22);   2
PUSH(80);   3
POP();   2
CONSULTA();
POP();   1
```

```
POP();   0
PUSH(4); 1
```

3. (FGV - Analista do Ministério Público (MPE AL) / 2018 / / Administrador de Rede)

Considere as seguintes operações sobre uma estrutura de dados, inicialmente vazia, organizada na forma de pilhas (ou stack),

PUSH (10)
PUSH (2)
POP ()
POP ()
PUSH (6)

Assinale a opção que apresenta a lista de elementos armazenados na estrutura, após a execução das operações acima.
a) 10, 2, 6
b) 10, 2
c) 2, 6
d) 6
e) 2

Gabarito: **Letra D**
Pilhas são coleções lineares no modelo FILO - Fist-In Last-Out (Primeiro a entrar é o último a sair). A operação PUSH(X) coloca X no topo da pilha, a operação POP() retira o elemento do topo da pilha, e a operação TOP() consulta o topo sem retirá-lo.

Assim, considere o conteúdo da pilha, após cada operação.

INICIAL - topo -> [ ]
PUSH (10) - topo -> [10]
PUSH (2) - topo -> [2, 10]
POP () - topo -> [10]
POP () - topo -> []
PUSH (6) - topo -> [6]

Portanto, resposta correta é **Letra D - 6**.

4. (FGV - Analista Legislativo (ALERO) / 2018 / Banco de Dados / Tecnologia da Informação)

Considere uma pilha de latas de sardinhas na prateleira de um supermercado.

Assinale a estrutura de dados que mais se assemelha ao modo como essas latas são manuseadas.
a) Array.
b) Binary tree.
c) Hashing.
d) Linked list.
e) Stack.

Gabarito: **Letra E**.

A questão é tão fácil que parece pega. O próprio examinador coloca "**pilha** de latas de sardinhas". Pilha, em inglês, *Stack*, **Letra E**.

5. (FUNRIO - Analista Legislativo (CM SJM) / 2018 / / Analista em Tecnologia)

Considere a estrutura de dados PILHA suportando três operações básicas, conforme definidas no quadro I abaixo.

Quadro I	
Operação	**Significado**
Push (SJM,e)	Insere um elemento qualquer e na pilha SJM.
Pop(SJM)	Remove o elemento de topo na pilha SJM.
Top(SJM)	Acessa, sem remover, o elemento do topo da pilha SJM.

Quadro II
Sequência de Operações
Push(SJM,HONDA)
Push(SJM,RENAULT)
Push(SJM,HYUNDAI)
Push(SJM,FIAT)
Top(SJM)
Push(SJM,Pop(SJM))
Push(SJM,VW)
Push(SJM,Top(SJM))
Pop(SJM)
Pop(SJM)

Considerando-se uma pilha SJM inicialmente vazia e a sequência de operações indicadas no quadro II, ao final das operações o elemento que se encontra no topo da pilha é:

a) VW.
b) FIAT.
c) HONDA.
d) RENAULT.
e) HYUNDAI.

Gabarito: **Letra B**

A pilha, após cada comando, fica da seguinte forma:

Operação	**Pilha**
PUSH(HONDA)	**[HONDA]** <-topo
PUSH(RENAULT)	**[HONDA, RENAULT]** <-topo
PUSH(HYUNDAI)	**[HONDA, RENAULT, HYUNDAI]** <-topo
PUSH(FIAT)	**[HONDA, RENAULT, HYUNDAI, FIAT]** <-topo
TOP()	**[HONDA, RENAULT, HYUNDAI, FIAT]** <-topo
PUSH(POP())*	**[HONDA, RENAULT, HYUNDAI, FIAT]** <-topo
PUSH(VW)	**[HONDA, RENAULT, HYUNDAI, FIAT, VW]** <-topo
PUSH(TOP())**	**[HONDA, RENAULT, HYUNDAI, FIAT, VW, VW]** <-topo
POP()	**[HONDA, RENAULT, HYUNDAI, FIAT, VW]** <-topo
POP()	**[HONDA, RENAULT, HYUNDAI, FIAT]** <-topo

* o comando PUSH(POP()) não modifica a pilha, pois coloca no topo (PUSH) o que acabou de tirar (POP).
** o comando PUSH(TOP()) duplica o elemento no topo, pois coloca no topo novamente (PUSH) o que já está lá (TOP).

Assim, a pilha termina com **FIAT** no topo, **letra b**.

6. (AOCP - Técnico em Gestão de Infraestrutura (SUSIPE) / 2018 / / Gestão de Informática)

Várias estruturas de dados podem ser utilizadas para armazenar dados de uma aplicação. Em relação ao assunto, assinale a alternativa correta.

a) Uma estrutura de dados do tipo pilha sempre retira os elementos que foram inseridos primeiro na estrutura.

b) Uma estrutura de dados do tipo lista utiliza a ideia do primeiro a chegar, primeiro a ser servido para inserir elementos.

c) Uma estrutura de dados do tipo fila sempre retira os elementos que entraram por último na fila.

d) Em uma estrutura de dados do tipo pilha, para retirar o elemento do topo da pilha, é necessário retirar o elemento base da pilha.

e) Uma estrutura de dados do tipo fila utiliza a ideia do primeiro a chegar, primeiro a ser servido.

Gabarito: **Letra E**.

a) Uma estrutura de dados do tipo pilha sempre retira os elementos que foram inseridos primeiro na estrutura.
ERRADO - Pilha é do tipo FILO (*first in, last out*), primeiro a sair foi o último a entrar.

b) Uma estrutura de dados do tipo lista utiliza a ideia do primeiro a chegar, primeiro a ser servido para inserir elementos.
ERRADO - lista é simplesmente uma coleção de elementos sequenciais. Não existe obrigatoriedade de inserção em local nenhum, nem no primeiro, nem no último.

c) Uma estrutura de dados do tipo fila sempre retira os elementos que entraram por último na fila.
ERRADO - Fila sempre retira os elementos que entraram primeiro na fila (FIFO).

d) Em uma estrutura de dados do tipo pilha, para retirar o elemento do topo da pilha, é necessário retirar o elemento base da pilha.
ERRADO - Não existe operação na base da pilha, somente no seu topo.

e) Uma estrutura de dados do tipo fila utiliza a ideia do primeiro a chegar, primeiro a ser servido.
CORRETO - Fila é do tipo FIFO (first in, first out) - primeiro a chegar, primeiro a servir.

7. (CESPE - 2011 - FUB - Analista de Tecnologia da Informação - Específicos)

As pilhas são listas encadeadas cujos elementos são retirados e acrescentados sempre ao final, enquanto as filas são listas encadeadas cujos elementos são retirados e acrescentados sempre no início.

Gabarito: ERRADO

Bem... o que é o final de uma Pilha? Pois é, não se sabe! O que existe é o Topo da Pilha, de onde sempre são retirados e acrescentados elementos. Em Filas, elementos são retirados do início e acrescentados no final.

8. (CESPE - 2013 - INPI - Analista de Planejamento - Desenvolvimento e Manutenção de Sistemas)

Na estrutura de dados do tipo lista, todo elemento novo que é introduzido na pilha torna-se o elemento do topo.

Gabarito: ERRADO

Que questão confusa! Vamos comigo: vocês sabem muito bem que filas e pilhas são considerados espécies de listas. A questão inicialmente fala de uma lista, mas depois ela menciona uma pilha – podemos inferir, então, que se trata de uma lista do tipo pilha. Em uma pilha, todo elemento novo que é introduzido torna-se o elemento do topo, logo... questão correta!

Bem, esse foi o Gabarito Preliminar, mas a banca mudou de opinião e, no Gabarito Definitivo, permaneceu como errada. E a justificativa dela foi: *"A ausência de especificação do tipo de lista no item torna correta a informação nele apresentada, razão pela qual se opta pela alteração de seu gabarito".* Vejam que bizarro: se torna correta a informação apresentada, o gabarito definitivo deveria ser C e, não, E.

Além disso, a questão informa em sua segunda parte que se trata de uma pilha. Logo, não há que se falar em "ausência de especificação do tipo de lista". Enfim, questão péssima, horrível e mal redigida :(

9. (CESPE - 2012 - TJ-RO - Analista Judiciário - Analista de Sistemas Suporte – E)

Visitas a sítios armazenadas em um navegador na ordem last-in-first-out é um exemplo de lista.

Gabarito: ERRADO

Não! Last-In-First-Out (LIFO) é um exemplo de Pilha! Cuidado, pilhas podem ser implementadas como listas, mas esse não é o foco da questão.

10. (ESAF - 2013 - DNIT - Analista Administrativo - Tecnologia da Informação)

Assinale a opção correta relativa às operações básicas suportadas por pilhas.

a) Push: insere um novo elemento no final da pilha.
b) Pop: adiciona elementos ao topo da pilha.
c) Pull: insere um novo elemento no interior da pilha.
d) Top: transfere o último elemento para o topo da pilha.
e) Top: acessa o elemento posicionado no topo da pilha.

Gabarito: E

(a) Não, é no topo; (b) Não, remove do topo; (c) Não, não existe; (d) Não, simplesmente acessa e consulta o elemento do topo; (e) Perfeito!

11. (FCC - 2012 – TST - Analista de Sistemas – C)

As pilhas e as filas são estruturas de dados essenciais para os sistemas computacionais. É correto afirmar que a pilha é conhecida como lista FIFO - First In First Out.

Gabarito: E

Não! Pilha é LIFO e Fila é FIFO.

12. (FCC - 2012 – TRE/CE - Analista de Sistemas)

Sobre pilhas é correto afirmar:

a) Uma lista LIFO (Last-In/First-Out) é uma estrutura estática, ou seja, é uma coleção que não pode aumentar e diminuir durante sua existência.

b) Os elementos na pilha são sempre removidos na mesma ordem em que foram inseridos.

c) Uma pilha suporta apenas duas operações básicas, tradicionalmente denominadas push (insere um novo elemento no topo da pilha) e pop (remove um elemento do topo da pilha).

d) Cada vez que um novo elemento deve ser inserido na pilha, ele é colocado no seu topo e, em qualquer momento, apenas aquele posicionado no topo da pilha pode ser removido.

e) Sendo P uma pilha e x um elemento qualquer, a operação Push(P,x) diminui o tamanho da pilha P, removendo o elemento x do seu topo.

Gabarito: D

(a) Não, é uma estrutura dinâmica; (b) Não, é na ordem inversa (último a entrar é o primeiro a sair); (c) Não, há também *Top* ou *Check*, que acessar e consulta o elemento do topo; (e) Push é a operação de inserção de novos elementos na pilha, portanto aumenta seu tamanho adicionando o elemento *x* no topo.

E a letra D? Vamos lá! Sabemos que uma pilha é um tipo de lista - o que muda é apenas a perspectiva. *Como assim?* Eu vejo um monte de elementos em sequência. Ora, se eu coloco uma regra em que o primeiro elemento a entrar é o primeiro a sair, chamamos essa lista de fila; se eu coloco uma regra em que o primeiro elemento a entrar é o último a sair, chamamos essa lista de pilha.

Ok! Dito isso, o algoritmo é exatamente o mesmo, eu vou simplesmente mudar a perspectiva e a minha visão sobre a estrutura. Bacana?

13. (CESPE - 2011 - EBC - Analista - Engenharia de Software)

As pilhas, também conhecidas como listas LIFO ou PEPS, são listas lineares em que todas as operações de inserção e remoção de elementos são feitas por um único extremo da lista, denominado topo.

Gabarito: ERRADO

Não! LIFO é similar a UEPS (Último a Entrar, Primeiro a Sair). PEPS refere-se a Primeiro a Entrar, Primeiro a Sair, ou seja, FIFO.

14. (VUNESP - 2011 - TJM-SP - Analista de Sistemas - Judiciário)

Lista do tipo LIFO (Last in, First Out) e lista do tipo FIFO (Firstin,First Out) são, respectivamente, características das estruturas de dados denominadas:

a) Fila e Pilha.
b) Pilha e Fila.
c) Grafo e Árvore.
d) Árvore e Grafo.
e) Árvore Binária e Árvore Ternária.

Gabarito: B

E aí, já está automático para responder? Tem que ser automática: Pilha (LIFO) e Fila (FIFO).

15. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Arquitetura de Tecnologia)

A definição da estrutura pilha permite a inserção e a eliminação de itens, de modo que uma pilha é um objeto dinâmico, cujo tamanho pode variar constantemente.

Gabarito: CERTO

Essa questão é polêmica, porque é inevitável pensar em Pilhas Sequenciais (implementadas por vetores estáticos)! No entanto, é comum que as bancas tratem por padrão Pilha como Pilha Encadeada (implementadas por listas dinâmicas). Dessa forma, a questão está perfeita!

16. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Administração de Dados)

Na representação física de uma pilha sequencial, é necessário uso de uma variável ponteiro externa que indique a extremidade da lista linear onde ocorrem as operações de inserção e retirada de nós.

Gabarito: CERTO

> *As Pilhas oferecem três operações básicas: push, que insere um novo elemento no topo da pilha; pop, que remove um elemento do topo da pilha; e top (também conhecida como check), que acessa e consulta o elemento do topo da pilha. Pilhas podem ser implementadas por meio de Vetores (Pilha Sequencial - Alocação Estática de Memória) ou Listas (Pilha Encadeada - Alocação Dinâmica de Memória).*

Conforme vimos em aula, a questão trata de uma Pilha Sequencial (i.e., implementada por meio de Vetores). Dessa forma, não é necessário o uso de ponteiros – esse seria o caso de uma Pilha Encadeada. Eu posso realmente dizer que é **suficiente**, mas não posso afirmar que é **necessária** a utilização de um ponteiro externo. Eu até poderia dizer que é necessário o uso de um indicador, mas ele também não necessariamente será um ponteiro. Logo, discordo do gabarito!

17. (CESPE - 2009 - ANAC - Técnico Administrativo - Informática)

As operações de inserir e retirar sempre afetam a base de uma pilha.

Gabarito: ERRADO

Não, sempre afetam o topo da pilha!

18. (FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação)

Pilha é uma estrutura de dados:

a) cujo acesso aos seus elementos segue tanto a lógica LIFO quanto a FIFO.
b) cujo acesso aos seus elementos ocorre de forma aleatória.
c) que pode ser implementada somente por meio de vetores.

d) que pode ser implementada somente por meio de listas.
e) cujo acesso aos seus elementos segue a lógica LIFO, apenas.

Gabarito: E

(a) Não, somente LIFO; (b) Não, somente pelo Topo; (c) Não, pode ser por listas; (d) Não, pode ser por vetores; (e) Perfeito, é exatamente isso.

19. (CESPE - 2004 – STJ – Analista de Sistemas)

Em geral, em uma pilha só se admite ter acesso ao elemento localizado em seu topo. Isso se adapta perfeitamente à característica das seqüências em que só o primeiro componente é diretamente acessível.

Gabarito: CERTO

Perfeito, é exatamente isso – muda-se apenas a perspectiva!

20. (CESPE - 2004 – STJ – Analista de Sistemas)

A seguir, está representada corretamente uma operação de desempilhamento em uma pilha de nome p.

se p.topo = 0 então
nada {pilha vazia}
senão p.topo ← p.topo-1

Gabarito: CERTO

Galera, a questão deu uma vaciladinha! O ideal seria dizer p.topo = null, mas vamos subentender que foi isso mesmo que ele quis dizer. Desse modo, se não tem topo, é porque a pilha está vazia. Se tiver topo, então o topo será o elemento anterior ao topo. *O que ocorreu?* Eu desempilhei a pilha!

21. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A)

O tipo nó é inadequado para implementar estruturas de dados do tipo pilha.

Gabarito: ERRADO

Não! Uma pilha pode ser implementada por meio de um vetor ou de uma lista. Nesse último caso, temos tipos nós!

22. (FGV – 2015 – DPE/MT – Analista de Sistemas)

Assinale a opção que apresenta a estrutura de dados na qual o primeiro elemento inserido é o último a ser removido.

a) Árvore
b) Fila
c) Pilha
d) Grafo
e) Tabela de dispersão

Gabarito: C

Também conhecida como Lista LIFO (Last In First Out), basta lembrar de uma pilha de pratos esperando para serem lavados, i.e., o último a entrar é o primeiro a sair. A ordem em que os pratos são retirados da pilha é o oposto da ordem em que eles são colocados sobre a pilha e, como consequência, apenas o prato do topo da pilha está acessível.

Conforme vimos em aula, trata-se da Pilha.

23. (FCC – 2012 – MPE/AP – Técnico Ministerial - Informática)
Nas estruturas de dados,

a) devido às características das operações da fila, o primeiro elemento a ser inserido será o último a ser retirado. Estruturas desse tipo são conhecidas como LIFO.

b) as pilhas são utilizadas para controlar o acesso de arquivos que concorrem a uma única impressora.

c) a fila é uma lista linear na qual as operações de inserção e retirada ocorrem apenas no início da lista.

d) a pilha é uma lista linear na qual as operações de inserção e retirada são efetuadas apenas no seu topo.

e) devido às características das operações da pilha, o último elemento a ser inserido será o último a ser retirado. Estruturas desse tipo são conhecidas como FIFO.

Gabarito: D

(a) as filas não são LIFO, mas sim FIFO, ou seja, o primeiro elemento da fila será, na verdade, o primeiro a ser retirado. Só pensarmos numa fila de banco, se alguém chega por último e é atendido primeiro, ficaria bem bravo, e vocês?? :D Item errado; (b) os trabalhos que chegam a uma impressora devem ser do tipo FIFO, ou seja, o primeiro trabalho enviado deve ser o primeiro a ser impresso. Item errado; (c) na fila os elementos são incluídos numa das extremidades e retirados da outra. Item errado; (d) na pilha as operações de inclusão na pilha quanto de retirada acontecem numa mesma extremidade. A extremidade escolhida é o topo da pilha. Item certo; (e) na verdade essas características são das filas. Item errado

ACERTEI	ERREI

5.6 EXERCÍCIOS COMENTADOS: ÁRVORES

1. (CESGRANRIO - Técnico Científico (BASA) / 2018 / / Tecnologia da Informação)

Uma árvore binária completa de busca, isto é, uma árvore em que todos os níveis têm o máximo número de elementos, tem um total de N nós. O número máximo de comparações necessárias para encontrar um elemento nessa árvore é

a) N
b) N 2
c) log 2(N+1)
d) (N+1)*log 2(N+1)
e) \sqrt{(N 1)}

Gabarito: **Letra C**.

Em uma árvore binária de busca, os elementos estão ordenados, sendo que os elementos menores que a chave do nó ficam na sub-árvore à esquerda, e os elementos maiores que a chave ficam na sub-árvore à direita.

Dessa forma, a cada teste você tem:

- Se o valor procurado for igual à chave do nó, você encontrou o elemento;
- Se o valor procurado for menor que a chave do nó, procure na sub-árvore à esquerda;
- Se o valor procurado for maior que a chave do nó, procure na sub-árvore à direita.
-

Como a árvore está completamente preenchida, a cada teste você irá encontrar o elemento, ou dividirá o conjunto de pesquisa na metade (sub-árvores à direita ou esquerda). Assim, você ira realizar, no máximo log2 (N +1) testes, **Letra C**.

2. (CESGRANRIO - Analista (PETROBRAS) / 2018 / / Sistema Júnior)

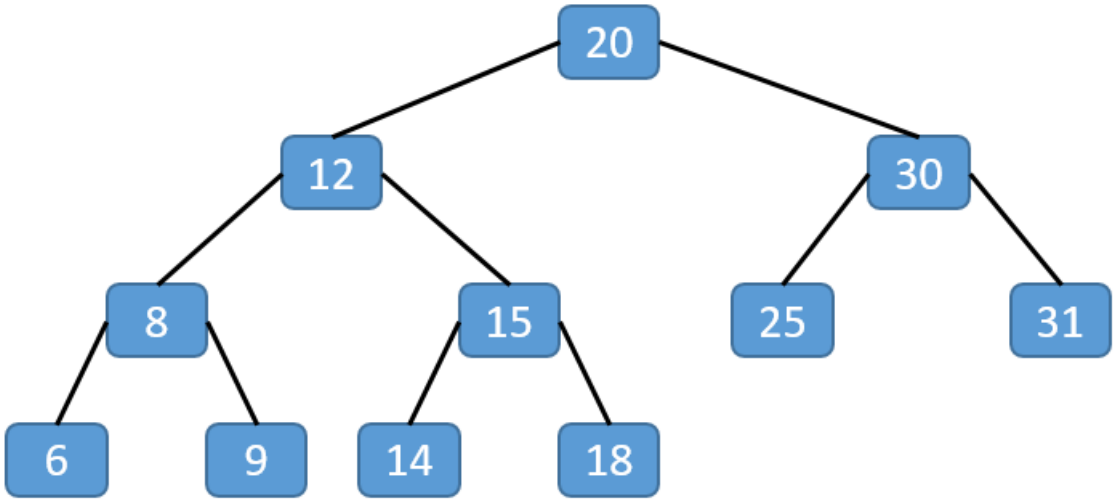
A sequência de chaves 20 – 30 – 25 – 31 – 12 – 15 – 8 – 6 – 9 – 14 – 18 é organizada em uma árvore binária de busca. Em seguida, a árvore é percorrida em pré-ordem.
Qual é a sequência de nós visitados?

a) 6 – 9 – 8 – 14 – 18 – 15 – 12 – 25 – 31 – 30 – 20
b) 20 – 12 – 8 – 6 – 9 – 15 – 14 – 18 – 30 – 25 – 31
c) 6 – 8 – 9 – 12 – 14 – 15 – 18 – 20 – 25 – 30 – 31
d) 20 – 30 – 31 – 25 – 12 – 15 – 18 – 14 – 8 – 9 – 6
e) 6 – 8 – 9 – 14 – 15 – 18 – 12 – 25 – 30 – 31 – 20

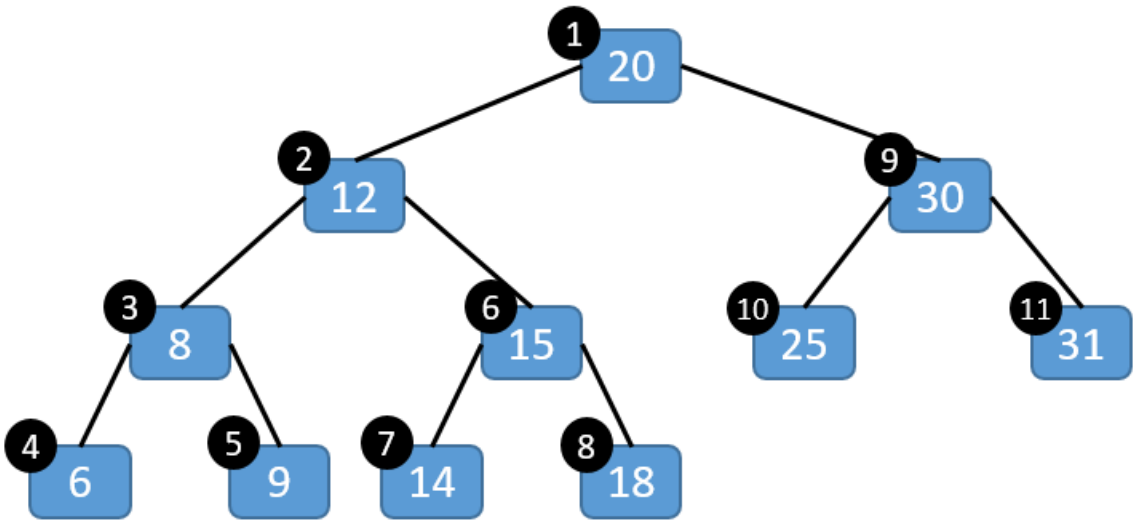
Gabarito: **Letra B**

A árvore binária de busca inclui os itens mantendo à esquerda os valores menores, e à direita os valores maiores. Sendo assim, a entrada 20 – 30 – 25 – 31 – 12 – 15 – 8 – 6 – 9 – 14 – 18 produzirá a árvore

binária:



A leitura em **Pré-Ordem** lê os nós da árvore da seguinte forma: nó raiz, sub-árvore esquerda, sub-árvore direita. Para cada sub-árvore, repete-se: primeiro o nó, depois esquerda, depois direita. Sendo assim, os nós serão lidos na seguinte ordem (nr. de ordem no círculo preto):



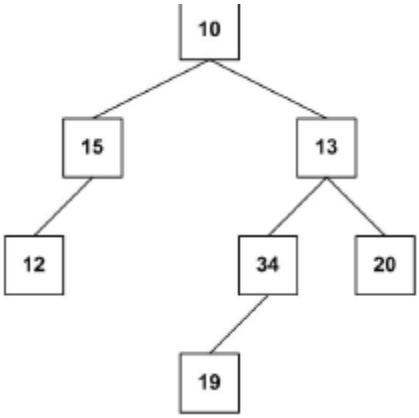
Portanto: 20 – 12 – 8 – 6 – 9 – 15 – 14 – 18 – 30 – 25 – 31, **Letra B.**

3. (CESGRANRIO - Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Infraestrutura)

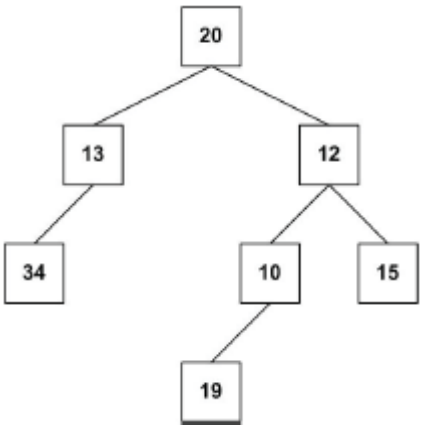
Uma árvore binária foi percorrida em ordem simétrica, e os valores de seus nós exibidos no console. O resultado desse procedimento foi o seguinte:

15 12 10 19 20 13 34

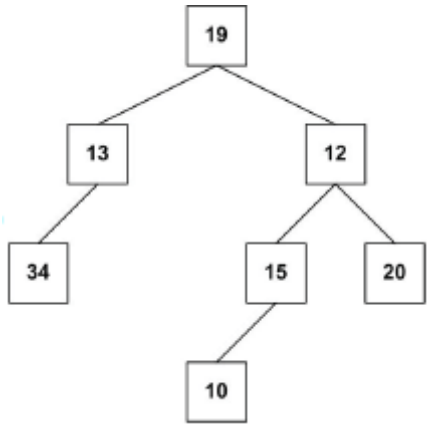
Dentre as árvores apresentadas, a única capaz de produzir o resultado acima é
a)



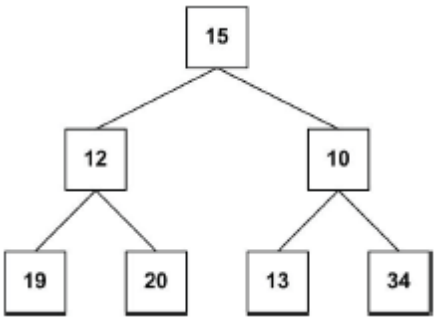
b)



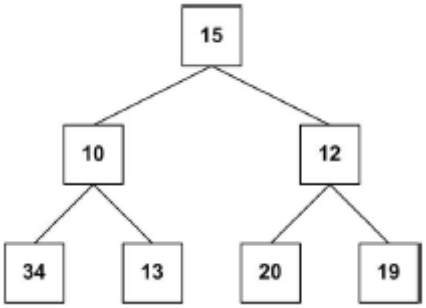
c)



d)



e)



Gabarito: **Letra B**.

A leitura em ordem é feita da seguinte forma: esquerda - nó - direita. O examinador *inventou* a leitura *em ordem simétrica:* direita - nó - esquerda. Curiosamente, não existe menção dessa forma de leitura em nenhuma literatura de referência... parece que foi coisa do examinador mesmo... Lendo as árvores seguindo: direita - nó - esquerda, temos:

a) 20-13-34-19-10-15-12
b) 15-12-10-19-20-13-34
c) 20-12-15-10-19-13-34
d) 34-10-13-15-20-12-19
e) 29-12-20-15-13-10-34

A ordem procurada é a da **Letra B**.

NOTA: infelizmente, a mesma banca já cobrou a "ordem simétrica" como esquerda-nó-direita... para mim, não existe de forma consolidada na literatura "em ordem simétrica" e, portanto, a questão **deveria ter sido anulada**.

4. (FCC - Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Certo documento possui 1 milhão de palavras não repetidas e foi editado em um editor de textos. Considerando que o editor de textos utiliza uma Árvore Binária de Busca − ABB de altura mínima para armazenar as palavras digitadas de forma a facilitar sua localização, para se localizar qualquer palavra nesta estrutura de dados serão necessárias, no máximo,

a) 1 milhão de comparações.
b) 20 comparações.
c) 32 comparações.
d) log101000000 comparações.
e) 2 milhões de comparações.

Gabarito: **Letra B**.

1 milhão de palavras em uma árvore binária. Cada comparação em uma árvore binária diminui o universo pesquisado pela metade. Sendo assim, o número de comparações máximas sempre será log _{2} (N). O valor exato de $Log_2(10^6)$ é 19.93, o seja, 19 testes não são o suficiente mas 20 certamente são.

Uma aproximação que facilita muito vários cálculos é que $10^3 \approx 2^{10}$ (1.000 é ligeiramente menor que 1024). Assim, 1 milhão é 10^6, que é o mesmo que $10^3 * 10^3$, que será ligeiramente menor que $2^{10}*2^{10}$ que é o mesmo que 2^{20}. Ora, $Log_2(2^{20}) = 20$, que será ligeiramente maior que $Log_2(10^6)$.

Portanto, resposta correta **Letra B**, 20 comparações.

5. (CESGRANRIO - Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Processos de Negócio)

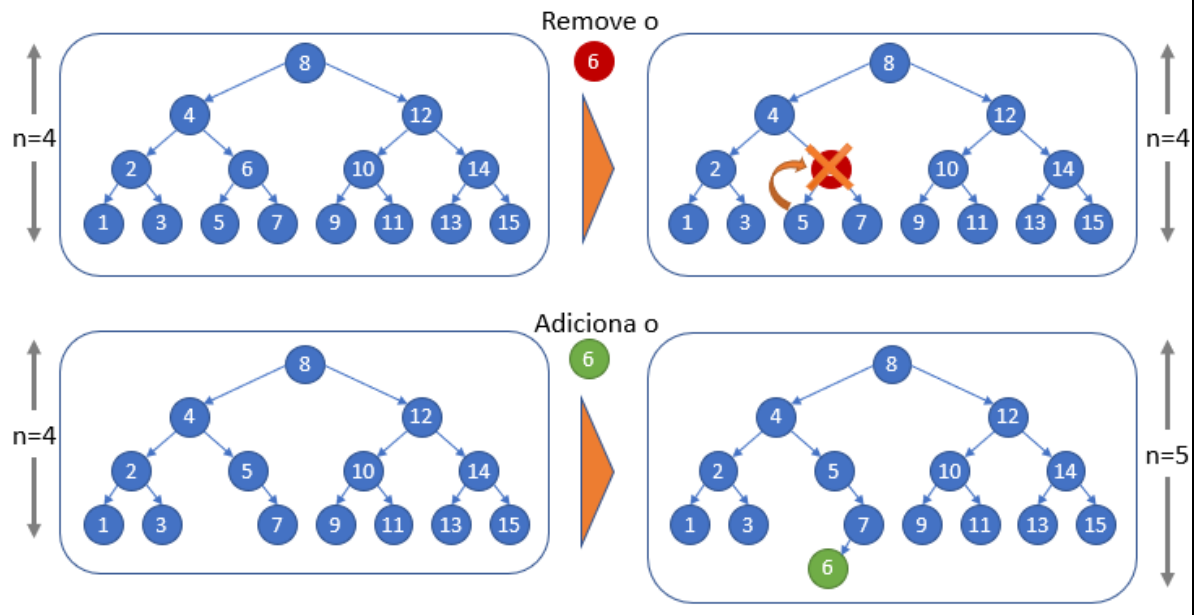
Considere uma árvore binária de busca (BST) com n (n>3) níveis (o nó raiz está no nível 1), 2n -1 nós e todas as chaves diferentes. Suponha, ainda, que algum dos pais de duas folhas seja removido da árvore e, mais tarde, uma chave com o mesmo valor da chave do nó removido seja inserida na árvore.

Quantas são as comparações necessárias para fazer a busca e encontrar o nó cuja chave foi removida e depois reinserida?

a) n – 2
b) n – 1
c) n
d) n +1
e) n + 2

Gabarito: **Letra D**.

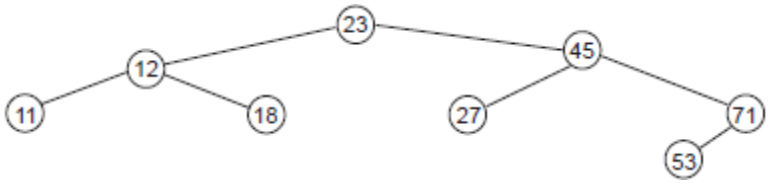
Imaginando uma BST completa, quando removemos um nó pai, o menor valor da sub-árvore à direita assume o local do nó removido. Ao adicioná-lo novamente, esse valor será maior que o nó que assumiu sua posição, e seguirá para sub-árvore direita. Porém, o valor é menor do que o valor do filho direito e, portanto, deverá se tornar filho esquerdo do filho direito, e a árvore ganhará 1 nível. Veja o exemplo abaixo, para n=4:



Logo, uma árvore de grau n, tornar-se-á n+1. Reposta, **Letra D**.

6. (CESGRANRIO - Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Processos de Negócio)

Analise a árvore binária de busca (BST), abaixo, representada pelas chaves dos seus nós.



Qual é a sequência de chaves representativa do seu percurso em pré-ordem?

a) 11; 12; 18; 23; 27; 45; 53; 71
b) 11; 18; 12; 27; 53; 71; 45; 23
c) 23; 12; 11; 18; 45; 27; 71; 53
d) 53; 71; 27; 45; 18; 11; 12; 23
e) 23; 45; 71; 27; 53; 18; 12; 11

Gabarito: **Letra C**.

Pré-Ordem: Lê o nó antes
In-Ordem (em Ordem): Lê o nó entre
Pós-Ordem: Lê o nó depois

A ordem de leitura das sub-árvores sempre é esquerda, depois direita. Em pré-ordem (nó antes), temos: NÓ-ESQUERDA-DIREITA. Portanto: 23-12-11-18-45-27-71-53 - **Letra C**.

7. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

As operações de busca em uma árvore binária não a alteram, enquanto operações de inserção e remoção de nós provocam mudanças sistemáticas na árvore.

Gabarito: CERTO

Perfeito! Operações de Busca não alteram nenhuma estrutura de dados. Já Operações de Inserção e Remoção podem provocar diversas mudanças estruturais.

8. (CETAP - 2010 - AL-RR - Analista de Sistemas - A)

Uma árvore binária é aquela que tem como conteúdo somente valores binários.

Gabarito: ERRADO

Não! Uma árvore binária é aquela que tem, no máximo, grau 2!

9. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

O tipo de dados árvore representa organizações hierárquicas entre dados.

Gabarito: CERTO

Perfeito, observem que alguns autores tratam Tipos de Dados como sinônimo de Estruturas de Dados.

10. (CETAP - 2010 - AL-RR - Analista de Sistemas - B)

Uma árvore é composta por duas raízes, sendo uma principal e a outra secundária.

Gabarito: ERRADO

Não, uma árvore possui somente um nó raíz!

11. (CESPE - 2010 - DETRAN-ES - Analista de Sistemas)

Denomina-se árvore binária a que possui apenas dois nós.

Gabarito: ERRADO

Não, árvore binária é aquela em que cada nó tem, no máximo, dois filhos!

12. (FUNCAB - 2010 - SEJUS-RO - Analista de Sistemas - II)

Árvores são estruturas de dados estáticas com sua raiz representada no nível um.

Gabarito: ERRADO

Não! Árvores são estruturas dinâmicas e sua raiz, em geral, é representada no nível 0 (mas depende de autor para autor).

13. (CESPE - 2009 - ANAC - Especialista em Regulação - Economia)

Considerando-se uma árvore binária completa até o nível 5, então a quantidade de folhas nesse nível será 2^4.

Gabarito: ERRADO

Não! A quantidade de folhas em um determinado nível – considerando a raiz como nível 0 –, é dada pela fórmula 2^d, portanto 2^5.

14. (CESPE - 2009 – ANAC - Analista de Sistemas)

Uma árvore binária completa até o nível 10 tem 2.047 nós.

Gabarito: CERTO

Se possui 10 níveis, possui (2^{d+1} -1): 2047 nós!

15. (FGV – 2015 – DPE/MT – Analista de Sistemas)

No desenvolvimento de sistemas, a escolha de estruturas de dados em memória é especialmente relevante. Dentre outras classificações, é possível agrupar essas estruturas em lineares e não lineares, conforme a quantidade de sucessores e antecessores que os elementos da estrutura possam ter. Assinale a opção que apresenta, respectivamente, estruturas de dados lineares e não lineares.

a) Tabela de dispersão e fila.
b) Estrutura de seleção e pilha.
c) Pilha e estrutura de seleção.

d) Pilha e árvore binária de busca.
e) Fila e pilha.

Gabarito: D

Além dessa classificação, existe outra também importante: Estruturas Lineares e Estruturas Não-Lineares. As Estruturas Lineares são aquelas em que cada elemento pode ter um único predecessor (exceto o primeiro elemento) e um único sucessor (exceto o último elemento). Como exemplo, podemos citar Listas, Pilhas, Filas, Arranjos, entre outros.

Já as Estruturas Não-Lineares são aquelas em que cada elemento pode ter mais de um predecessor e/ou mais de um sucessor. Como exemplo, podemos citar Árvores, Grafos e Tabelas de Dispersão. Essa é uma classificação muito importante e muito simples de entender. Pessoal, vocês perceberão que esse assunto é cobrado de maneira superficial na maioria das questões, mas algumas são nível doutorado!

Conforme vimos em aula, trata-se de pilha e árvore respectivamente.

16. (CESPE - 2010 – TRE/MT – Analista de Sistemas – B)
As listas, pilhas, filas e árvores são estruturas de dados que têm como principal característica a sequencialidade dos seus elementos.

Gabarito: ERRADO

Não! Árvores ntêm como principal característica a sequencialidade dos seus elementos.

17. (FCC - 2012 - MPE-AP - Analista Ministerial - Tecnologia da Informação – A)
A árvore é uma estrutura linear que permite representar uma relação de hierarquia. Ela possui um nó raiz e subárvores não vazias.

Gabarito: ERRADO

Árvore é uma estrutura linear? Não, hierárquica!

18. (CESPE - 2010 – TRE/MT - Analista de Sistemas – E)

O uso de recursividade é totalmente inadequado na implementação de operações para manipular elementos de uma estrutura de dados do tipo árvore.

Gabarito: ERRADO

Pelo contrário, é fundamental para implementação de operações.

19. (FCC - 2011 - TRT - 19ª Região (AL) - Técnico Judiciário - Tecnologia da Informação)

Em uma árvore binária, todos os nós têm grau:

a) 2.
b) 0, 1 ou 2.
c) divisível por 2.
d) maior ou igual a 2.
e) 0 ou 1.

Gabarito: B

Olha a pegadinha! Todos os nós têm grau 0 (Folha), 1 (Único filho) ou 2 (Dois filhos).

20. (CESPE - 2011 – STM – Analista de Sistemas)

Enquanto uma lista encadeada somente pode ser percorrida de um único modo, uma árvore binária pode ser percorrida de muitas maneiras diferentes.

Gabarito: CERTO

Galera, pense em uma árvore bem simples com um pai (raiz) e dois filhos. O Modo Pré-fixado vai ler primeiro a raíz, depois a sub-árvore da esquerda e depois a sub-árvore da direita. O Modo In-fixado vai ler primeiro a sub-árvore da esquerda, depois a raiz e depois a sub-árvore da direita. O Modo Pós-fixado vai ler primeiro a sub-árvore da esquerda, depois a sub-árvore da direita e depois a raiz.

Vamos resumir: o modo de percorrimento de uma árvore pode obedecer três regras de acordo com a posição da raiz (pai): pré-fixado (raiz, esquerda, direita); in-fixado (esquerda, raiz, direita); e pós-fixada (esquerda, direita, raiz). Vamos agora ver uma árvore de exemplo:

- Percorrimento Pré-fixado: X Y Z
- Percorrimento In-fixado: Y X Z
- Percorrimento Pós-fixado: Y Z X

Logo, uma árvore pode ser percorrida de modo pré-fixado, in-fixado e pós-fixado.

21. (FCC - 2016 - Prefeitura de Teresina - PI - Analista Tecnológico - Analista de Suporte Técnico)

Considerando a estrutura de dados denominada árvore,

a) a sua altura é definida como a profundidade média de todos os seus vértices.

b) um vértice com um ou dois filhos é denominado folha.

c) cada nó tem no mínimo dois filhos em uma árvore binária.

d) as folhas de uma árvore binária completa podem ter profundidades distintas entre si.

e) a profundidade de um vértice em uma árvore é definida como o comprimento da raiz da árvore até esse vértice.

Gabarito: E

(a) Errado! Altura é definida pela folha mais profunda; (b) Errado! Folhas não possuem filhos, do contrário não seriam folhas (*as arves... somo nozes... péssimo, professor :P*); (c) Errado! Os nós de uma árvore binaria podem ter NO MÁXIMO dois filhos, e não no mínimo; (d) Errado! Uma árvore binária completa é aquele em que todos os nós internos possuem seus dois filhos (máximo). Desse modo, todas as folhas devem ter a mesma profundidade; (e) Certo!

22. (CESPE – 2017 – TRE/BA - Analista de Sistemas)

No estabelecimento de uma estrutura hierárquica, foi definida a seguinte árvore binária S:

S = (12(10(9(8))(11))(14(13)(15)))

Considerando o resultado da operação de exclusão do nó 12, assinale a opção que corresponde a nova estrutura da árvore S.

a) (10(9(8))(11(14(13)(15)))
b) (11(9(8)(10))(14(13)(15)))
c) (11(10(9(8))(14(13)(15)))
d) (13(10(9)(11))(14(15)))
e) (13(11(9)(10))(14(15)))

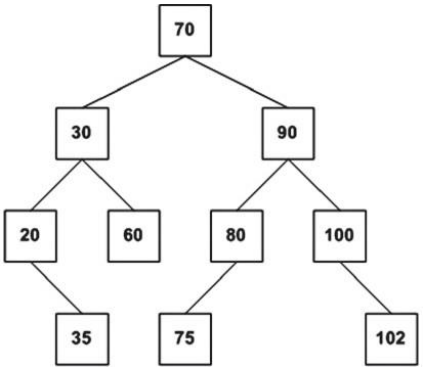
Gabarito: X

Conforme vimos em aula, a questão já inicia com um problema grave: ela não especifica qual tipo de árvore. Ainda assim, ela está errada e é fácil descobrir que não pode ser a Letra C (Gabarito Preliminar) porque a quantidade de parênteses abertos e fechados são diferentes – logo jamais poderia ser essa a resposta. Concluímos, então, que essa questão não possui resposta alguma, visto que falta fechar um parêntese. A Letra C (11(10(9(8))(14(13)(15))) está quase certa, mas faltou um parêntese: (11(10(9(8)))(14(13)(15))).

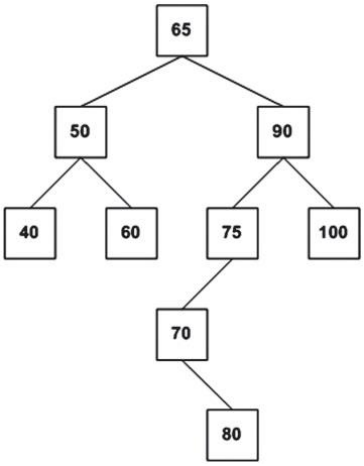
23. (CESGRANRIO – 2012 – PETROBRÁS - Analista de Sistemas)

Qual figura representa uma árvore AVL?

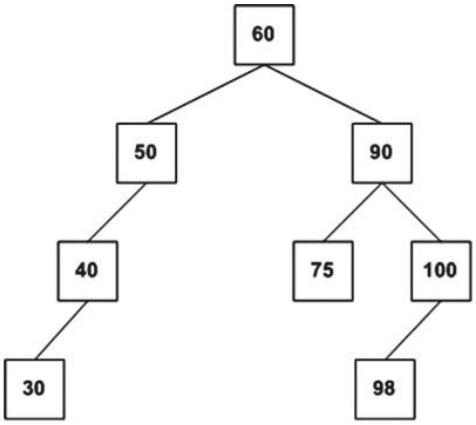
a)


70
30
90
20
60
80
100
35
75
102

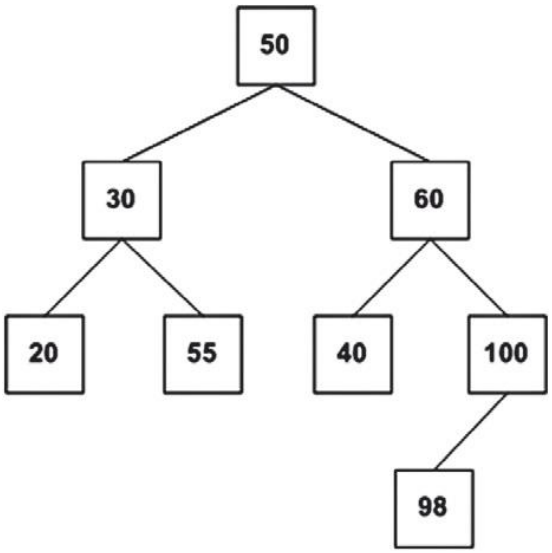
b)


65
50
90
40
60
75
100
70
80

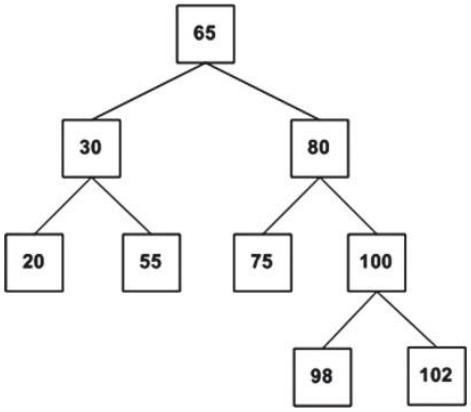
c)


60
50
90
40
75
100
30
98

d)



e)



Gabarito: E

Vamos relembrar dois conceitos: (1) Uma Árvore AVL é uma árvore binária de busca em que, para todos os seus nós, a diferença da altura da subárvore da esquerda para a altura da subárvore da direita deve ser no máximo 1. (2) Uma Árvore Binária de Busca é aquela em que, para todos os nós, a subárvore da esquerda possui um valor menor que a subárvore da direita. Dito isso, vamos analisar agora as opções:

(a) Errado. Não é uma árvore binária de busca, visto que o 35 é maior que 30 e está na subárvore da esquerda;

(b) Errado. Não é uma árvore binária de busca, visto que o 80 é maior que 75 e está na subárvore da esquerda;

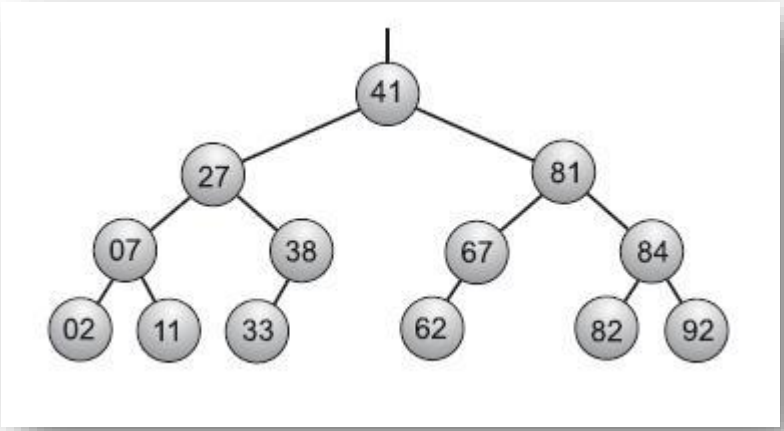
(c) Errado. Observem que se trata realmente de uma árvore binária de busca, porém está desbalanceada. A diferença de altura da subárvore da esquerda de 50 e a subárvore da direita de 50 é 2 (que é maior do 1), portanto não é uma Árvore AVL.

(d) Errado. Não é uma árvore binária de busca, visto que o 55 é maior que 50 e está na subárvore da esquerda;

(e) Correto. Trata-se de árvore binária de busca e está completamente balanceada – requisitos para ser definida como uma Árvore AVL.

24. (CESGRANRIO – 2006 – DECEA - Analista de Sistemas)

Suponha a seguinte árvore AVL.



A inserção do elemento 30 nessa árvore:

a) aumenta a profundidade da árvore após uma rotação.
b) provoca uma rotação à direita.
c) deixa os nós 02 e 07 no mesmo nível.
d) altera a raiz da árvore (nó 41).
e) torna o nó 33 pai do nó 27.

Gabarito: B

Vamos lá! A questão quer inserir 30. *Onde ele entraria?* À esquerda do 33! No entanto, isso tornaria a árvore desbalanceada. *Por que?* Porque a árvore à esquerda do nó 38 teria altura 2 e o nó à direita do nó 38 teria altura 0 – a diferença seria 2, que é maior do que 1. Podemos notar que se trata de um Caso Esquerda-Esquerda, logo temos que fazer uma rotação simples para direita.

Rotacionamos o Ramo 38-33 para direita. Dessa forma, teríamos o 33 no lugar do 38 e o 38 à direita do 33. Pronto! Agora vamos analisar as opções: (a) Errado. A profundidade permanece a mesma; (b) Correto. Foi isso que nós fizemos; (c) Errado. Isso não faz nenhum sentido; (d) Errado. Isso também não faz nenhum sentido; (e) Errado. Também nenhum sentido.

25. (CESPE – 2012 – TJ/RO - Analista de Sistemas)

Assinale a opção em que é apresentado exemplo de estrutura de informação do tipo abstrata, balanceada, não linear e com relacionamento hierárquico.

a) lista duplamente encadeada
b) árvore binária
c) pilha
d) árvore AVL
e) deque

Gabarito: D

Tipo abstrato? Todos são! *Não Linear?* Árvore Binária e Árvore AVL! *Com relacionamento hierárquico?* Árvore Binária e Árvore AVL! *Balanceada?* Somente a Árvore AVL.

26. (FCC – 2008 – TRT/18 - Analista de Sistemas)

Árvore AVL balanceada em altura significa que, para cada nó da árvore, a diferença entre as alturas das suas sub- árvores (direita e esquerda) sempre será:

a) menor ou igual a 2.
b) igual a 0 ou -1.
c) maior que 1.
d) igual a 1.
e) igual a -1, 0 ou 1.

Gabarito: E

Por fim, vamos falar um pouco sobre Árvores AVL! Uma Árvore AVL (Adelson-Vesky e Landis) é uma Árvore Binária de Busca em que, para qualquer nó, a altura das subárvores da esquerda e da direita não podem ter uma diferença maior do que 1, portanto uma Árvore AVL é uma Árvore Binária de Busca autobalanceável. Calma que nós vamos ver isso em mais detalhes...

Conforme vimos em aula, trata-se da última opção. Lembrando que, na aula, nós falamos que era a diferença não podia ser maior que 1. Uma outra forma de escrever isso é dizer que o módulo da diferença entre as alturas não pode ser maior do 1. E outra forma de escrever isso é dizer que um nó só poder ter uma diferença de altura de suas subárvores de 1, 0 ou -1. *Certinho?*

27. (CESGRANRIO – 2010 – PETROBRÁS - Analista de Sistemas)

Uma árvore AVL é uma estrutura de dados muito usada para armazenar dados em memória. Ela possui algumas propriedades que fazem com que sua altura tenha uma relação muito específica com o número de elementos nela armazenados. Para uma folha, cuja altura é igual a um, tem-se uma árvore AVL com 6 nós.

Qual é a altura máxima que esta árvore pode ter?

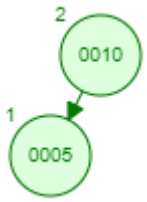
a) 6
b) 5
c) 4
d) 3
e) 2

Gabarito: D

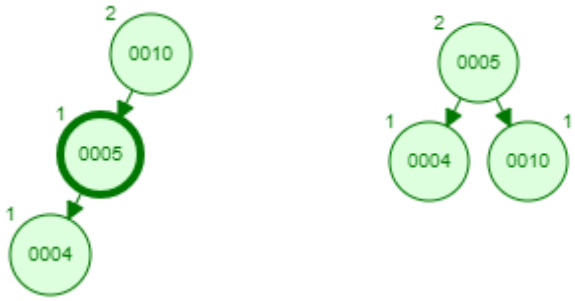
Galera, vocês precisam saber uma coisa: uma minoria dos autores considera que as folhas de uma árvore possuem altura 1 (sendo que a maioria considera que a altura de uma folha é 0) – a questão afirma que,

para uma folha, a altura é igual a 1. Dito isso, vamos analisar a questão! Ela te deu uma regra: você tem que construir uma Árvore AVL com 6.

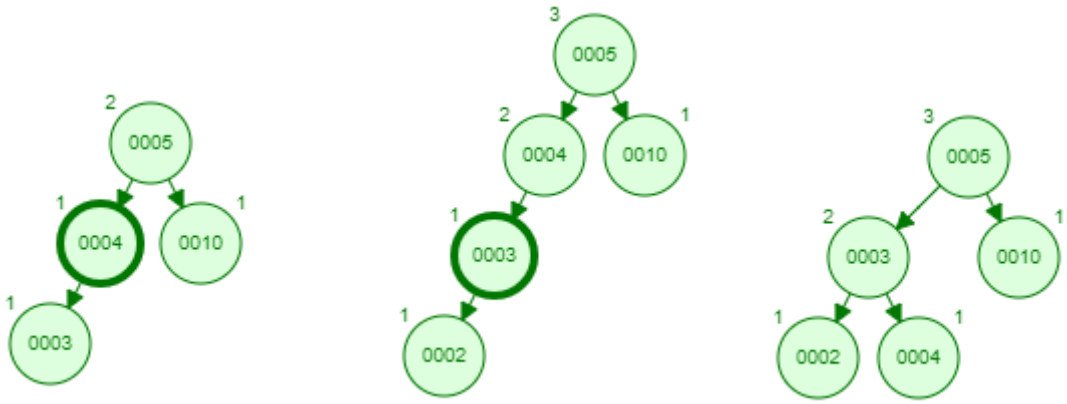
Em outras palavras, você tem que tentar construir a árvore com a maior altura possível, mas ela tem que estar balanceada. Então, eu começo com dois nós:



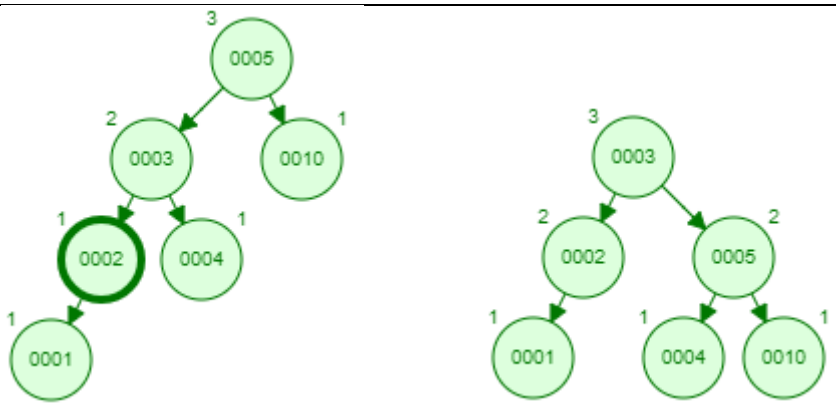
Como quero atingir a altura máxima, coloco mais um nó abaixo de 0005 e a árvore ficará com altura 3, mas ficará desbalanceada. Após balancear, fica como na direita:



Preciso encontrar a altura máxima, então coloco mais um nó (0003) e me sobrará mais duas tentativas. Se eu colocar mais um abaixo de 0003, ficará desbalanceado:



Vejam na imagem acima! Coloquei 0003 e não desbalanceou; coloquei 0002, desbalanceou e na última eu tive que rebalancear. Vamos inserir o último:



Pronto! Com seis nós, a altura máxima que atingiremos será 3. Portanto, a resposta para nossa questão é Letra D.

28. (CESGRANRIO – 2011 – PETROBRÁS - Analista de Sistemas)

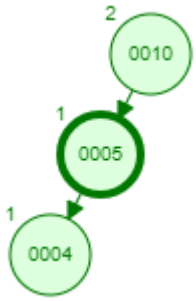
Uma árvore AVL é uma árvore binária de busca autobalanceada que respeita algumas propriedades fundamentais. Como todas as árvores, ela tem uma propriedade chamada altura, que é igual ao valor da altura de sua raiz.

Sabendo que a altura de uma folha é igual a um e que a altura de um nó pai é igual ao máximo das alturas de seus filhos mais um, qual estrutura NÃO pode representar uma árvore AVL?

a) Uma árvore vazia
b) Uma árvore com dois nós
c) Uma árvore com três nós e altura igual a dois
d) Uma árvore com três nós e altura igual a três
e) Uma árvore com seis nós e altura igual a três

Gabarito: D

(a) Errado, uma árvore vazia é uma Árvore AVL; (b) Errado, é impossível uma árvore com dois nós não ser uma Árvore AVL; (c) Errado, uma árvore com três nós e altura igual a dois é perfeitamente balanceada; (d) Correto, uma árvore com três nós e altura igual a três não pode estar balanceada (vide imagem abaixo); (e) Errado, uma árvore com seis nós e altura igual a três também está balanceada.



29. (CESGRANRIO – 2011 – PETROBRÁS - Analista de Sistemas)

Após a inserção de um nó, é necessário verificar cada um dos nós ancestrais desse nó inserido, relativamente à consistência com as regras estruturais de uma árvore AVL.

PORQUE

O fator de balanceamento de cada nó, em uma árvore AVL, deve pertencer ao conjunto formado por {-2, -1, 0, +1, +2}.

Analisando-se as afirmações acima, conclui-se que:

a) as duas afirmações são verdadeiras, e a segunda justifica a primeira.
b) as duas afirmações são verdadeiras, e a segunda não justifica a primeira.
c) a primeira afirmação é verdadeira, e a segunda é falsa.
d) a primeira afirmação é falsa, e a segunda é verdadeira.
e) as duas afirmações são falsas.

Gabarito: C

Galera, a primeira frase está perfeita! Após inserir um novo nó, você tem que verificar os nós ancestrais para se certificar de que a Árvore AVL continua balanceada. No entanto, o fator de balanceamento de cada nó deve pertencer ao conjunto formado por {-1, 0, 1}. Lembrem-se: o módulo da diferença jamais pode ser maior do que 1, portanto a primeira está verdadeira e a segunda falsa.

30. (CESGRANRIO – 2010 – EPE - Analista de Sistemas)

Um programador decidiu utilizar, em determinado sistema de análise estatística, uma árvore AVL como estrutura de dados. Considerando-se n a quantidade de elementos dessa árvore, o melhor algoritmo de pesquisa, com base em comparações, possui complexidade de tempo, no pior caso, igual a:

a) O(1)
b) O(log n).
c) Ω(n)
d) Ω(n log n)
e) $\Omega(n^2)$

Gabarito: B

ÁRVORE B / ÁRVORE AVL		
ALGORITMO	CASO MÉDIO	PIOR CASO
Espaço	O(n)	O(n)
Busca	O(log n)	O(log n)
Inserção	O(log n)	O(log n)
Remoção	O(log n)	O(log n)

Questão tranquila! Trata-se do O(log n).

31. (CESGRANRIO – 2012 – PETROBRÁS - Analista de Sistemas)

Todos os N nomes de uma lista de assinantes de uma companhia telefônica foram inseridos, em ordem alfabética, em três estruturas de dados: uma árvore binária de busca, uma árvore AVL e uma árvore B.

As alturas resultantes das três árvores são, respectivamente,

a) O(Log(N)), O(Log(N)), O(1)
b) O(Log(N)), O(N), O(Log(N))
c) O(N), O(Log(N)), O(1)

d) O(N), O(Log(N)), O(Log(N))
e) O(N), O(N), O(Log(N))

Gabarito: D

ÁRVORE BINÁRIA DE BUSCA		
ALGORITMO	CASO MÉDIO	PIOR CASO
Espaço	O(n)	O(n)
Busca	O(log n)	O(n)
Inserção	O(log n)	O(n)
Remoção	O(log n)	O(n)

ÁRVORE B / ÁRVORE AVL		
ALGORITMO	CASO MÉDIO	PIOR CASO
Espaço	O(n)	O(n)
Busca	O(log n)	O(log n)
Inserção	O(log n)	O(log n)
Remoção	O(log n)	O(log n)

Questão tranquila! Trata-se do O(n), O(log n) e O(log n) respectivamente.

32. (IBFC – 2014 – TRE/AM - Analista de Sistemas)

Quanto ao Algoritmo e estrutura de dados no caso de árvore AVL (ou árvore balanceada pela altura), analise as afirmativas abaixo, dê valores Verdadeiro (V) ou Falso (F) e assinale a alternativa que apresenta a sequência correta de cima para baixo:

( ) Uma árvore AVL é dita balanceada quando, para cada nó da árvore, a diferença entre as alturas das suas sub-árvores (direita e esquerda) não é maior do que um.

( ) Caso a árvore não esteja balanceada é necessário seu balanceamento através da rotação simples ou rotação dupla.

Assinale a alternativa correta:

a) F-F
b) F-V
c) V-F
d) V-V

Gabarito: D

A primeira alternativa está impecável, assim como a segunda. Vimos exaustivamente em aula!

33. (CESGRANRIO – 2010 – PETROBRÁS - Analista de Sistemas)

Uma sequência desordenada de números armazenada em um vetor é inserida em uma árvore AVL. Após a inserção nesta árvore, é feito um percurso em ordem simétrica (em ordem) e o valor de cada nó visitado é inserido em uma pilha. Depois de todos os nós serem visitados, todos os números são retirados da pilha e apresentados na tela. A lista de números apresentada na tela está:

a) ordenada ascendentemente de acordo com os números.
b) ordenada descendentemente de acordo com os números.

c) na mesma ordem do vetor original.
d) na ordem inversa do vetor original.
e) ordenada ascendentemente de acordo com sua altura na árvore.

Gabarito: B

Essa questão é legal! Olha só...

Eu tenho um vetor com um bocado de valor desordenado. Esses valores são colocados em uma Árvore AVL. Ora, uma árvore AVL é uma árvore binária de busca, portanto segue suas propriedades. Logo, não importa se estava desordenado. À medida que são inseridos os valores desordenados na árvore, ela vai se balanceando e ordenando os dados.

Após isso, vamos retirar os dados da Árvore AVL. *Como?* Da esquerda para a direita! E vamos colocá-los em uma pilha. Se estamos lendo da esquerda para a direita, estamos retirando do menor valor para o maior valor, logo o maior valor da pilha será o maior valor da Árvore AVL. Por fim, ao retirar os elementos da pilha, retiramos do topo (maior) para a base (menor), logo em ordem descendente.

34. (FGV – 2009 – MEC - Analista de Sistemas)

Acerca das estruturas de dados Árvores, analise as afirmativas a seguir.

I. A árvore AVL é uma árvore binária com uma condição de balanço, porém não completamente balanceada.

II. Árvores admitem tratamento computacional eficiente quando comparadas às estruturas mais genéricas como os grafos.

III. Em uma Árvore Binária de Busca, todas as chaves da subárvore esquerda são maiores que a chave da raiz.

Assinale:

a) se somente a afirmativa I estiver correta.
b) se somente as afirmativas I e II estiverem corretas.
c) se somente as afirmativas I e III estiverem corretas.
d) se somente as afirmativas II e III estiverem corretas.
e) se todas as afirmativas estiverem corretas.

Gabarito: B

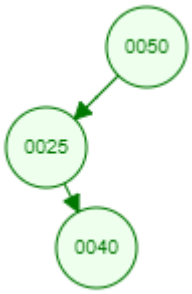
(I) Correto, ela é completamente balanceada; (II) Correto, isso é verdade! (III) Errado, são menores que a chave da raiz.

35. (CESPE – 2014 – TJ/SE - Analista de Sistemas)

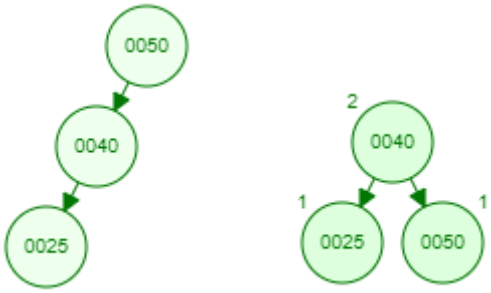
Em uma árvore AVL (Adelson-Velsky e Landis), caso a diferença de altura entre as sub-árvores de um nó seja igual a 2 e a diferença de altura entre o nó filho do nó desbalanceado seja igual a -1, deve-se realizar uma rotação dupla com o filho para a direita e o pai para a esquerda a fim de que a árvore volte a ser balanceada.

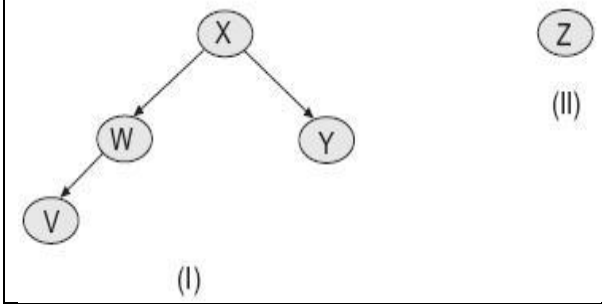
Gabarito: CERTO

Vamos entender a questão! Nós temos um nó (0050) cujo fator de balanceamento é 2. Isso significa que a altura da subárvore à esquerda desse nó (2) menos a altura da subárvore à direita (0) é igual a 2 [2-0 = 2]. O nó filho (0025) desse nó pai (0050) está com balanceamento igual a -1, visto que a altura da subárvore à esquerda desse nó (0) menos a altura da subárvore à direita (1) é igual a -1 [0-1 = -1].



Logo, temos um Caso Esquerda-Direita. E sabemos que para balancear essa árvore, devemos fazer uma rotação dupla: primeiro à esquerda no Ramo 0025-0040 (primeira imagem) e depois à direita no Ramo 0025-0050 (segunda imagem), portanto é exatamente o oposto do que a questão diz. No entanto, o Gabarito Definitivo foi Correto! Sinceramente, não faço ideia de onde o CESPE tirou isso...





36. (CESPE – 2010 – PETROBRÁS - Analista de Sistemas)

As árvores usadas como estruturas de pesquisa têm características especiais que garantem sua utilidade e propriedades como facilidade de acesso aos elementos procurados em cada instante. A esse respeito, considere as afirmações abaixo.

I - A árvore representada na figura (I) acima não é uma árvore AVL, pois as folhas não estão no mesmo nível.

II - A sequência 20, 30, 35, 34, 32, 33 representa um percurso sintaticamente correto de busca do elemento 33 em uma árvore binária de busca.

III - A árvore representada na figura (II) acima é uma árvore binária, apesar da raiz não ter filhos.

É (São) correta(s) APENAS a(s) afirmativa(s):

a) I.
b) II.
c) III.
d) I e II.
e) II e III.

Gabarito: E

(I) Errado. Trata-se, sim, de uma Árvore AVL; (II) Correto. Temos a sequência 20, 30, 35, 34, 32, 33 e estamos procurando o 33. 33>20, logo descemos à direita; 33>30, logo descemos à direita de novo; 33<35, logo descemos à esquerda; 33<34, logo descemos à esquerda de novo; 33>32, logo descemos à direita. Pronto, encontramos o 33; (III) Correto. Trata-se de um Árvore Binária sem filhos.

37. (CESPE – 2010 – PETROBRÁS - Analista de Sistemas)

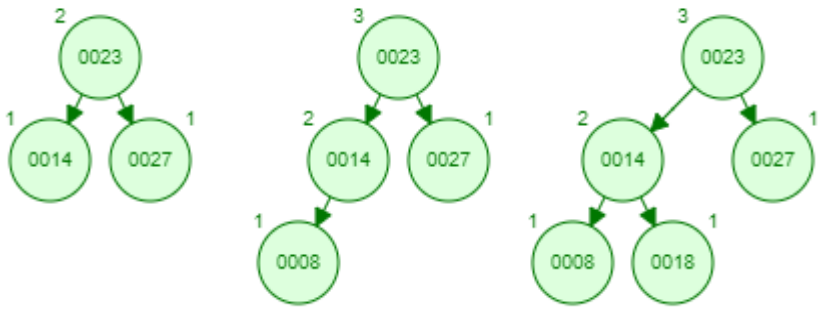
No sistema de dados do Departamento de Recursos Humanos de uma grande empresa multinacional, os registros de funcionários são armazenados em uma estrutura de dados do tipo árvore binária AVL, onde cada registro é identificado por uma chave numérica inteira. Partindo de uma árvore vazia, os registros cujas chaves são 23, 14, 27, 8, 18, 15, 30, 25 e 32 serão, nessa ordem, adicionados à árvore.

Dessa forma, o algoritmo de inserção na árvore AVL deverá realizar a primeira operação de rotação na árvore na ocasião da inserção do elemento:

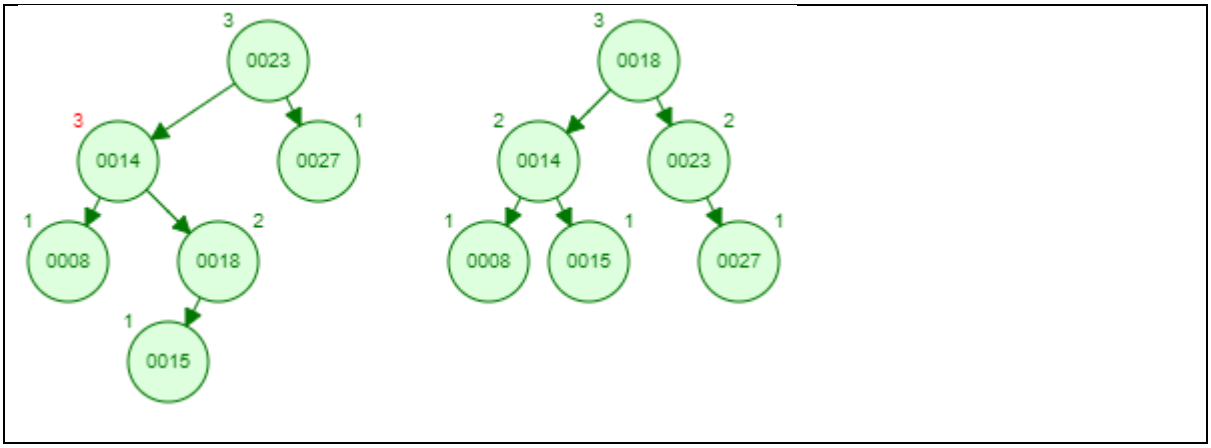
a) 30
b) 25
c) 18
d) 15
e) 8

Gabarito: D

Vamos simular! Vamos inserir logo os três primeiros: 23, 14 e 27 (primeira imagem); depois inserimos o número 8 (segunda imagem); e inserimos o número 18 (terceira imagem). Vejamos como ficou:



Agora vamos inserir o 15! Esse nó iria para a esquerda de 18 como mostra a primeira imagem abaixo. Nesse caso, o nó 23 ficaria desbalanceado. *O que fazer?* Temos um Caso Esquerda-Direita, portanto temos que fazer uma rotação dupla: primeiro à esquerda no Ramo 14-18 e depois à direita no Ramo 18-23. O resultado é mostrado a imagem abaixo e é na inserção do nó 15 que devemos fazer a primeira rotação.



38. (CESPE – 2014 – TJ/SE - Analista de Sistemas)

Existem dois vetores, chamados A e B, que estão ordenados e contêm N elementos cada, respeitando a propriedade A[N-1]<B[0], onde os índices de ambos os vetores vão de 0 a N-1. Retiram-se primeiro todos os elementos de A na ordem em que se apresentam e inserem-se esses elementos em uma árvore binária de busca, fazendo o mesmo depois com os elementos de B, que são inseridos na mesma árvore de busca que os de A. Depois, retiram-se os elementos da árvore em um percurso pós ordem, inserindo-os em uma pilha. Em seguida retiram-se os elementos da pilha, que são inseridos de volta nos vetores, começando pelo elemento 0 do vetor A e aumentando o índice em 1 a cada inserção, até preencher todas as N posições, inserindo, então, os N elementos restantes no vetor B da mesma maneira.

Ao final do processo, tem-se que os vetores:

a) estão ordenados e A[i] < B[i], para todo i=0,...., N-1.
b) estão ordenados e A[i] > B[i], para todo i=0,...., N-1.
c) estão ordenados e não existe mais uma propriedade que relacione A[i] e B[i].
d) não estão ordenados e A[i] < B[i], para todo i=0,...., N-1.
e) não estão ordenados e A[i] > B[i], para todo i=0,...., N-1.

Gabarito: A

Ele diz que nós temos dois vetores em que A[N-1] < B[0]. Então, vamos imaginá-los aqui:

- Vetor A = [1, 3, 5]
- Vetor B = [7, 9, 11]

Depois ele diz que são retirados todos os elementos de A na ordem que se apresentam e são inseridos em uma árvore binária de busca (lembre-se que uma árvore binária de busca é aquela em que todos os nós da subárvore esquerda possuem um valor numérico menor que o da raiz e os nós da subárvore direita possuem um valor numérico maior que o da raiz). Se você desenhar essa árvore, vai perceber que ela vai ficar em ordem toda para a direita - sem nenhum elemento para esquerda. Dito isso, ficou:

- 1, 3, 5, 7, 9, 11.

Depois ele disse que os elementos foram retirados da árvore em pós-ordem, ou seja, subárvore à esquerda, depois subárvore à direita e só depois raiz. Portanto, não tem elemento na esquerda, você retira o elemento da direita e depois a raiz. E são colocados em uma pilha, logo ficaria:

- 11, 9, 7, 5, 3, 1

Lembrando que numa pilha você insere sempre elementos no topo, logo 1 seria o topo da pilha. Depois ele diz que você retira os elementos da pilha (também sempre pelo topo) e coloca de volta nos vetores. Logo, ficaria:

- Vetor A = [1, 3, 5]
- Vetor B = [7, 9, 11]

Pronto! Note que fica exatamente a mesma coisa e os vetores ficariam ordenados e A[i] < B[i], para todo i=0,...., N-1. Bacana?

ACERTEI	ERREI

5.7 EXERCÍCIOS COMENTADOS: PESQUISA

1. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações / Análise de Sistemas)

Para poder ser aplicado, o algoritmo de pesquisa binária exige que os elementos do *array*:
a) sejam números;
b) estejam ordenados;
c) estejam representados em base múltipla de 2;
d) ocupem somente as posições pares;
e) não sejam repetidos.

Gabarito: **Letra B**.

O método de busca binária consiste em acessar o elemento no meio do array, e testar se é o elemento procurado. Se não for o elemento procurado, repete a operação no lado com elementos menores (se o elemento procurado for menor que o elemento no meio), ou maiores (caso seja maior).

Ora, para que exista um lado com elementos menores e maiores, é preciso que o array esteja ordenado. Portanto, **a busca binária só pode ser feita sobre vetores ordenados**!

Por isso, resposta correta, **Letra B**. Todas as outras condições não são limitações do método.

2. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

Considere que um algoritmo de pesquisa, em um arquivo previamente ordenado, é caracterizado por realizar comparação de chaves e sucessivas divisões no espaço de busca até encontrar o termo pesquisado ou até haver um único registro. Trata-se de um algoritmo de

a) pesquisa por interpolação.
b) pesquisa binária.
c) pesquisa sequencial.
d) árvore de busca binária.

Gabarito: **Letra B**.

a) pesquisa por interpolação.
ERRADO. A pesquisa por interpolação é uma melhoria feita no método de busca binária, quando a coleção, além de ordenada, tem os elementos com separados por uma diferença constante (ou quase constante) - por exemplo, um elemento é sempre 10 unidades maior que o anterior. Nesse caso, após 2 consultas é possível "adivinhar" a posição do elemento procurado.

b) pesquisa binária.
CERTO. A busca binária é feita sobre uma coleção ordenada, buscando o elemento no meio (pivô), e testando contra o elemento procurado. Se o elemento não for igual, procura na sub-coleção com elementos menores que o pivô, se for maior, procura na sub-coleção com elementos maiores.

c) pesquisa sequencial.
ERRADO. A pesquisa sequencial testa o valor procurado elemento por elemento, em ordem sequencial. A única vantagem sobre a busca binária é que funciona sobre coleções não-ordenadas.

d) árvore de busca binária.
ERRADO. A árvore de busca binária é uma estrutura de dados, não um algoritmo de pesquisa.

Portanto, a **Letra B** é a alternativa correta.

3. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Considere que na Defensoria há uma lista ordenada com o nome de 1000 cidadãos amazonenses. Utilizando o método de pesquisa binária para localizar o nome de um destes cidadãos, serão necessárias, no máximo,
a) 1.000 comparações.
b) 10 comparações.
c) 500 comparações.
d) 200 comparações.
e) 5 comparações.

Gabarito: **Letra B**

A busca binária busca no meio da lista ordenada. Se for menor, repete na sub-lista à esquerda, se for maior, na sub-lista à direita. Como a lista é sempre repartida pela metade, é efetuada, no pior caso, o primeiro número inteiro maior ou igual a log2(n). Como log2(1.000) = 9,966, teremos, no máximo, 10 comparações.

Eu acho mais fácil pensar na potência de 2 que é ligeiramente maior que o número de elementos. No caso, 2^10 = 1024 é a primeira potência de 2 maior que 1.000, portanto, são necessárias, no máximo, **10 comparações**.

4. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

Na pesquisa do tipo sequencial, há aumento do desempenho se a tabela estiver ordenada pelo valor da chave.

Gabarito: **CERTO**, mas MUITO polêmico...

Uma pesquisa sequencial, por padrão, independe da ordenação da coleção (no caso, da tabela). O algoritmo passa item por item sequencialmente testando a igualdade. Ela pode ser realizada em coleções desordenada, ou ordenadas. Portanto, **por essa análise, não há aumento do desempenho pela ordenação**.

Contudo, **é POSSÍVEL optimizar a busca caso a coleção estiver ordenada**: uma vez que eu encontre um valor maior do que o item que eu procuro, posso assumir que o item procurado não está presente na coleção.

Na minha opinião, descuido do examinador e da banca.

Reescrevendo de forma melhor:

Na pesquisa do tipo sequencial, ~~há~~ **é possível o** aumento do desempenho se a tabela estiver ordenada pelo valor da chave.

5. (CESGRANRIO / Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Infraestrutura)

Um método que implementa um algoritmo de busca binária recebe como parâmetros um vetor de inteiros ordenados descendentemente, o comprimento desse vetor e um número inteiro que se deseja localizar no vetor. O cabeçalho desse método é o seguinte:

public int buscaBin(int vet[], int n, int val)

Admitindo-se que o vetor passado como parâmetro tenha 750 elementos, qual será o número máximo de iterações que o algoritmo irá realizar até que o valor (val) seja localizado ou que seja detectado que esse valor não se encontra no vetor?

a) 8
b) 9
c) 10
d) 11
e) 12

Gabarito: **Letra C**.

O jeito "decorado" é saber que o número de iterações para buscar com a busca binária é $\log_2$ (N), arredondado para cima. Como N é 750, o resultado será 9,xxx. Arredondado para cima, temos 10.

Mas é fácil resolver mesmo sem fórmula!

A busca binária só ocorre sobre listas ordenadas. Esse método busca o elemento procurado no meio da estrutura e, caso seja menor, repete a busca na sub-lista com elementos menores, e caso seja maior, repete na sub-lista com elementos maiores. Em ambos os casos, o universo de elementos pesquisados sempre cai pela metade depois de um teste.

Sendo assim, no pior cenário, faríamos o 1o. teste, dividindo a lista em 374 elementos à esquerda, 1 testado, e 375 elementos à direita. Como é o pior cenário, não teríamos encontrado, e teríamos de procurar do lado maior com 375 elementos.

Novamente, procuraríamos entre 375, sendo 187 a esquerda, 1 testado, e 187 à direita. Não encontraríamos e teríamos que procurar em 187...

Em cada passo dividimos o universo na metade. Resumindo:
1a. iteração - sobram 375
2a. iteração - sobram 187
3a. iteração - sobram 93
4a. iteração - sobram 46

5a. iteração - sobram 23
6a. iteração - sobram 11
7a. iteração - sobram 5
8a. iteração - sobram 2
9a. iteração - sobram 1
10a. iteração - sobra 0

Após 10 iterações sabemos se o elemento está ou não na lista, portanto **Letra C**.

6. (CESGRANRIO / Escriturário (BB) / 2018)

Deseja-se realizar uma busca sobre um vetor não ordenado de inteiros. Para tal, deve-se criar um método Java que receba como parâmetros o vetor em questão e um número inteiro (elemento) que se deseja procurar no vetor, além de outros parâmetros que se julgarem necessários. Essa função deve retornar

• o índice do elemento no vetor, caso ele seja encontrado;
• o inteiro -1, caso o elemento não seja encontrado.

Assumindo-se que todos os pacotes necessários foram devidamente importados, qual método Java irá realizar corretamente essa busca?

a)

```
static int busca(int[] vet, int elem, int ini, int fim) {
    if (ini > fim)
        return -1;

    int m = (ini + fim) / 2;
    if (elem == vet[m])
        return m;
    else
        if (elem > vet[m])
            return busca(vet, elem, m + 1, fim);
        else
            return busca(vet, elem, ini, m - 1);
```

---

b)

```
public static int busca(int vet[], int elem) {
    for(int i=0; i < vet.length; i++)
        if(elem == vet[i])
            return i;
        else
            if(elem < vet[i])
                return -1;

    return -1;
```

---

c)

```
public static int busca(int[] vet, int elem) {
    int ini = 0, fim = vet.length-1;

    while(ini <= fim) {
        int m = (ini + fim) / 2;
        if (elem == vet[m])
            return m;
        else
            if (elem > vet[m])
                ini = m + 1;
            else
                fim = m - 1;
    }

    return -1;
```

---

d)

```
public static int busca(int vet[],int elem) {
    if(vet.length==0)
        return -1;

    if(elem==vet[vet.length-1])
        return vet.length-1;

    return busca(Arrays.copyOf(vet,vet.length-1),elem);
```

---

e)

```
public static int busca(int vet[], int elem) {
    if(vet.length == 0)
        return -1;

    if(elem == vet[vet.length-1])
        return vet.length-1;
    else
        if(elem > vet[vet.length-1])
            return -1;

    return busca(Arrays.copyOf(vet, vet.length-1), elem);
```

Gabarito: **Letra D**.

Analisando item por item:

Letra A) ERRADO. O algoritmo faz uma busca binária recursiva - Compara o elemento procurado com o elemento no meio. Se for menor, chama novamente o método para a sub-coleção na esquerda, se for maior, na direita.

Letra B) ERRADO. O algoritmo faz uma busca sequencial, mas considera a coleção ordenada, e para caso o elemento procurado seja menor que o elemento comparado.

Letra C) **ERRADO**. O algoritmo faz uma busca binária ajustando os intervalos ini e fim.

Letra D) **CERTO**. O algoritmo faz uma busca sequencial recursiva, de trás para frente, comparando o elemento procurado com o último elemento do array. Caso não encontre, chama novamente o método, passando um novo array sem a última posição (o que vai fazer com que se compare com o penúltimo, e assim sucessivamente).

Letra E) **ERRADO**. É semelhante ao algoritmo da Letra D, mas considera a coleção ordenada, pois se o elemento procurado for maior que o último, assume que a coleção não contém o elemento.

7. (CESGRANRIO / Engenheiro (PETROBRAS) / 2018 / Eletrônica / Equipamentos Júnior)

A função a seguir implementa um algoritmo de busca binária sobre um vetor de inteiros ordenado de modo ascendente.

```
int busca(int vet[], int elem, int ini, int fim) {
int m;
if(fim < ini)
  return -1;

m=(ini + fim) / 2;

System.out.println(vet[m]);

if(vet[m] == elem)
      return m;

if(vet[m] > elem)
        return busca(vet, elem, ini, m-1);

return busca(vet, elem, m+1, fim);
}
```

Essa função recebe como parâmetros um vetor (vet), o elemento que se deseja procurar no vetor (elem), o índice do primeiro elemento do vetor (ini) e o índice do último elemento do vetor (fim).

O comando System.out.println(vet[m]) exibe no console o valor do elemento de índice **m** do vetor **vet**.

Seja o seguinte vetor (vt) de inteiros:

20	25	27	38	51	57	60	65	73	74	78	80	83	88	90
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Suponha que a função busca seja chamada por meio do seguinte comando:

busca(vt, 39, 0, 14);

Qual será o 3o valor exibido no console?
a) 65
b) 51
c) 57
d) 38
e) 27

Gabarito: **Letra C**.

A função sempre imprime o valor no meio da sequência procurada, portanto basta verificar o elemento no meio na 3a. iteração. Assim:

Inicial: [20,25,27,38,51,57,60,65,73,74,78,80,83,88,90] - 15 posições

Iteração 1:
meio: 65
Como 39 é menor que 65, procura nos menores:
menores: [20,25,27,38,51,57,60]

Iteração 2:
meio: 38
Como 39 é maior que 38, procura nos maiores:
menores: [51,57,60]
Iteração 3:
meio: 57

Pronto! O elemento no meio, na 3a. iteração será o 57, **Letra C**.

8. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

Julgue o item a seguir, relativo aos conceitos de construção de algoritmos.

O algoritmo a seguir apresenta um exemplo de busca sequencial.

```
var a:vetor[1..5] de inteiro;
    temp:inteiro;
    i,k:inteiro;
início
    a[1]:=57;
    a[2]:=58;
    a[3]:=60;
    a[4]:=56;
    a[5]:=59;
    para i:=1 até 4 faça início
        para k:=i+1 até 5 faça início
            se a[i] > a[k] então início
               temp:=a[i];
               a[i]:=a[k];
               a[k]:=temp;
            fim se;
        fim para;
    fim para;
    escreva('Resultado: ');
    para i:=1 até 5 faça início
        escreva(a[i],',');
    fim para;
fim;
```

Gabarito: **ERRADO**.

Existem diversas evidências que contrariam a declaração de que esse seja um algoritmo de busca sequencial:

- Só são necessárias 2 variáveis para a busca sequencial: o vetor, e o elemento procurado. O algoritmo declara o vetor, e mais 3 variáveis de controle.
- Só é necessário percorrer o vetor 1 vez na busca sequencial (complexidade O(n)), e existem 2 loops aninhados no algoritmo ($O(n^2)$).
- Só é necessário apresentar 1 valor como resposta - seja o elemento, ou o índice do elemento procurado. O algoritmo imprime o vetor inteiro como resposta.

Por isso, rapidamente é possível dizer que o item está **ERRADO**.

Se quiser ir mais fundo...
Perceba que o que está acontecendo no algoritmo é o teste do valor em cada posição do array com as posições seguintes do array e, se estiverem fora da ordem crescente, trocam

a posição. Trata-se de um **algoritmo de ordenação** muito semelhante ao Bubble Sort, conhecido como **Exchange Sort.**

9. (CESPE / Agente Controlador de Arrecadação (AL) / 2002)

Os dados que são tratados por um sistema de computação podem possuir diferentes tipos de organização natural. Para lidar com essa diversidade de organizações naturais, diferentes tipos de estruturas de dados podem ser utilizados.

Acerca dos principais tipos de estruturas de dados, julgue o item subseqüente.

Operações de busca em uma árvore binária envolvem, no mínimo, N/2 operações de comparação, em que N é o número de elementos contidos na árvore binária.

Gabarito: **Errado**

A quantidade de comparações máxima em uma árvore binária $\log_2 N$, desde que ela esteja balanceada. Contudo, não há quantidade mínima! Se você der a sorte de procurar exatamente o valor que está no nó raiz, só acontecerá 1 teste!

RELEMBRANDO CONCEITOS DE ÁRVORES BINÁRIAS!

A árvore é uma estrutura recursiva, formada por um nó, uma sub-árvore esquerda, e outra sub-árvore à direita. A árvore binária tem todos os elementos menores que o nó na sub-árvore esquerda, e os maiores na sub-árvore à direita. A árvore binária é dita balanceada se, para qualquer nó da árvore, a sub-árvore à direita não tenha a altura diferente da sub-árvore à esquerda em mais de 1 unidade.

10. (FCC / Analista (DPE RS) / 2017 / Desenvolvimento de Sistemas / Tecnologia da Informação)

Um Analista, estudando a complexidade de algoritmos de busca linear (ou sequencial), concluiu corretamente que no pior caso, considerando um vetor de n elementos, este tipo de algoritmo tem complexidade

a) O(n).
b) O(log_2n-1).
c) O(√n).
d) O(log_2n).
e) O(log_2n^2).

Gabarito: **Letra A**.

O pior caso de uma busca linear em um vetor de *n* elementos é quando o elemento procurado é o último. Nesse caso, serão realizados N testes e, portanto, temos a complexidade O(n). Portanto, **Letra A** está correta.

Lembrando que no melhor caso da busca linear, o primeiro elemento testado já é o procurado, e só é realizado 1 teste. No caso médio, são realizados *n/2* testes (a complexidade permanece O(n)).

11. (FGV / Analista do Ministério Público (MPE AL) / 2018 / / Desenvolvimento de Sistemas)

A complexidade do algoritmo de busca binária, sobre uma lista indexada ordenada pela chave de busca, é

a) O (N)
b) O ($\log_2$ N)
c) O (2 $\log_2$ N)
d) O (N $\log_2$ N)
e) O (N^2)

Gabarito: **Letra B.**

Pessoal, cada iteração da busca binária reparte a lista na metade. No final de x iterações, teremos testado 2^x elementos. Se n = 2^x, x = $\log_2$n. Mas é mais fácil decorar:

- complexidade da busca binária: O(log N)
- complexidade da busca sequencial: O(N)

6 LISTA DE EXERCÍCIOS

6.1 LISTA DE EXERCÍCIOS: INTRODUÇÃO

1. (FGV – 2015 – DPE/MT – Analista de Sistemas)

No desenvolvimento de sistemas, a escolha de estruturas de dados em memória é especialmente relevante. Dentre outras classificações, é possível agrupar essas estruturas em lineares e não lineares, conforme a quantidade de sucessores e antecessores que os elementos da estrutura possam ter. Assinale a opção que apresenta, respectivamente, estruturas de dados lineares e não lineares.

a) Tabela de dispersão e fila.
b) Estrutura de seleção e pilha.
c) Pilha e estrutura de seleção.

d) Pilha e árvore binária de busca.
e) Fila e pilha.

2. (CESPE – 2010 – DETRAN/ES – Analista de Sistemas)
Um tipo abstrato de dados apresenta uma parte destinada à implementação e outra à especificação. Na primeira, são descritas, em forma sintática e semântica, as operações que podem ser realizadas; na segunda, os objetos e as operações são representados por meio de representação, operação e inicialização.

3. (CESPE – 2010 – TRT/RN – Analista de Sistemas)
O tipo abstrato de dados consiste em um modelo matemático (v,o), em que v é um conjunto de valores e o é um conjunto de operações que podem ser realizadas sobre valores.

4. (CESPE – 2010 – BASA – Analista de Sistemas)
A escolha de estruturas internas de dados utilizados por um programa pode ser organizada a partir de TADs que definem classes de objetos com características distintas.

5. (CESPE – 2010 – BASA – Analista de Sistemas)
A descrição dos parâmetros das operações e os efeitos da ativação das operações representam, respectivamente, os níveis sintático e semântico em que ocorre a especificação dos TDAs.

6. (FCC – 2010 – TRE/AM – Analista de Sistemas)
Em relação aos tipos abstratos de dados - TAD, é correto afirmar:

a) O TAD não encapsula a estrutura de dados para permitir que os usuários possam ter acesso a todas as operações sobre esses dados.

b) Na transferência de dados de uma pilha para outra, não é necessário saber como a pilha é efetivamente implementada.

c) Alterações na implementação de um TAD implicam em alterações em seu uso.

d) Um programador pode alterar os dados armazenados, mesmo que não tenha conhecimento de sua implementação.

e) TAD é um tipo de dados que esconde a sua implementação de quem o manipula.

7. (FCC – 2009 – TRE/PI – Analista de Sistemas)

Em relação a tipos abstratos de dados, é correto afirmar que:

a) o TAD não encapsula a estrutura de dados para permitir que os usuários possam ter acesso a todas as operações disponibilizadas sobre esses dados.

b) algumas pilhas admitem serem declaradas como tipos abstratos de dados.

c) filas não permitem declaração como tipos abstratos de dados.

d) os tipos abstratos de dados podem ser formados pela união de tipos de dados primitivos, mas não por outros tipos abstratos de dados.

e) são tipos de dados que escondem a sua implementação de quem o manipula; de maneira geral as operações sobre estes dados são executadas sem que se saiba como isso é feito.

ACERTEI	ERREI

6.2 LISTA DE EXERCÍCIOS: VETORES E MATRIZES

1. (FCC - 2009 - TJ-PA - Analista Judiciário - Tecnologia da Informação)

Considere uma estrutura de dados do tipo vetor. Com respeito a tal estrutura, é correto que seus componentes são, caracteristicamente,

a) heterogêneos e com acesso FIFO.
b) heterogêneos e com acesso LIFO.
c) heterogêneos e com acesso indexado-sequencial.
d) homogêneos e acesso não indexado.
e) homogêneos e de acesso aleatório por intermédio de índices.

2. (CETAP - 2010 - AL-RR - Analista de Sistemas)

Matrizes são estruturas de dados de n-dimensões. Por simplicidade, chamaremos de matrizes as matrizes bidimensionais numéricas (que armazenam números inteiros). Sendo assim, marque a alternativa INCORRETA.

a) Uma matriz de m linhas e n colunas contêm (m * n) dados.

b) Uma matriz pode ser representada utilizando listas ligadas.

c) A soma dos elementos de uma matriz pode ser calculada fazendo dois laços aninhados, um sobre as linhas e o outro sobre as colunas.

d) A soma de duas matrizes de m linhas e n colunas resulta em uma matriz de 2*m linhas e 2*n colunas.

e) O produto de duas matrizes de n linhas e n colunas resulta em uma matriz de n linhas e n colunas.

3. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Arquitetura de Tecnologia)

Os dados armazenados em uma estrutura do tipo matriz não podem ser acessados de maneira aleatória. Portanto, usa-se normalmente uma matriz quando o volume de inserção e remoção de dados é maior que o volume de leitura dos elementos armazenados.

4. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação)

Entre alguns tipos de estrutura de dados, podem ser citados os vetores, as pilhas e as filas.

5. (CESPE - 2011 - EBC - Analista - Engenharia de Software)

Vetores são utilizados quando estruturas indexadas necessitam de mais que um índice para identificar um de seus elementos.

6. (CESPE - 2010 - TRE-BA - Analista Judiciário - Análise de Sistemas)

Vetores podem ser considerados como listas de informações armazenadas em posição contígua na memória.

7. (CESPE - 2010 - TRE-BA - Analista Judiciário - Análise de Sistemas)

Uma posição específica de um vetor pode ser acessada diretamente por meio de seu índice.

8. (FCC - 2016 - Copergás - PE - Analista Tecnologia da Informação)

Considere o algoritmo a seguir, na forma de pseudocódigo:

```
Var n, i, j, k, x: inteiro
Var v: vetor[0..7] inteiro
Início
 v[0] ← 12
 v[1] ← 145
 v[2] ← 1
 v[3] ← 3
 v[4] ← 67
 v[5] ← 9
 v[6] ← 45
 n ← 8
 k ← 3
 x ← 0

 Para j ← n-1 até k passo -1 faça
   v[j] ← v[j - 1];
 Fim_para

 v[k] ← x;
Fim
```

Este pseudocódigo

a) exclui o valor contido na posição x do vetor v.
b) insere o valor de x entre v[k-1] e v[k] no vetor v.
c) exclui o valor contido na posição k do vetor v.
d) tentará, em algum momento, acessar uma posição que não existe no vetor.
e) insere o valor de k entre v[x] e v[x+1] no vetor v.

ACERTEI	ERREI

6.3 LISTA DE EXERCÍCIOS: LISTAS

1. (CESPE –Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação)

Considere uma estrutura de dados em que cada elemento armazenado apresenta ligações de apontamento com seu sucessor e com o seu predecessor, o que possibilita que ela seja percorrida em qualquer sentido. Trata-se de
a) uma fila.
b) um grafo.
c) uma lista duplamente encadeada.
d) uma pilha.

2. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

O tempo de busca de um elemento em uma lista duplamente encadeada é igual à metade do tempo da busca de um elemento em uma lista simplesmente encadeada.

3. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

Em algumas implementações, uma lista vazia pode ter um único nó, chamado de sentinela, nó cabeça ou header. Entre suas possíveis funções, inclui-se simplificar a implementação de algumas operações realizadas sobre a lista, como inserir novos dados, recuperar o tamanho da lista, entre outras.

4. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

Estruturas ligadas como listas encadeadas superam a limitação das matrizes que não podem alterar seu tamanho inicial.

5. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

As listas duplamente encadeadas diferenciam-se das listas simplesmente encadeadas pelo fato de, na primeira, os nós da lista formarem um anel com o último elemento ligado ao primeiro da lista.

6. (FCC - 2012 - TRE-SP - Analista Judiciário - Análise de Sistemas - E)

Numa lista singularmente encadeada, para acessar o último nodo é necessário partir do primeiro e ir seguindo os campos de ligação até chegar ao final da lista.

7. (CESPE - 2011 - EBC - Analista - Engenharia de Software)

Uma lista é uma coleção de elementos do mesmo tipo dispostos linearmente, que podem ou não seguir determinada organização. As listas podem ser dos seguintes tipos: de encadeamento simples, duplamente encadeadas e ordenadas.

8. (CESPE - 2009 - ANAC - Técnico Administrativo - Informática)

Em uma lista circular duplamente encadeada, cada nó aponta para dois outros nós da lista, um anterior e um posterior.

9. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação)

A principal característica de uma lista encadeada é o fato de o último elemento da lista apontar para o elemento imediatamente anterior.

10. (CESPE - 2009 - TCE-AC - Analista de Controle Externo - Processamentos de Dados)

Uma lista encadeada é uma coleção de nodos que, juntos, formam uma ordem linear. Se é possível os nodosse deslocarem em ambas as direções na lista, diz-se que se trata de uma lista simplesmente encadeada.

11. (CESPE - 2008 – HEMOBRÁS – Técnico de Informática)

Uma estrutura do tipo lista, em que é desejável percorrer o seu conteúdo nas duas direções indiferentemente, é denominado lista duplamente encadeada.

12. (CESPE - 2010 – TRE/MT – Analista de Sistemas – C)

Uma lista duplamente encadeada é uma lista em que o seu último elemento referencia o primeiro.

13. (CESPE - 2006 – SGA/AC – Analista de Sistemas)

O principal problema da alocação por lista encadeada é a fragmentação.

14. (CESPE - 2008 – MCT – Analista de Sistemas)

O armazenamento de arquivos em disco pode ser feito por meio de uma lista encadeada, em que os blocos de disco são ligados por ponteiros. A utilização de lista encadeada elimina completamente o problema de fragmentação interna.

15. (CESPE - 2009 – FINEP – Analista de Sistemas)

Uma lista encadeada é uma representação de objetos na memória do computador que consiste de uma sequência de células em que:

a) cada célula contém apenas o endereço da célula seguinte.

b) cada célula contém um objeto e o tipo de dados da célula seguinte.

c) o último elemento da sequência aponta para o próximo objeto que normalmente possui o endereço físico como not null.

d) cada célula contém um objeto de algum tipo e o endereço da célula seguinte.

e) a primeira célula contém o endereço da última célula.

16. (CESPE - 2010 – BASA – Analista de Sistemas)

Em uma lista encadeada, o tempo de acesso a qualquer um de seus elementos é constante e independente do tamanho da estrutura de dados.

17. (CESPE - 2010 – INMETRO – Analista de Sistemas – C)

Considere que Roberto tenha feito uso de uma lista encadeada simples para programar o armazenamento e o posterior acesso aos dados acerca dos equipamentos instalados em sua empresa. Considere, ainda, que, após realizar uma consulta acerca do equipamento X, Roberto precisou acessar outro equipamento Y que se encontrava, nessa lista, em posição anterior ao equipamento X. Nessa situação, pela forma como os ponteiros são implementados em uma lista encadeada simples, o algoritmo usado por Roberto realizou a consulta ao equipamento Y sem reiniciar a pesquisa do começo da lista.

18. (FCC - 2003 – TRE/AM – Analista de Sistemas)

Os dados contidos em uma lista encadeada estão:

a) ordenados seqüencialmente.
b) sem ordem lógica ou física alguma.
c) em ordem física e não, necessariamente, em ordem lógica.
d) em ordem lógica e, necessariamente, em ordem física.
e) em ordem lógica e não, necessariamente, em ordem física.

19. (FCC - 2010 – DPE/SP – Analista de Sistemas)

Uma estrutura de dados que possui três campos: dois ponteiros e campo de informação denomina-se:

a) lista encadeada dupla.
b) Lista encadeada simples.
c) pilha.
d) fila.
e) vetor.

20. (CESPE - 2010 – TRE/MT – Analista de Sistemas)

O algoritmo para inclusão de elementos em uma pilha é usado sem nenhuma alteração para incluir elementos em uma lista.

ACERTEI	ERREI

6.4 LISTA DE EXERCÍCIOS: FILAS

1. (CESPE – 2017 - Analista Judiciário (TRT 7ª Região))

A lógica FIFO (first-in first-out) é utilizada na estrutura de dados do tipo
a) pointer ou ponteiros.
b) queue ou filas.
c) stack ou pilhas.
d) array ou matrizes.

2. (IADES - Profissional de Suporte Técnico (CFM) / 2018 / / Assistente de Tecnologia da Informação)

A sigla FIFO refere-se a estruturas de dados do tipo fila. Como é o funcionamento em uma FIFO?
a) O primeiro objeto inserido na fila é o último a ser removido.
b) O primeiro objeto inserido na fila é também o primeiro a ser removido.

c) O último objeto inserido na fila é o primeiro a ser removido.
d) O programador irá definir a ordem de entrada e de saída dos objetos em uma FIFO.
e) Uma FIFO e uma LIFO possuem as mesmas características de entrada e de saída dos objetos.

3. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Análise de Sistemas)

Em um programa existe a necessidade de guardar todas as alterações feitas em determinado dado para que seja possível desfazer alterações feitas ao longo de toda a sua existência. Nessa situação, a estrutura de dados mais adequada para o armazenamento de todas as alterações citadas seria uma fila.

4. (CESPE - 2012 – TST – Analista de Sistemas – A)

As pilhas e as filas são estruturas de dados essenciais para os sistemas computacionais. É correto afirmar que a fila é conhecida como lista LIFO - Last In First Out.

5. (CESPE - 2012 - TRE-RJ - Técnico Judiciário - Programação de Sistemas)

As filas são estruturas com base no princípio LIFO (last in, first out), no qual os dados que forem inseridos primeiro na fila serão os últimos a serem removidos. Existem duas funções que se aplicam a todas as filas: PUSH, que insere um dado no topo da fila, e POP, que remove o item no topo da fila.

6. (FCC - 2012 - MPE-AP – Analista de Sistemas - A)

Nas estruturas de dados, devido às características das operações da fila, o primeiro elemento a ser inserido será o último a ser retirado. Estruturas desse tipo são conhecidas como LIFO.

7. (FCC - 2012 - MPE-AP – Analista de Sistemas - C)

Nas estruturas de dados, a fila é uma lista linear na qual as operações de inserção e retirada ocorrem apenas no início da lista.

8. (FCC - 2012 - TRE-SP - Analista Judiciário - Análise de Sistemas – D)

Pela definição de fila, se os elementos são inseridos por um extremo da lista linear, eles só podem ser removidos pelo outro.

9. (FCC - 2011 - TRT - 19ª Região (AL) - Analista Judiciário - Tecnologia da Informação)

FIFO refere-se a estruturas de dados do tipo:

a) fila.
b) árvore binária.
c) pilha.
d) matriz quadrada.
e) cubo.

10. (ESAF - 2010 - CVM - Analista de Sistemas - prova 2)

Uma fila é um tipo de lista linear em que:

a) as inserções são realizadas em um extremo e as remoções no outro extremo.
b) as inserções e remoções são realizadas em um mesmo extremo.
c) podem ser realizadas apenas inserções.
d) a inserção de um elemento requer a remoção de outro elemento.

e) a ordem de saída não corresponde à ordem de entrada dos elementos.

11. (CESPE - 2010 - DETRAN-ES - Analista de Sistemas)

No armazenamento de dados pelo método FIFO (first in - first out), a estrutura de dados é representada por uma fila, em cuja posição final ocorrem inserções e, na inicial, retiradas.

12. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação)

Entre alguns tipos de estrutura de dados, podem ser citados os vetores, as pilhas e as filas.

13. (CESPE - 2004 – SES/PA – Analista de Sistemas)

Uma estrutura mais geral que as pilhas e filas é o deque, em que as inserções, retiradas e acessos são permitidos em ambas as extremidades.

14. (CESPE - 2009 – TCE/AC – Analista de Sistemas – D)

Um deque (double ended queue) requer inserção e remoção no topo de uma lista e permite a implementação de filas com algum tipo de prioridade. A implementação de um deque, geralmente é realizada com a utilização de uma lista simplesmente encadeada.

15. (FCC - 2007 – TRT/23 – Analista de Sistemas)

Uma estrutura de dados com vocação de FIFO de duplo fim e que admite a rápida inserção e remoção em ambos os extremos é:

a) uma pilha.
b) uma splay tree.
c) um deque.
d) uma lista linear.
e) uma árvore AVL.

16. (CESPE - 2004 – PBV/RR - Analista de Sistemas)

As filas com prioridade são listas lineares nas quais os elementos são pares da forma (q_i, p_i), em que q é o elemento do tipo base e p é uma prioridade. Elas possuem uma política de fila do tipo FIFO (first in first out) entre os elementos de mesma prioridade.

17. (CESPE - 2004 – STJ – Analista de Sistemas)

A seguir, está representada corretamente uma operação de retirada em uma fila de nome f.

se f.começo = nil então
erro {fila vazia}
senão j ← f.começo.info

18. (FCC - 2016 - Prefeitura de Teresina - PI - Analista Tecnológico - Analista de Suporte Técnico)

Considerando uma estrutura de dados do tipo fila, e a seguinte sequência de comandos sobre essa fila (sendo que o comando Push representa uma inserção de elemento e o comando Pop representa uma exclusão de elemento) e considerando também que a fila estava inicialmente vazia:

Push 3, Push 5, Pop 3, Push 7, Pop 5, Push 9, Push 8

Após a execução dessa sequência de comandos, o conjunto de elementos que resulta na fila é:

a) 3 – 5 – 7 – 9 – 8.
b) 7 – 9 – 8 – 3 – 5.
c) 3 – 3 – 5 – 5 – 7 – 9 – 8.
d) 7 – 9 – 8.
e) 3 – 5 – 3 – 7 – 5 – 9 – 8.

19. (FCC - 2016 – TRT - 23ª REGIÃO (MT) - Técnico Judiciário - Tecnologia da Informação)

Estruturas de dados básicas, como as pilhas e filas, são usadas em uma gama variada de aplicações. As filas, por exemplo, suportam alguns métodos essenciais, como o:

a) enqueue(x), que insere o elemento x no fim da fila, sobrepondo o último elemento.

b) dequeue(), que remove e retorna o elemento do começo da fila; um erro ocorrerá se a fila estiver vazia.

c) push(x), que insere o elemento x no topo da fila, sem sobrepor nenhum elemento.

d) pop(), que remove o elemento do início da fila e o retorna, ou seja, devolve o último elemento inserido.

e) top(), que retorna o elemento do fim da fila sem removê-lo; um erro ocorrerá se a fila estiver vazia.

20. (FCC - 2017 – TRE/BA - Analista de Sistemas)

A estrutura que, além de ser similar à fila, é apropriada para ampliar as características desta, permitindo inserir e retirar elementos tanto do início quanto do fim da fila, é o(a):

a) árvore.
b) lista duplamente encadeada.
c) deque.
d) fila circular.
e) pilha

ACERTEI	ERREI

6.5 LISTA DE EXERCÍCIOS: PILHAS

1. (CESPE - Oficial Técnico de Inteligência / 2018 / / Área 8)

Pilha é uma estrutura de dados em que o último elemento a ser inserido será o primeiro a ser retirado.

2. (FUNDATEC - Analista Legislativo (ALERS) / 2018 / / Analista de Tecnologia da Informação e Comunicação)

Suponha que, após a criação de uma pilha vazia de números inteiros, a sequência de operações abaixo tenha sido executada. Quantos elementos esta pilha terá ao término da execução desta sequência?

PUSH(7); PUSH(5); PUSH(3); PUSH(3); POP(); CONSULTA(); PUSH(2); PUSH(1); POP(); POP(); PUSH(17); PUSH(33); POP(); CONSULTA(); POP(); POP(); CONSULTA(); POP(); PUSH(22); PUSH(80); POP(); CONSULTA(); POP(); POP(); PUSH(4);
a) 0.
b) 1.
c) 2.
d) 3.
e) 4.

3. (FGV - Analista do Ministério Público (MPE AL) / 2018 / / Administrador de Rede)

Considere as seguintes operações sobre uma estrutura de dados, inicialmente vazia, organizada na forma de pilhas (ou stack),

PUSH (10)
PUSH (2)
POP ()
POP ()
PUSH (6)

Assinale a opção que apresenta a lista de elementos armazenados na estrutura, após a execução das operações acima.
a) 10, 2, 6
b) 10, 2
c) 2, 6
d) 6
e) 2

4. (FGV - Analista Legislativo (ALERO) / 2018 / Banco de Dados / Tecnologia da Informação)

Considere uma pilha de latas de sardinhas na prateleira de um supermercado.

Assinale a estrutura de dados que mais se assemelha ao modo como essas latas são manuseadas.
a) Array.
b) Binary tree.
c) Hashing.
d) Linked list.
e) Stack.

5. (FUNRIO - Analista Legislativo (CM SJM) / 2018 / / Analista em Tecnologia)

Considere a estrutura de dados PILHA suportando três operações básicas, conforme definidas no quadro I abaixo.

Quadro I	
Operação	Significado
Push (SJM,e)	Insere um elemento qualquer e na pilha SJM.
Pop(SJM)	Remove o elemento de topo na pilha SJM.
Top(SJM)	Acessa, sem remover, o elemento do topo da pilha SJM.

Quadro II
Sequência de Operações
Push(SJM,HONDA)
Push(SJM,RENAULT)
Push(SJM,HYUNDAI)
Push(SJM,FIAT)
Top(SJM)
Push(SJM,Pop(SJM))
Push(SJM,VW)
Push(SJM,Top(SJM))
Pop(SJM)
Pop(SJM)

Considerando-se uma pilha SJM inicialmente vazia e a sequência de operações indicadas no quadro II, ao final das operações o elemento que se encontra no topo da pilha é:
a) VW.
b) FIAT.
c) HONDA.
d) RENAULT.
e) HYUNDAI.

6. (AOCP - Técnico em Gestão de Infraestrutura (SUSIPE) / 2018 / / Gestão de Informática)

Várias estruturas de dados podem ser utilizadas para armazenar dados de uma aplicação. Em relação ao assunto, assinale a alternativa correta.
a) Uma estrutura de dados do tipo pilha sempre retira os elementos que foram inseridos primeiro na estrutura.
b) Uma estrutura de dados do tipo lista utiliza a ideia do primeiro a chegar, primeiro a ser servido para inserir elementos.
c) Uma estrutura de dados do tipo fila sempre retira os elementos que entraram por último na fila.
d) Em uma estrutura de dados do tipo pilha, para retirar o elemento do topo da pilha, é necessário retirar o elemento base da pilha.
e) Uma estrutura de dados do tipo fila utiliza a ideia do primeiro a chegar, primeiro a ser servido.

7. (CESPE - 2011 - FUB - Analista de Tecnologia da Informação - Específicos)

As pilhas são listas encadeadas cujos elementos são retirados e acrescentados sempre ao final, enquanto as filas são listas encadeadas cujos elementos são retirados e acrescentados sempre no início.

8. (CESPE - 2013 - INPI - Analista de Planejamento - Desenvolvimento e Manutenção de Sistemas)

Na estrutura de dados do tipo lista, todo elemento novo que é introduzido na pilha torna-se o elemento do topo.

9. (CESPE - 2012 - TJ-RO - Analista Judiciário - Analista de Sistemas Suporte – E)

Visitas a sítios armazenadas em um navegador na ordem last-in-first-out é um exemplo de lista.

10. (ESAF - 2013 - DNIT - Analista Administrativo - Tecnologia da Informação)

Assinale a opção correta relativa às operações básicas suportadas por pilhas.

a) Push: insere um novo elemento no final da pilha.
b) Pop: adiciona elementos ao topo da pilha.
c) Pull: insere um novo elemento no interior da pilha.
d) Top: transfere o último elemento para o topo da pilha.
e) Top: acessa o elemento posicionado no topo da pilha.

11. (FCC - 2012 – TST - Analista de Sistemas – C)

As pilhas e as filas são estruturas de dados essenciais para os sistemas computacionais. É correto afirmar que a pilha é conhecida como lista FIFO - First In First Out.

12. (FCC - 2012 – TRE/CE - Analista de Sistemas)

Sobre pilhas é correto afirmar:

a) Uma lista LIFO (Last-In/First-Out) é uma estrutura estática, ou seja, é uma coleção que não pode aumentar e diminuir durante sua existência.

b) Os elementos na pilha são sempre removidos na mesma ordem em que foram inseridos.

c) Uma pilha suporta apenas duas operações básicas, tradicionalmente denominadas push (insere um novo elemento no topo da pilha) e pop (remove um elemento do topo da pilha).

d) Cada vez que um novo elemento deve ser inserido na pilha, ele é colocado no seu topo e, em qualquer momento, apenas aquele posicionado no topo da pilha pode ser removido.

e) Sendo P uma pilha e x um elemento qualquer, a operação Push(P,x) diminui o tamanho da pilha P, removendo o elemento x do seu topo.

13. (CESPE - 2011 - EBC - Analista - Engenharia de Software)

As pilhas, também conhecidas como listas LIFO ou PEPS, são listas lineares em que todas as operações de inserção e remoção de elementos são feitas por um único extremo da lista, denominado topo.

14. (VUNESP - 2011 - TJM-SP - Analista de Sistemas - Judiciário)

Lista do tipo LIFO (Last in, First Out) e lista do tipo FIFO (Firstin,First Out) são, respectivamente, características das estruturas de dados denominadas:

a) Fila e Pilha.
b) Pilha e Fila.
c) Grafo e Árvore.
d) Árvore e Grafo.
e) Árvore Binária e Árvore Ternária.

15. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Arquitetura de Tecnologia)

A definição da estrutura pilha permite a inserção e a eliminação de itens, de modo que uma pilha é um objeto dinâmico, cujo tamanho pode variar constantemente.

16. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Administração de Dados)

Na representação física de uma pilha sequencial, é necessário uso de uma variável ponteiro externa que indique a extremidade da lista linear onde ocorrem as operações de inserção e retirada de nós.

17. (CESPE - 2009 - ANAC - Técnico Administrativo - Informática)

As operações de inserir e retirar sempre afetam a base de uma pilha.

18. (FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação)

Pilha é uma estrutura de dados:

a) cujo acesso aos seus elementos segue tanto a lógica LIFO quanto a FIFO.
b) cujo acesso aos seus elementos ocorre de forma aleatória.
c) que pode ser implementada somente por meio de vetores.
d) que pode ser implementada somente por meio de listas.
e) cujo acesso aos seus elementos segue a lógica LIFO, apenas.

19. (CESPE - 2004 – STJ – Analista de Sistemas)

Em geral, em uma pilha só se admite ter acesso ao elemento localizado em seu topo. Isso se adapta perfeitamente à característica das seqüências em que só o primeiro componente é diretamente acessível.

20. (CESPE - 2004 – STJ – Analista de Sistemas)

A seguir, está representada corretamente uma operação de desempilhamento em uma pilha de nome p.

se p.topo = 0 então
nada {pilha vazia}
senão p.topo ← p.topo-1

21. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A)

O tipo nó é inadequado para implementar estruturas de dados do tipo pilha.

22. (FGV – 2015 – DPE/MT – Analista de Sistemas)

Assinale a opção que apresenta a estrutura de dados na qual o primeiro elemento inserido é o último a ser removido.

a) Árvore
b) Fila
c) Pilha
d) Grafo

e) Tabela de dispersão

23. (FCC – 2012 – MPE/AP – Técnico Ministerial - Informática)
Nas estruturas de dados,

a) devido às características das operações da fila, o primeiro elemento a ser inserido será o último a ser retirado. Estruturas desse tipo são conhecidas como LIFO.

b) as pilhas são utilizadas para controlar o acesso de arquivos que concorrem a uma única impressora.

c) a fila é uma lista linear na qual as operações de inserção e retirada ocorrem apenas no início da lista.

d) a pilha é uma lista linear na qual as operações de inserção e retirada são efetuadas apenas no seu topo.

e) devido às características das operações da pilha, o último elemento a ser inserido será o último a ser retirado. Estruturas desse tipo são conhecidas como FIFO.

ACERTEI	ERREI

6.6 LISTA DE EXERCÍCIOS: ÁRVORES

1. (CESGRANRIO - Técnico Científico (BASA) / 2018 / / Tecnologia da Informação)
Uma árvore binária completa de busca, isto é, uma árvore em que todos os níveis têm o máximo número de elementos, tem um total de N nós. O número máximo de comparações necessárias para encontrar um elemento nessa árvore é
a) N
b) N 2
c) log 2(N+1)
d) (N+1)*log 2(N+1)
e) \sqrt{(N 1)}

2. (CESGRANRIO - Analista (PETROBRAS) / 2018 / / Sistema Júnior)
A sequência de chaves 20 – 30 – 25 – 31 – 12 – 15 – 8 – 6 – 9 – 14 – 18 é organizada em uma árvore binária de busca. Em seguida, a árvore é percorrida em pré-ordem.
Qual é a sequência de nós visitados?
a) 6 – 9 – 8 – 14 – 18 – 15 – 12 – 25 – 31 – 30 – 20
b) 20 – 12 – 8 – 6 – 9 – 15 – 14 – 18 – 30 – 25 – 31
c) 6 – 8 – 9 – 12 – 14 – 15 – 18 – 20 – 25 – 30 – 31
d) 20 – 30 – 31 – 25 – 12 – 15 – 18 – 14 – 8 – 9 – 6
e) 6 – 8 – 9 – 14 – 15 – 18 – 12 – 25 – 30 – 31 – 20

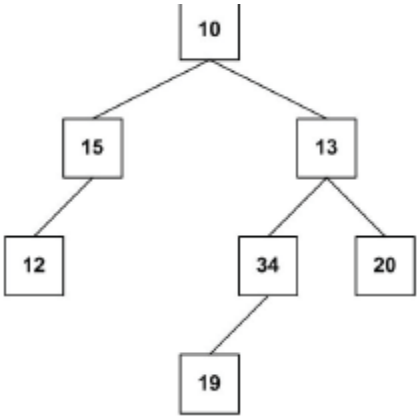
3. (CESGRANRIO - Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Infraestrutura)

Uma árvore binária foi percorrida em ordem simétrica, e os valores de seus nós exibidos no console. O resultado desse procedimento foi o seguinte:

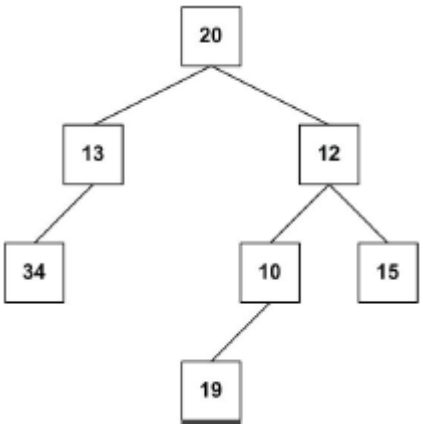
15 12 10 19 20 13 34

Dentre as árvores apresentadas, a única capaz de produzir o resultado acima é

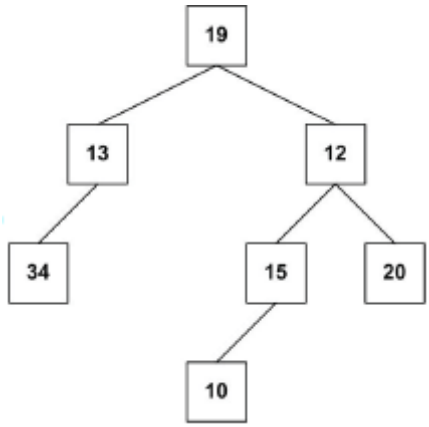
a)



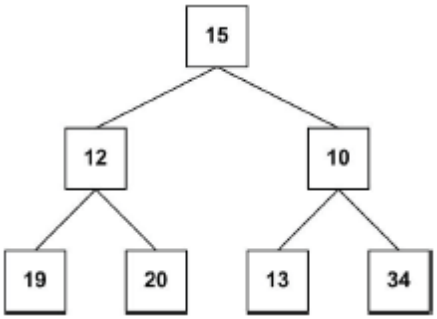
b)



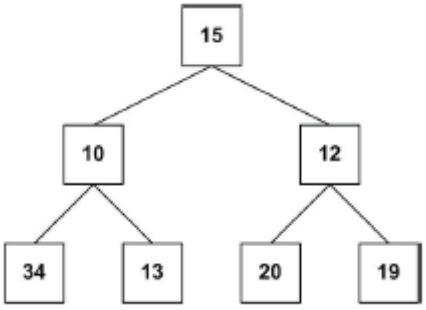
c)



d)



e)



4. (FCC - Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Certo documento possui 1 milhão de palavras não repetidas e foi editado em um editor de textos. Considerando que o editor de textos utiliza uma Árvore Binária de Busca − ABB de altura mínima para armazenar as palavras digitadas de forma a facilitar sua localização, para se localizar qualquer palavra nesta estrutura de dados serão necessárias, no máximo,

a) 1 milhão de comparações.
b) 20 comparações.
c) 32 comparações.
d) log101000000 comparações.
e) 2 milhões de comparações.

5. (CESGRANRIO - Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Processos de Negócio)

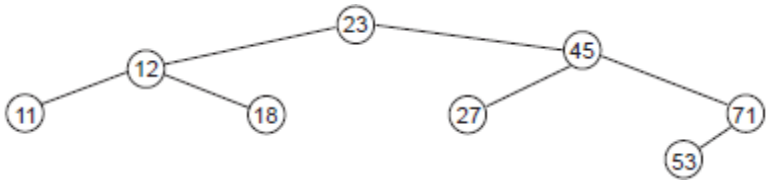
Considere uma árvore binária de busca (BST) com n (n>3) níveis (o nó raiz está no nível 1), 2n -1 nós e todas as chaves diferentes. Suponha, ainda, que algum dos pais de duas folhas seja removido da árvore e, mais tarde, uma chave com o mesmo valor da chave do nó removido seja inserida na árvore.

Quantas são as comparações necessárias para fazer a busca e encontrar o nó cuja chave foi removida e depois reinserida?

a) n – 2
b) n – 1
c) n
d) n +1
e) n + 2

6. (CESGRANRIO - Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Processos de Negócio)

Analise a árvore binária de busca (BST), abaixo, representada pelas chaves dos seus nós.



Qual é a sequência de chaves representativa do seu percurso em pré-ordem?

a) 11; 12; 18; 23; 27; 45; 53; 71
b) 11; 18; 12; 27; 53; 71; 45; 23
c) 23; 12; 11; 18; 45; 27; 71; 53
d) 53; 71; 27; 45; 18; 11; 12; 23
e) 23; 45; 71; 27; 53; 18; 12; 11

7. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

As operações de busca em uma árvore binária não a alteram, enquanto operações de inserção e remoção de nós provocam mudanças sistemáticas na árvore.

8. (CETAP - 2010 - AL-RR - Analista de Sistemas - A)

Uma árvore binária é aquela que tem como conteúdo somente valores binários.

9. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Administração de Dados)

O tipo de dados árvore representa organizações hierárquicas entre dados.

10. (CETAP - 2010 - AL-RR - Analista de Sistemas - B)

Uma árvore é composta por duas raízes, sendo uma principal e a outra secundária.

11. (CESPE - 2010 - DETRAN-ES - Analista de Sistemas)

Denomina-se árvore binária a que possui apenas dois nós.

12. (FUNCAB - 2010 - SEJUS-RO - Analista de Sistemas - II)

Árvores são estruturas de dados estáticas com sua raiz representada no nível um.

13. (CESPE - 2009 - ANAC - Especialista em Regulação - Economia)

Considerando-se uma árvore binária completa até o nível 5, então a quantidade de folhas nesse nível será 2^4.

14. (CESPE - 2009 – ANAC - Analista de Sistemas)

Uma árvore binária completa até o nível 10 tem 2.047 nós.

15. (FGV – 2015 – DPE/MT – Analista de Sistemas)

No desenvolvimento de sistemas, a escolha de estruturas de dados em memória é especialmente relevante. Dentre outras classificações, é possível agrupar essas estruturas em lineares e não lineares, conforme a quantidade de sucessores e

antecessores que os elementos da estrutura possam ter. Assinale a opção que apresenta, respectivamente, estruturas de dados lineares e não lineares.

a) Tabela de dispersão e fila.
b) Estrutura de seleção e pilha.
c) Pilha e estrutura de seleção.
d) Pilha e árvore binária de busca.
e) Fila e pilha.

16. (CESPE - 2010 – TRE/MT – Analista de Sistemas – B)

As listas, pilhas, filas e árvores são estruturas de dados que têm como principal característica a sequencialidade dos seus elementos.

17. (FCC - 2012 - MPE-AP - Analista Ministerial - Tecnologia da Informação – A)

A árvore é uma estrutura linear que permite representar uma relação de hierarquia. Ela possui um nó raiz e subárvores não vazias.

18. (CESPE - 2010 – TRE/MT - Analista de Sistemas – E)

O uso de recursividade é totalmente inadequado na implementação de operações para manipular elementos de uma estrutura de dados do tipo árvore.

19. (FCC - 2011 - TRT - 19ª Região (AL) - Técnico Judiciário - Tecnologia da Informação)

Em uma árvore binária, todos os nós têm grau:

a) 2.
b) 0, 1 ou 2.
c) divisível por 2.
d) maior ou igual a 2.
e) 0 ou 1.

20. (CESPE - 2011 – STM – Analista de Sistemas)

Enquanto uma lista encadeada somente pode ser percorrida de um único modo, uma árvore binária pode ser percorrida de muitas maneiras diferentes.

21. (FCC - 2016 - Prefeitura de Teresina - PI - Analista Tecnológico - Analista de Suporte Técnico)

Considerando a estrutura de dados denominada árvore,

a) a sua altura é definida como a profundidade média de todos os seus vértices.

b) um vértice com um ou dois filhos é denominado folha.

c) cada nó tem no mínimo dois filhos em uma árvore binária.

d) as folhas de uma árvore binária completa podem ter profundidades distintas entre si.

e) a profundidade de um vértice em uma árvore é definida como o comprimento da raiz da árvore até esse vértice.

22. (CESPE – 2017 – TRE/BA - Analista de Sistemas)

No estabelecimento de uma estrutura hierárquica, foi definida a seguinte árvore binária S:

S = (12(10(9(8))(11))(14(13)(15)))

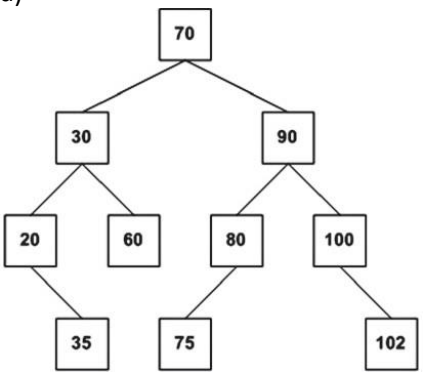
Considerando o resultado da operação de exclusão do nó 12, assinale a opção que corresponde a nova estrutura da árvore S.

a) (10(9(8))(11(14(13)(15)))
b) (11(9(8)(10))(14(13)(15)))
c) (11(10(9(8))(14(13)(15)))
d) (13(10(9)(11))(14(15)))
e) (13(11(9)(10))(14(15)))

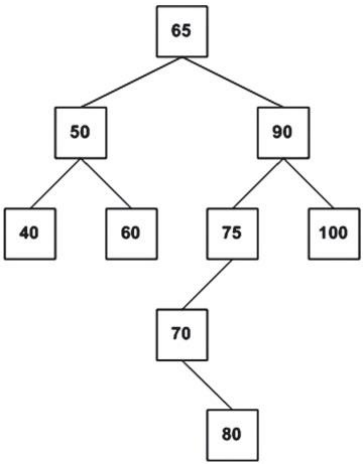
23. (CESGRANRIO – 2012 – PETROBRÁS - Analista de Sistemas)

Qual figura representa uma árvore AVL?

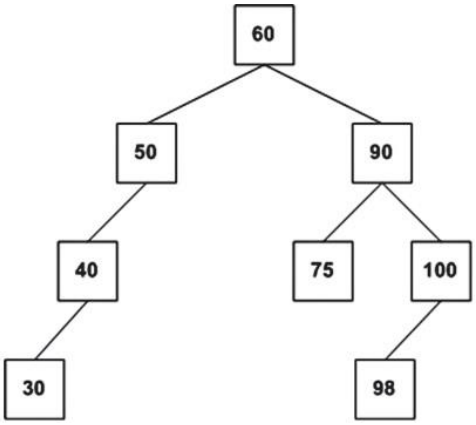
a)



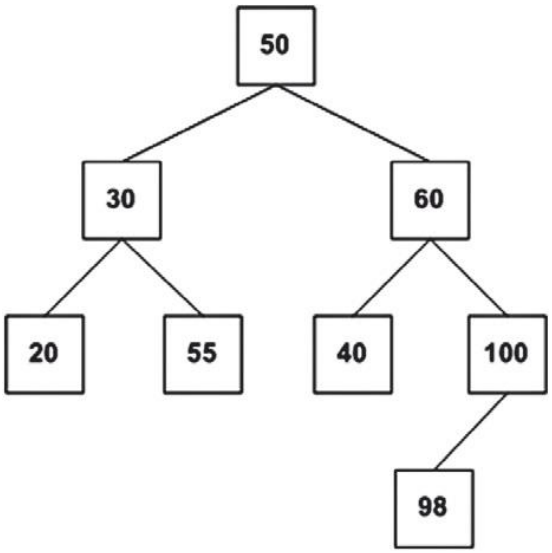
b)



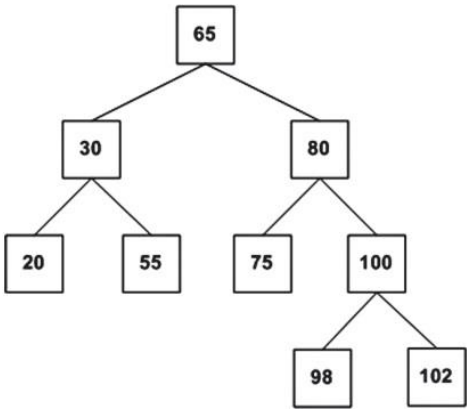
c)



d)

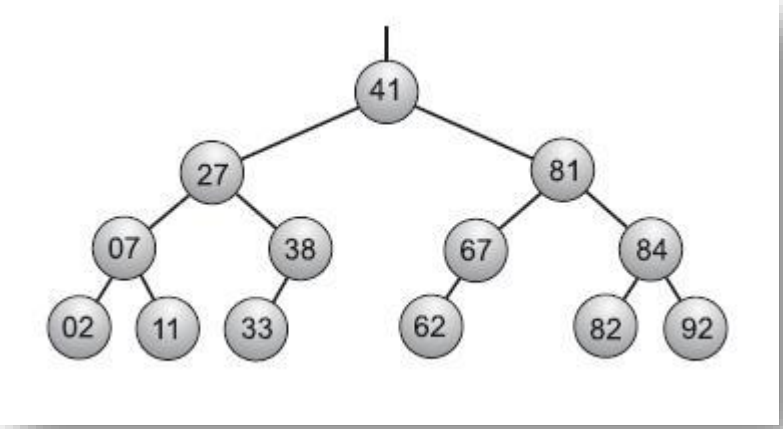


e)



24. (CESGRANRIO – 2006 – DECEA - Analista de Sistemas)

Suponha a seguinte árvore AVL.



A inserção do elemento 30 nessa árvore:

a) aumenta a profundidade da árvore após uma rotação.
b) provoca uma rotação à direita.
c) deixa os nós 02 e 07 no mesmo nível.
d) altera a raiz da árvore (nó 41).
e) torna o nó 33 pai do nó 27.

25. (CESPE – 2012 – TJ/RO - Analista de Sistemas)

Assinale a opção em que é apresentado exemplo de estrutura de informação do tipo abstrata, balanceada, não linear e com relacionamento hierárquico.

a) lista duplamente encadeada
b) árvore binária
c) pilha
d) árvore AVL
e) deque

26. (FCC – 2008 – TRT/18 - Analista de Sistemas)

Árvore AVL balanceada em altura significa que, para cada nó da árvore, a diferença entre as alturas das suas sub- árvores (direita e esquerda) sempre será:

a) menor ou igual a 2.
b) igual a 0 ou -1.
c) maior que 1.
d) igual a 1.
e) igual a -1, 0 ou 1.

27. (CESGRANRIO – 2010 – PETROBRÁS - Analista de Sistemas)

Uma árvore AVL é uma estrutura de dados muito usada para armazenar dados em memória. Ela possui algumas propriedades que fazem com que sua altura tenha uma relação muito específica com o número de elementos nela armazenados. Para uma folha, cuja altura é igual a um, tem-se uma árvore AVL com 6 nós.

Qual é a altura máxima que esta árvore pode ter?

a) 6
b) 5
c) 4
d) 3
e) 2

28. (CESGRANRIO – 2011 – PETROBRÁS - Analista de Sistemas)

Uma árvore AVL é uma árvore binária de busca autobalanceada que respeita algumas propriedades fundamentais. Como todas as árvores, ela tem uma propriedade chamada altura, que é igual ao valor da altura de sua raiz.

Sabendo que a altura de uma folha é igual a um e que a altura de um nó pai é igual ao máximo das alturas de seus filhos mais um, qual estrutura NÃO pode representar uma árvore AVL?

a) Uma árvore vazia
b) Uma árvore com dois nós
c) Uma árvore com três nós e altura igual a dois
d) Uma árvore com três nós e altura igual a três
e) Uma árvore com seis nós e altura igual a três

29. (CESGRANRIO – 2011 – PETROBRÁS - Analista de Sistemas)

Após a inserção de um nó, é necessário verificar cada um dos nós ancestrais desse nó inserido, relativamente à consistência com as regras estruturais de uma árvore AVL.

PORQUE

O fator de balanceamento de cada nó, em uma árvore AVL, deve pertencer ao conjunto formado por {-2, -1, 0, +1, +2}.

Analisando-se as afirmações acima, conclui-se que:

a) as duas afirmações são verdadeiras, e a segunda justifica a primeira.
b) as duas afirmações são verdadeiras, e a segunda não justifica a primeira.
c) a primeira afirmação é verdadeira, e a segunda é falsa.
d) a primeira afirmação é falsa, e a segunda é verdadeira.
e) as duas afirmações são falsas.

30. (CESGRANRIO – 2010 – EPE - Analista de Sistemas)

Um programador decidiu utilizar, em determinado sistema de análise estatística, uma árvore AVL como estrutura de dados. Considerando-se n a quantidade de elementos dessa árvore, o melhor algoritmo de pesquisa, com base em comparações, possui complexidade de tempo, no pior caso, igual a:

a) O(1)
b) O(log n).
c) Ω(n)
d) Ω(n log n)
e) Ω(n²)

31. (CESGRANRIO – 2012 – PETROBRÁS - Analista de Sistemas)

Todos os N nomes de uma lista de assinantes de uma companhia telefônica foram inseridos, em ordem alfabética, em três estruturas de dados: uma árvore binária de busca, uma árvore AVL e uma árvore B.

As alturas resultantes das três árvores são, respectivamente,

a) O(Log(N)), O(Log(N)), O(1)
b) O(Log(N)), O(N), O(Log(N))
c) O(N), O(Log(N)), O(1)
d) O(N), O(Log(N)), O(Log(N))
e) O(N), O(N), O(Log(N))

32. (IBFC – 2014 – TRE/AM - Analista de Sistemas)

Quanto ao Algoritmo e estrutura de dados no caso de árvore AVL (ou árvore balanceada pela altura), analise as afirmativas abaixo, dê valores Verdadeiro (V) ou Falso (F) e assinale a alternativa que apresenta a sequência correta de cima para baixo:

( ) Uma árvore AVL é dita balanceada quando, para cada nó da árvore, a diferença entre as alturas das suas sub-árvores (direita e esquerda) não é maior do que um.

( ) Caso a árvore não esteja balanceada é necessário seu balanceamento através da rotação simples ou rotação dupla.

Assinale a alternativa correta:

a) F-F
b) F-V
c) V-F
d) V-V

33. (CESGRANRIO – 2010 – PETROBRÁS - Analista de Sistemas)

Uma sequência desordenada de números armazenada em um vetor é inserida em uma árvore AVL. Após a inserção nesta árvore, é feito um percurso em ordem simétrica (em ordem) e o valor de cada nó visitado é inserido em uma pilha. Depois de todos os nós serem visitados, todos os números são retirados da pilha e apresentados na tela. A lista de números apresentada na tela está:

a) ordenada ascendentemente de acordo com os números.
b) ordenada descendentemente de acordo com os números.
c) na mesma ordem do vetor original.
d) na ordem inversa do vetor original.
e) ordenada ascendentemente de acordo com sua altura na árvore.

34. (FGV – 2009 – MEC - Analista de Sistemas)

Acerca das estruturas de dados Árvores, analise as afirmativas a seguir.

I. A árvore AVL é uma árvore binária com uma condição de balanço, porém não completamente balanceada.

II. Árvores admitem tratamento computacional eficiente quando comparadas às estruturas mais genéricas como os grafos.

III. Em uma Árvore Binária de Busca, todas as chaves da subárvore esquerda são maiores que a chave da raiz.

Assinale:

a) se somente a afirmativa I estiver correta.
b) se somente as afirmativas I e II estiverem corretas.
c) se somente as afirmativas I e III estiverem corretas.
d) se somente as afirmativas II e III estiverem corretas.
e) se todas as afirmativas estiverem corretas.

35. (CESPE – 2014 – TJ/SE - Analista de Sistemas)

Em uma árvore AVL (Adelson-Velsky e Landis), caso a diferença de altura entre as sub-árvores de um nó seja igual a 2 e a diferença de altura entre o nó filho do nó desbalanceado seja igual a -1, deve-se realizar uma rotação dupla com o filho para a direita e o pai para a esquerda a fim de que a árvore volte a ser balanceada.

36. (CESPE – 2010 – PETROBRÁS - Analista de Sistemas)

As árvores usadas como estruturas de pesquisa têm características especiais que garantem sua utilidade e propriedades como facilidade de acesso aos elementos procurados em cada instante. A esse respeito, considere as afirmações abaixo.

I - A árvore representada na figura (I) acima não é uma árvore AVL, pois as folhas não estão no mesmo nível.

II - A sequência 20, 30, 35, 34, 32, 33 representa um percurso sintaticamente correto de busca do elemento 33 em uma árvore binária de busca.

III - A árvore representada na figura (II) acima é uma árvore binária, apesar da raiz não ter filhos.

É (São) correta(s) APENAS a(s) afirmativa(s):

a) I.
b) II.
c) III.
d) I e II.
e) II e III.

37. (CESPE – 2010 – PETROBRÁS - Analista de Sistemas)

No sistema de dados do Departamento de Recursos Humanos de uma grande empresa multinacional, os registros de funcionários são armazenados em uma estrutura de dados do tipo árvore binária AVL, onde cada registro é identificado por uma chave numérica inteira. Partindo de uma árvore vazia, os registros cujas chaves são 23, 14, 27, 8, 18, 15, 30, 25 e 32 serão, nessa ordem, adicionados à árvore.

Dessa forma, o algoritmo de inserção na árvore AVL deverá realizar a primeira operação de rotação na árvore na ocasião da inserção do elemento:

a) 30
b) 25
c) 18
d) 15
e) 8

38. (CESPE – 2014 – TJ/SE - Analista de Sistemas)

Existem dois vetores, chamados A e B, que estão ordenados e contêm N elementos cada, respeitando a propriedade A[N-1]<B[0], onde os índices de ambos os vetores vão de 0 a N-1. Retiram-se primeiro todos os elementos de A na ordem em que se apresentam e inserem-se esses elementos em uma árvore binária de busca, fazendo o mesmo depois com os elementos de B, que são inseridos na mesma árvore de busca que os de A. Depois, retiram-se os elementos da árvore em um percurso pós ordem, inserindo-os em uma pilha. Em seguida retiram-se os elementos da pilha, que são inseridos de volta nos vetores, começando pelo elemento 0 do vetor A e aumentando o índice em 1 a cada inserção, até preencher todas as N posições, inserindo, então, os N elementos restantes no vetor B da mesma maneira.

Ao final do processo, tem-se que os vetores:

a) estão ordenados e A[i] < B[i], para todo i=0,...., N-1.
b) estão ordenados e A[i] > B[i], para todo i=0,...., N-1.
c) estão ordenados e não existe mais uma propriedade que relacione A[i] e B[i].
d) não estão ordenados e A[i] < B[i], para todo i=0,...., N-1.
e) não estão ordenados e A[i] > B[i], para todo i=0,...., N-1.

ACERTEI	ERREI

6.7 LISTA DE EXERCÍCIOS: COMPLEXIDADE DE ALGORITMOS

12. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações - WEB Mobile / Análise de Sistemas)

Para projetar algoritmos eficientes um desenvolvedor deve estar preocupado com a complexidade deste algoritmo, desde sua concepção.

Considere a seguinte função T(n) que mede os recursos (ex. tempo de execução) que um algoritmo necessita no pior caso para processar uma entrada qualquer de tamanho n:

T(n) = O(log(n))

Sabendo que O(log(n)) é a ordem da complexidade de tempo do algoritmo seguindo a notação "big O", é correto afirmar que este algoritmo tem complexidade de ordem:
a) constante;
b) sublinear;
c) linear;
d) polinomial;

e) exponencial.

13. (FCC / Analista Judiciário (TRF 5ª Região) / 2017 / Informática - Desenvolvimento / Apoio Especializado)

Considere o algoritmo abaixo.

```
static int fibonacci(int n) {
  if (n <= 1) {
     return n;
  }
  return fibonacci(n - 2) + fibonacci(n - 1);
}
```

A complexidade deste algoritmo, na notação Big O, é
a) $O(2^n)$.
b) $O(n^2)$.
c) O(n).
d) O(log(n)).
e) $O(n^4)$.

14. (FGV / Analista Legislativo (ALERO) / 2018 / Infraestrutura de Redes e Comunicação / Tecnologia da Informação)

Considere a Sequência de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, ...), onde os dois primeiros termos valem 0 e 1 respectivamente, e cada termo seguinte é a soma de seus dois predecessores.

O pseudocódigo a seguir apresenta um algoritmo simples para o cálculo do N-ésimo termo dessa sequência.

```
function fibo (N)
if n = 1 then
     return 0
elif n = 2 then
     return 1
else
       penultimo := 0
       ultimo := 1
       for i := 3 until N do
            atual := penultimo + ultimo
            penultimo := ultimo
            ultimo := atual
       end for
       return atual
end if
```

Assinale a opção que mostra a complexidade desse algoritmo.
a) O(n/2)
b) O(n)

c) O(n2)
d) O(log n)
e) O(2n)

6.8 LISTA DE EXERCÍCIOS: ORDENAÇÃO

15. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações / Análise de Sistemas)

O algoritmo de ordenação baseado em vários percursos sobre o *array*, realizando, quando necessárias, trocas entre pares de elementos consecutivos denomina-se método:
a) das trocas (*exchange sort*);
b) da inserção (*insertion sort*);
c) da bolha (*bubble sort*);
d) da seleção (*selection sort*);
e) da permuta (*permutation sort*).

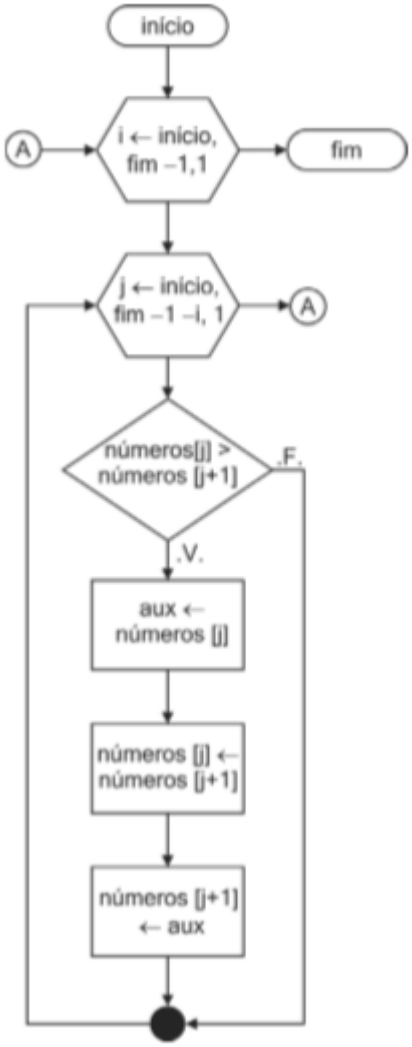
16. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações - WEB Mobile / Análise de Sistemas)

Considere o seguinte algoritmo, responsável por realizar a ordenação de um array de dados.

```
public int[] mySortingAlgorithm (int[] date){
  int size = data.length;
  int tmp = 0;
  for(int i = 0;i<size;i++){
   for(int j = (size-1);j>=(i+1);j--){
     if(data[j]<data[j-1]){
        tmp = data[j];
        data[j]=data[j-1];
        data[j-1]=tmp;
      }
    }
  }
  return data;
}
```

Podemos afirmar que o método de ordenação utilizado pelo algoritmo é o:
a) quickSort;
b) insertionSort;
c) mergeSort;
d) shellSort;
e) bubbleSort.

17. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)



Se, no fluxograma precedente, *início* indica o primeiro elemento do vetor e *fim*, o último elemento, então, para o vetor [11,6,2,7,8,3,5], o resultado final é
a) [2,7,3,5].
b) [11,7,3,5].
c) [11,6,2,7,8,3,5].
d) [2,3,5,6,7,8,11].
e) [6,2,8,3,5].

18. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Para responder à questão a seguir, considere a estratégia de ordenação apresentada em Java abaixo.

```
 private static void ordena(int[] vetor, int inicio, int fim) {
     if (inicio < fim) {
             int posicaoP = separar(vetor, inicio, fim);
             ordena(vetor, inicio, posicaoP - 1);
             ordena(vetor, posicaoP + 1, fim);
     }
}
```

```
private static int separar(int[] vetor, int inicio, int fim) {
     int P = vetor[inicio];
     int i = inicio + 1, f = fim;
     while (i <= f) {
             if (vetor[i] <= P)
                     i++;
             else if (P < vetor[f])
                     f--;
             else { int troca = vetor[i];
                     vetor[i] = vetor[f];
                     vetor[f] = troca;
                     i++;
                     f--;
             }
     }
     vetor[inicio] = vetor[f];
     vetor[f] = P;
     return f;
}
```

A estratégia apresentada em Java é o método de ordenação
a) Bubblesort.
b) Mergesort.
c) Insertion Sort.
d) Quicksort.
e) Heapsort.

19. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Para responder à questão a seguir, considere a estratégia de ordenação apresentada em Java abaixo.

```
 private static void ordena(int[] vetor, int inicio, int fim) {
     if (inicio < fim) {
             int posicaoP = separar(vetor, inicio, fim);
             ordena(vetor, inicio, posicaoP - 1);
             ordena(vetor, posicaoP + 1, fim);
     }
}

private static int separar(int[] vetor, int inicio, int fim) {
     int P = vetor[inicio];
     int i = inicio + 1, f = fim;
     while (i <= f) {
             if (vetor[i] <= P)
                     i++;
             else if (P < vetor[f])
                     f--;
             else { int troca = vetor[i];
```

```
                        vetor[i] = vetor[f];
                        vetor[f] = troca;
                        i++;
                        f--;
                }
        }
        vetor[inicio] = vetor[f];
        vetor[f] = P;
        return f;
}
```

Considerando que N é número de elementos do vetor a ser ordenado, a estratégia de ordenação apresentada em Java
a) também é conhecida como método de ordenação por intercalação e possui uma versão para unir dois vetores já ordenados.
b) tem complexidade $O(N^2)$ no pior caso e no caso médio, mas apresenta complexidade O(N) no melhor caso.
c) faz um número fixo de comparações dado por $\log_2 N$, independente dos valores do vetor original. Isso é garantido pelas chamadas recursivas ao método ordena().
d) utiliza o método separar() para dividir o vetor original em 2 sublistas de igual tamanho. Isso garante que mesmo no pior caso o método realize Nlog2N comparações.
e) utiliza o método separar() para fazer a partição do vetor, por meio da seleção de um elemento chamado pivô. A escolha do pivô é crucial para o bom desempenho do método, já que a fase de partição é a parte crítica do algoritmo.

20. (CESGRANRIO / Analista (PETROBRAS) / 2018 / / Sistema Júnior)

Dada a sequência numérica (15,11,16,18,23,5,10,22,21,12) para ordenar pelo algoritmo Selection Sort, qual é a sequência parcialmente ordenada depois de completada a quinta passagem do algoritmo?
a) [15, 11, 16, 18, 12, 5, 10, 21, 22, 23]
b) [15, 11, 5, 10, 12, 16, 18, 21, 22, 23]
c) [15, 11, 16, 10, 12, 5, 18, 21, 22, 23]
d) [10, 11, 5, 12, 15, 16, 18, 21, 22, 23]
e) [12, 11, 5, 10, 15, 16, 18, 21, 22, 23]

21. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

O método de ordenação conhecido como quick sort utiliza o maior elemento, o qual é sempre colocado ao final do vetor, para garantir que a ordenação seja realizada em ordem decrescente.

22. (AOCP / Técnico em Gestão de Infraestrutura (SUSIPE) / 2018 / / Gestão de Informática)

Assinale a alternativa correta a respeito dos principais algoritmos de ordenação.
a) O algoritmo de ordenação QuickSort é um algoritmo eficiente, porém deve ser implementado visando a uma boa escolha do elemento pivô.

b) O algoritmo de ordenação por inserção tem uma implementação cara, porém com um desempenho estável.
c) O algoritmo de ordenação HeapSort é um algoritmo eficiente em relação ao tempo, porém ineficiente em relação à memória.
d) O algoritmo de ordenação ShellSort tem sua principal desvantagem quando os dados a serem ordenados estão parcialmente ordenados.
e) O algoritmo de ordenação por seleção tem uma implementação cara, porém com desempenho estável.

23. (CESGRANRIO / Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Processos de Negócio)

Analise o algoritmo de ordenação que se segue.

```
def ordenar(dado):
    for passnum in range(len(dado)-1,0,-1):
      for i in range(passnum):
          if dado[i]>dado[i+1]:
             temp = dado[i]
             dado[i] = dado[i+1]
             dado[i+1] = temp
dado = [16,18,15,37,13]
ordenar(dado)
print(dado)
```

Com o uso desse algoritmo, qual é a quantidade de trocas realizadas para ordenar a sequência **dado**?
a) 4
b) 5
c) 6
d) 7
e) 8

24. (CESGRANRIO / Escriturário (BB) / 2018)

O programa a seguir, em Python, implementa o algoritmo do método de bolha, imprimindo o resultado de cada passo.

```
def bolha(lista):
    for passo in range(len(lista)-1,0,-1):
         for i in range(passo):
               if lista[i]>lista[i+1]:
                     lista[i],lista[i+1]=lista[i+1],lista[i]
         print(lista)
```

Qual será a quarta linha impressa para a chamada **bolha([4, 3, 1, 9, 8, 7, 2, 5])** ?
a) [3, 1, 4, 8, 7, 2, 5, 9]
b) [1, 3, 4, 7, 2, 5, 8, 9]
c) [1, 2, 3, 4, 5, 7, 8, 9]
d) [1, 3, 2, 4, 5, 7, 8, 9]

e) [1, 3, 4, 2, 5, 7, 8, 9]

25. (FCC / Analista Judiciário (TRF 5ª Região) / 2017 / Informática - Desenvolvimento / Apoio Especializado)

O algoritmo QuickSort usa uma técnica conhecida por divisão e conquista, onde problemas complexos são reduzidos em problemas menores para se tentar chegar a uma solução.

A complexidade média deste algoritmo em sua implementação padrão e a complexidade de pior caso são, respectivamente,

a) $O(n-1)$ e $O(n^3)$.
b) $O(n^2)$ e $O(n \log n^2)$.
c) $O(n^2)$ e $O(n^3)$.
d) $O(n)$ e $O(n^2)$.
e) $O(n \log n)$ e $O(n^2)$.

26. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Para ordenar um vetor com N elementos, o método de ordenação Seleção (Selection Sort) faz o seguinte número de comparações:
a) $(N^2 - N)/2$, sendo muito lento e inadequado para valores grandes de N.
b) $\log_2(N^2 + N)$ no melhor caso.
c) $(N^2 + N - 1)/2$ no caso médio, ficando lento para valores grandes de N.
d) $(N - 1)$ quando o vetor já está originalmente ordenado.
e) $(N^2 + N)/4$ no pior caso, sendo melhor que o pior caso do Bolha (Bubble Sort) pois faz menos trocas.

27. (CESGRANRIO - 2010 – BACEN – Analista de Sistemas)

Uma fábrica de software foi contratada para desenvolver um produto de análise de riscos. Em determinada funcionalidade desse software, é necessário realizar a ordenação de um conjunto formado por muitos números inteiros. Que algoritmo de ordenação oferece melhor complexidade de tempo (Big O notation) no pior caso?

a) Merge sort
b) Insertion sort
c) Bubble sort
d) Quick sort
e) Selection sort

28. (FGV - 2013 – MPE/MS – Analista de Sistemas)

Assinale a alternativa que indica o algoritmo de ordenação capaz de funcionar em tempo O(n) para alguns conjuntos de entrada.

a) Selectionsort (seleção)
b) Insertionsort (inserção)
c) Merge sort
d) Quicksort
e) Heapsort

29. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – III)

O tempo médio do QuickSort é de ordem igual ao tempo médio do MergeSort.

6.9 EXERCÍCIOS COMENTADOS: PESQUISA

30. (FGV / Analista Censitário (IBGE) / 2017 / Desenvolvimento de Aplicações / Análise de Sistemas)

Para poder ser aplicado, o algoritmo de pesquisa binária exige que os elementos do *array*:
a) sejam números;
b) estejam ordenados;
c) estejam representados em base múltipla de 2;
d) ocupem somente as posições pares;
e) não sejam repetidos.

31. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

Considere que um algoritmo de pesquisa, em um arquivo previamente ordenado, é caracterizado por realizar comparação de chaves e sucessivas divisões no espaço de busca até encontrar o termo pesquisado ou até haver um único registro. Trata-se de um algoritmo de

a) pesquisa por interpolação.
b) pesquisa binária.
c) pesquisa sequencial.
d) árvore de busca binária.

32. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Considere que na Defensoria há uma lista ordenada com o nome de 1000 cidadãos amazonenses. Utilizando o método de pesquisa binária para localizar o nome de um destes cidadãos, serão necessárias, no máximo,
a) 1.000 comparações.
b) 10 comparações.
c) 500 comparações.
d) 200 comparações.
e) 5 comparações.

33. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

Na pesquisa do tipo sequencial, há aumento do desempenho se a tabela estiver ordenada pelo valor da chave.

34. (CESGRANRIO / Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Infraestrutura)

Um método que implementa um algoritmo de busca binária recebe como parâmetros um vetor de inteiros ordenados descendentemente, o comprimento desse vetor e um número inteiro que se deseja localizar no vetor. O cabeçalho desse método é o seguinte:

public int buscaBin(int vet[], int n, int val)

Admitindo-se que o vetor passado como parâmetro tenha 750 elementos, qual será o número máximo de iterações que o algoritmo irá realizar até que o valor (val) seja localizado ou que seja detectado que esse valor não se encontra no vetor?

a) 8
b) 9
c) 10
d) 11
e) 12

35. (CESGRANRIO / Escriturário (BB) / 2018)

Deseja-se realizar uma busca sobre um vetor não ordenado de inteiros. Para tal, deve-se criar um método Java que receba como parâmetros o vetor em questão e um número inteiro (elemento) que se deseja procurar no vetor, além de outros parâmetros que se julgarem necessários. Essa função deve retornar

• o índice do elemento no vetor, caso ele seja encontrado;
• o inteiro -1, caso o elemento não seja encontrado.

Assumindo-se que todos os pacotes necessários foram devidamente importados, qual método Java irá realizar corretamente essa busca?

a)

```
static int busca(int[] vet, int elem, int ini, int fim) {
    if (ini > fim)
        return -1;

    int m = (ini + fim) / 2;
    if (elem == vet[m])
        return m;
    else
        if (elem > vet[m])
            return busca(vet, elem, m + 1, fim);
        else
            return busca(vet, elem, ini, m - 1);
```

---

b)

```
public static int busca(int vet[], int elem) {
    for(int i=0; i < vet.length; i++)
        if(elem == vet[i])
            return i;
        else
            if(elem < vet[i])
                return -1;

    return -1;
```

---

c)

```
public static int busca(int[] vet, int elem) {
    int ini = 0, fim = vet.length-1;

    while(ini <= fim) {
            int m = (ini + fim) / 2;
            if (elem == vet[m])
                    return m;
            else
                    if (elem > vet[m])
                            ini = m + 1;
                    else
                            fim = m - 1;
    }

    return -1;
```

---

d)

```
public static int busca(int vet[],int elem) {
    if(vet.length==0)
            return -1;

    if(elem==vet[vet.length-1])
            return vet.length-1;

    return busca(Arrays.copyOf(vet,vet.length-1),elem);
```

---

e)

```
public static int busca(int vet[], int elem) {
    if(vet.length == 0)
            return -1;

    if(elem == vet[vet.length-1])
            return vet.length-1;
    else
            if(elem > vet[vet.length-1])
                    return -1;

    return busca(Arrays.copyOf(vet, vet.length-1), elem);
```

---

36. (CESGRANRIO / Engenheiro (PETROBRAS) / 2018 / Eletrônica / Equipamentos Júnior)

A função a seguir implementa um algoritmo de busca binária sobre um vetor de inteiros ordenado de modo ascendente.

```
int busca(int vet[], int elem, int ini, int fim) {
int m;
if(fim < ini)
  return -1;
```

```
m=(ini + fim) / 2;

System.out.println(vet[m]);

if(vet[m] == elem)
      return m;

if(vet[m] > elem)
        return busca(vet, elem, ini, m-1);

return busca(vet, elem, m+1, fim);
}
```

Essa função recebe como parâmetros um vetor (vet), o elemento que se deseja procurar no vetor (elem), o índice do primeiro elemento do vetor (ini) e o índice do último elemento do vetor (fim).

O comando System.out.println(vet[m]) exibe no console o valor do elemento de índice **m** do vetor **vet**.

Seja o seguinte vetor (vt) de inteiros:

20	25	27	38	51	57	60	65	73	74	78	80	83	88	90
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Suponha que a função busca seja chamada por meio do seguinte comando:

busca(vt, 39, 0, 14);

Qual será o 3o valor exibido no console?
a) 65
b) 51
c) 57
d) 38
e) 27

37. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

Julgue o item a seguir, relativo aos conceitos de construção de algoritmos.

O algoritmo a seguir apresenta um exemplo de busca sequencial.

```
var a:vetor[1..5] de inteiro;
    temp:inteiro;
    i,k:inteiro;
início
    a[1]:=57;
    a[2]:=58;
    a[3]:=60;
    a[4]:=56;
    a[5]:=59;
    para i:=1 até 4 faça início
        para k:=i+1 até 5 faça início
            se a[i] > a[k] então início
               temp:=a[i];
               a[i]:=a[k];
               a[k]:=temp;
            fim se;
        fim para;
    fim para;
    escreva('Resultado: ');
    para i:=1 até 5 faça início
        escreva(a[i],',');
    fim para;
fim;
```

38. (CESPE / Agente Controlador de Arrecadação (AL) / 2002)

Os dados que são tratados por um sistema de computação podem possuir diferentes tipos de organização natural. Para lidar com essa diversidade de organizações naturais, diferentes tipos de estruturas de dados podem ser utilizados.

Acerca dos principais tipos de estruturas de dados, julgue o item subseqüente.

Operações de busca em uma árvore binária envolvem, no mínimo, N/2 operações de comparação, em que N é o número de elementos contidos na árvore binária.

39. (FCC / Analista (DPE RS) / 2017 / Desenvolvimento de Sistemas / Tecnologia da Informação)

Um Analista, estudando a complexidade de algoritmos de busca linear (ou sequencial), concluiu corretamente que no pior caso, considerando um vetor de n elementos, este tipo de algoritmo tem complexidade

a) O(n).
b) O(log_2n-1).
c) O(√n).
d) O(log_2n).
e) O(log_2n^2).

40. (FGV / Analista do Ministério Público (MPE AL) / 2018 / / Desenvolvimento de Sistemas)

A complexidade do algoritmo de busca binária, sobre uma lista indexada ordenada pela chave de busca, é

a) O (N)
b) O ($\log_2$ N)
c) O (2 $\log_2$ N)
d) O (N $\log_2$ N)
e) O (N^2)

7 GABARITOS

7.1 GABARITO: INTRODUÇÃO

1. D
2. ERRADO
3. CERTO
4. ERRADO
5. CERTO
6. E
7. E

7.2 GABARITO: VETORES E MATRIZES

1. E
2. D
3. E
4. CERTO
5. ERRADO
6. CERTO
7. CERTO
8. B

7.3 GABARITO: LISTAS

1. C
2. ERRADO
3. CERTO
4. CERTO
5. ERRADO
6. CERTO
7. CERTO
8. CERTO
9. ERRADO
10. ERRADO
11. CERTO
12. ERRADO
13. ERRADO
14. ERRADO
15. D
16. ERRADO
17. ERRADO
18. E
19. A
20. C

7.4 GABARITO: FILAS

1. B
2. B
3. ERRADO
4. ERRADO
5. ERRADO
6. ERRADO
7. ERRADO
8. CERTO
9. A
10. A
11. CERTO
12. CERTO
13. CERTO
14. ERRADO
15. C
16. CERTO
17. ERRADDO
18. D
19. B
20. C

7.5 GABARITO: PILHAS

1. CERTO
2. B
3. D
4. E
5. B
6. E
7. ERRADO
8. ERRADO
9. ERRADO
10. E
11. E
12. D
13. ERRADO
14. B
15. CERTO
16. CERTO
17. ERRADO
18. E
19. CERTO
20. CERTO
21. ERRADO
22. C
23. D

7.6 GABARITO: ÁRVORES

1. C
2. B
3. B
4. B
5. D
6. C
7. CERTO
8. ERRADO
9. CERTO
10. ERRADO
11. ERRADO
12. ERRADO
13. ERRADO
14. CERTO
15. D
16. ERRADO
17. ERRADO
18. ERRADO
19. B
20. CERTO
21. E
22. X
23. E
24. B
25. D
26. E
27. D
28. D
29. C
30. B
31. D
32. D
33. B
34. B
35. CERTO
36. E
37. D
38. A

7.7 GABARITO: COMPLEXIDADE DE ALGORITMOS

1. B
2. A
3. B

7.8 GABARITO: ORDENAÇÃO

1. C
2. E
3. D
4. D
5. E
6. B
7. ERRADO
8. A
9. C
10. D
11. E
12. A
13. A
14. B
15. CERTO

7.9 GABARITO: PESQUISA

1. B
2. B
3. B
4. CERTO
5. C
6. D
7. C
8. ERRADO
9. ERRADO
10. A
11. B

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.