Estratégia
CONCURSOS

# Aul

Desenvolvimento de Software para Concursos - Curso Regular 2017

Professor: Diego Carvalho, Equipe Informática e TI

# AULA 00



Lógica de Programação. Tipos de Dados. Operadores e Expressões. Estruturas de Controle, Seleção, Repetição e Desvio. Recursividade. Funções e Procedimentos. Complexidade de Algoritmos. Métodos de Ordenação. BubbleSort, InsertionSort, SelectionSort, QuickSort, ShellSort, MergeSort e HeapSort. Estruturas de Dados. Vetores e Matrizes. Lista Encadeada. Pilhas. Filas. Árvore. Grafos. Hashing. Estrutura de Arquivos. Padrões de Projeto GOF. Padrões de Projeto Java EE. Padrões de Projeto GRASP. Java SE. Conceitos Básicos. Plataforma Java. Compilação e Interpretação. Passagem por Valor e Referência. Empacotamento. Raiz. Identificadores. Blocos e Comandos. Comentários. Palavras Reservadas. Tipos Primitivos. Operadores. Vetores. Conversão de Tipos. Controle de Fluxos. Classes. Objetos. Atributos. Métodos. Herança. Encapsulamento. Interface. Polimorfismo. Interface Gráfica. Tipos Enumerados. Anotações. Classes Internas. Reflexão e Genéricos. Tratamento de Exceções. Sincronismo e Multithreading. Coleções. Streams e Serialização. Classes e Operações de I/O. Novidades Java 8. Arquitetura Java EE. JSP. Servlets. JSF. JPA. Hibernate. JDBC. JVM. Spring. Struts. JMS. JNDI. JTA. JSTL. HTML e CSS. JavaScript. JQuery. AJAX. DHTML. XHTML. XML. XSLT. XSD. Sistemas Móveis. Android. iOS. Segurança no Desenvolvimento. Compiladores e Interpretadores. Ferramentas de Controle de Versão. SVN. CVS. Git. Análise Estática de Código-Fonte. SonarQube. Integração Contínua (e outros) Framework .NET, C#, Visual Basic, ASP.NET e Visual Studio Delphi e PHP.

E aí, querem mais teoria? Mais exercícios? Tem muito mais! Essa é apenas a aula demonstrativa para que vocês conheçam o método e a escrita! Espero que vocês venham conosco... grande abraço ;)

## A APRESENTAÇÃO...

Olá, sejam bem-vindos! Galera, esse é o nosso curso regular de Desenvolvimento de Sistemas para Concursos! É o que você acha de mais completo sobre o assunto, cobrindo uma matéria bastante ampla com muita teoria e muitos exercícios resolvidos. Vamos ficar ligados para não perder o foco e porque concorrência não dorme! Estou aqui para ajudá-los. Vamos lá...

# DÚVIDAS DOS ALUNOS

# TOP 5

1. Peço encarecidamente que leiam as instruções dessa primeira aula. Eu sei que é chato, mas assim nós alinhamos nossas expectativas a respeito do curso.

2. Essa é a Aula Demonstrativa (está disponível para todos na internet) – o restante do conteúdo estará disponível na Aula 01 (apenas para aqueles que adquirirem o curso).

3. Esse curso não possui vídeo-aulas! Estamos trabalhando para disponibilizá-las em breve, nesse segundo semestre – talvez ainda não seja possível disponibilizá-las para esse curso.

4. Esse curso contempla somente aquilo que está em seu cronograma. Ele não contempla todo edital de tecnologia da informação, nem outras disciplinas, nem discursivas, estudos de caso, etc.

5. Existem questões de Múltipla Escolha (A, B, C, D, E) e existem questões de Certo/Errado (C, E). Quando não há itens para escolha na questão, é porque a questão é da Modalidade Certo/Errado.

## O PROFESSOR...

Uma breve apresentação: meu nome é Diego Carvalho, bacharel em Ciência da Computação pela Universidade de Brasília, pós-graduado em Gestão de Tecnologia da Informação na Administração Pública e Analista de Finanças e Controle da Secretaria do Tesouro Nacional. Já passei por esses perrengues de concurseiro e sei de duas coisas: a estrada é difícil, mas o prêmio compensa! E muito!

www.facebook.com/professordiegocarvalho

| ÁREA | ÓRGÃOS PARA OS QUAIS JÁ MINISTREI CURSOS | | | |
|---|---|---|---|---|
| Agência | ANCINE | ANTAQ | ANATEL | |
| Jurídica | TRT/1 | TRT/2 | TRT/3 | TRT/4 |
| | TJ/BA | CNMP | MP/PB | TRT/15 |
| Legislativa | CÂMARA DOS DEPUTADOS | | | |
| Auditoria | TCE/RS | TCE/SP | TCE/CE | TCM/GO |
| | TCU | TCM/SP | | |
| Fiscal | ISS/SP | ISS/BA | | |
| Outros | CEF | DATAPREV | DEPEN | INMETRO |

Nesse curso, e nos próximos de desenvolvimento, estará com vocês também o Prof. Leon Sólon. Ele é bacharel em Ciência da Computação pela Universidade de Brasília (2002), mestre em Computação Aplicada também pela UnB (2016) e possui MBA em Gerência de Projetos de Software pela UniEuro (2008). Tem bastante experiência em concursos de TI tendo sido aprovado nos concursos de Auditor-Fiscal da Receita Federal do Brasil (concurso de TI 2005), Analista de Informática Legislativa da Câmara dos Deputados (2007), Técnico em Informática do TST (2005), Técnico em Informática do MPU (2004) e Analista de Sistema do Serpro em (2004). Prof. Leon já ministrou aulas de TI em outros cursos preparatórios em Brasília e agora está no Estratégia para somar ao time de TI.

Galera, lá no site, nós – professores – temos algumas métricas para medir se o nosso desempenho nos cursos está bacana! Os alunos podem avaliar com notas e, inclusive, escrever anonimamente o que acharam do professor e do curso. Apresento abaixo o resultado de alguns cursos ministrados recentemente. Portanto, confiem em mim... vocês vão aprender muito com esse curso!

| Comentário sobre o curso |
|---|
| Curso bom. |
| Muito bom o curso, abrange os tópicos cobrados conforme o edital. |
| Atende ao conteúdo pertinente ao edital! |
| Pontos positivos:<br>-Marcações coloridas no texto para destacar as informações<br>-Boa cobertura dos pontos do edital<br>-Imagens ilustrativas<br>-Ilustrações feitas pelo próprio professor (eu acho) que ficaram bem legais<br>-Muitas questões resolvidas<br>-Qualidade do conteúdo foi muito boa<br>Pontos negativos:<br>-Muita informalidade, pode ser só uma frescura minha, mas o "Galera" é usado em excesso, acabou prejudicando um pouco a seriedade da aula.<br>-Muitas questões do CESPE, apesar de a banca ser FCC<br>-Comentários muito sucintos em algumas questões: "Perfeito, é isso mesmo."<br>Considerações adicionais:<br>Não sei se o professor tinha experiência prévia no ramo, mas pelo menos para mim ficou clara a melhora das aulas ao longo do curso. Ao final realmente estava TOP, digno do Estratégia Concursos. No mais, eu gostaria de agradecer ao professor Diego por ter se disposto a dar essas aulas. Foram só 500 inscritos no concurso, é bastante provável que poucas pessoas tenham feito o curso, então a rentabilidade deve ter sido baixa. Tenho certeza que muito cursinho por aí cancelaria, mas mesmo assim ele foi adiante. Valeu!<br>Essas aulas serão úteis não só para esse concurso. Vem aí o TCU TI e praticamente tudo do seu curso deve cair, então foi um ótimo investimento. Com certeza estudarei todo o material várias vezes! |

| Comentário sobre o curso |
|---|
| Fui colega do Diego no CF/STN mas durante o período do CF não conversei muito com ele. Soube que era professor de TI também além de concurseiro. Não conhecia o trabalho dele como professor mas me surpreendi positivamente, muito mesmo. A aula é elaborada com muito cuidado, criatividade e é uma aula leve de se estudar. Apesar de eu estar afastado da área de Engenharia de Software e Desenvolvimento já há algum tempo foi muito tranquilo acompanhar a exposição dele nas aulas. Parabéns Diego! Muito bom trabalho!! |
| ótimo curso |
| No começo estava meio cético com relação ao curso, mas quanto mais ia me aprofundando nas apostilas percebi que o professor fez um bom trabalho.<br>No mais, gostaria de ter mais questões comentadas, e com comentários um pouco mais completos. |
| Otima didática com conteúdo e exercicios de diversas bancas logo em seguida. Quando tiver as videos aulas será melhor ainda.<br>Continue assim ... |
| O curso me atendeu muito bem. Aspectos que gostei bastante:<br>- Linguagem prática e objetiva<br>- Sem enrolações<br>- Sem copy/paste desnecessários no comentários de questões repetitivas (que cobram o mesmo conhecimento), explicou uma vez, ok, as outras basta pincelar de forma rápida.<br>- Conteúdo bem direcionado. Não foi superficial mas também não desceu ao nível NASA.<br>- Cumpriu o cronograma<br>Como fatores negativos, apenas com relação aos pequenos erros encontrados vez ou outra nos comentários das questões ou gabaritos trocados. Entretanto, conforme já explicado pelo próprio professor, o curso está amadurecendo.<br>Estou satisfeito. Tenho pretenção de adquirir novamente esse curso daqui a algum tempo e conferir a evolução, mas, cá entre nós: Quero mesmo é passar de uma vez agora e não precisar disso nunca mais :D<br><br>Sucesso Diego.<br>Att, Benjamin Pinto |
| Gostei muito, doo curso. Apenas senti falta dos vídeos para diversificar um pouco, não ficar tão teórico. |

| Comentário sobre o curso |
| --- |
| Excelente professor na área de TI. Um dos melhores que já vi em todos os cursos que já realizei, seja presencial ou a distância. Realizaria qualquer outro curso ministrado pelo professor. |
| Professor muito bom! Aulas totalmente direcionadas no edital, clareza para escrever, boa seleção de exercícios e bons comentários. Passa segurança para o aluno e tem domínio do conteúdo. |

| Comentário sobre o curso |
| --- |
| Excelente curso!<br>Excelente material, excelente professor, excelente equipe.<br>Recomendo a todos! |
| "Excelente" é pouco para esse curso. Eu adquiri outras aulas para me preparar para esse concurso, mas nenhuma outra agregou tanto. Imagino o trabalho que deve ter dado para "compilar" todos esses assuntos. Além disso, colocá-los de forma tão didática em aulas relativamente pequenas. Isso é altíssima qualidade!<br><br>O conteúdo foi muito bem elaborado! E eu reparei isso porque depois que estudei por ele consegui resolver muitas questões que procurei sobre esses assuntos. Questões que antes pareciam ser de "outro mundo" pra mim.<br><br>A organização da aula também merece destaque... As figuras, os esquemas, as questões, as notas de rodapé e a evolução do conteúdo estudado de acordo com os itens no edital... Sem falar na didática!<br><br>Quanto ao professor, sem palavras. Respondeu todas minhas dúvidas e sempre se mostrou bastante receptivo. Bom, não sei como irei me sair na prova domingo, mas devo MUITO da minha preparação ao professor Diego Carvalho. Ele nem imagina o quanto me ajudou.<br>Com certeza recomendo esse curso! |
| Eu me considero uma pessoa leiga em TI, pois sou formado em Engenharia Mecânica e pos graduado em economia, além de nunca ter trabalhado em departamentos de TI nas empresas onde trabalhei.<br>Por isso, apesar de ter tido dificuldades, achei muito adequada a linguagem adotada pelo professor. Os assuntos são muito complexos para serem ensinados a pessoas com pouca base de conhecimento na área, e ainda assim eu entendo que o curso tenha atingido seu objetivo de permitir a transmissão do conhecimento proposto aos alunos.<br>Como pontos de melhoria, sugeriria:<br>- utilização de videoaulas, pois diversos assuntos poderiam ter seu entendimento facilitado com a utilização de imagens<br>- redistribuição na sequência de assuntos. Ainda que eu entenda que vários assuntos sejam interligados, o que dificulta a escolha sobre "o que ensinar primeiro", acho que determinados assuntos poderiam ter sua ordem invertida, ainda que isto não tenha impedido o entendimento da matéria.<br>Muito obrigado.<br>André. |
| NULL |
| Excelente curso, muito didático, material de alta qualidade. |
| Ok |

| Comentário sobre o curso |
|---|
| òtima didática do Professor Diego!! |
| "Sempre podemos melhorar"<br>Está muito bom o curso, o pessoal da Estrategia está de parabéns.<br>abs.<br>Airton. |
| O curso ajuda demais a estruturar e ordenar o conteudo dos cursos, coisa quase impossível de fazer sozinho. Será o primeiro concurso que irei fazer para avaliar realmente a eficácia do curso.<br>Sempre lemabrando que é muito importante e a exempliação da teoria com exemplos práticas reais e ou fictícios que ajudam a fixar o conteúdo, principalmente para aqueles cursos que tem provas discursivas ou questões abertas.<br>Marco Costa |
| Aulas muito bem explicadas. Respostas muito rápidas. Conteúdo abrangente, focado e muito bem baseado bibliograficamente. Parabéns. Excelente curso. |
| Esta foi a matéria que mais gostei dentre as abordadas para o concurso do TRT. Gostei muito da linguagem que o professor utilizou e os exercícios no meio do conteúdo deixam a aula mais dinâmica! |
| Excelente material. Focado para o concurso e muita questão para fixação. |
| Melhor didática dos pacotes de TI pro TRT. |
| É uma pena que a última aula chegou tão perto da prova, mas eu entendo a complicação e é fato que o professor fez um trabalho muito bom.<br>Vou revizar o último PDF amanhã, mas até o momento, de longe, foi o melhor material que eu encontrei sobre os outros assuntos.<br>Uma sugestão de melhoria é o próprio site. O sistema de perguntas e respostas é horrível, feio e de baixíssima usabilidade. |
| Excelente didática! As apostilas dão vontade de ler até quando o assunto não é dos mais legais. Parabéns pelo curso! |
| O curso é excelente. Sugiro ao Professor lançar um curso regular de Java tratando todos assuntos (programação, JEE, etc.). |
| Muito bom o material, apostilas com muito exercícios. |

| Comentário sobre o curso |
|---|
| O curso está excelente para a preparação para o concurso do TCE-SP e outros que tem abordado os mesmos temas. Bem focado nos temas e apresenta os aspectos gerais de cada um deles. O professor comentou muitos exemplos e exercícios. Nem senti falta de vídeos. |
| Olá professor!<br>de forma geral gostei do curso.<br>Eis alguns pontos para melhorar:<br>- o fato de limitar o tamanho dos parágrafos em poucas linhas facilita, mas acho que não deveriam ser quebrados no meio do mesmo assunto...algumas vezes me perdi...<br>- a observação a respeito do modelo espiral, no qual alguns autores consideram incremental e outros não, poderia ser reforçado no tópico do modelo espiral, mais para o fim da aula (já não lembrava mais desta informação quando cheguei neste modelo...)<br>Desejo muito sucesso! |
| Achei o material muito bom, objetivo e com muitos exercícios. São resumões sobre os asssuntos, então só vou saber se a profundida é adequada para concursos depois que fizer a prova, mas no geral me pareceu que cobriu bem os temas.<br>Att,<br>Nádia. |
| Muito bom o curso. |
| okjok |
| DADA A QUANTIDADE DE TEMAS ABORDADOS E EXPLICADOS, ACHEI UM CURSO BEM COMPACTO E PRÁTICO. |
| Apresentou um número de questões comentadas muito bom e com bons esclarecimentos. |
| Muito bom mesmo.<br>Muitas questões do CESPE mesmo que o concurso fosse da Vunesp. Será que a FCC, FGV e Cesgranrio não tem questões desses assuntos?<br>As questões CESPE são \"diferentes\"... Não tive problemas com isso porque foco o TCU mas fica a dica.<br>Discursivas sobre BI e big data. Sugestões, por favor. |
| O curso foi excelente e atendeu perfeitamente o meu plano de estudo. Pois precisava de mobilidade e consegui estudar tanto no smartfone , tablet e notebook principalmente no status Off Line porque nem sempre tinha acesso a internet disponível. A didática do professor foi um ponto forte neste curso : A explicação da teoria com uma linguagem clara, cheias de questões comentadas e bom humor como se eu estivesse em uma aula presencial !! Os assuntos-chaves destacados em vermelho otimizou muito as minhas revisões! |
| Curso com um conteudo excelente, bem produtivo e rico nos detalhes passados que me ajudaram muito. |
| Gostei muito da linguagem com certeza compraria outros cursos |
| Não tenho referencia de outros cursos porque este é o primeiro que estou estudando pela internet mas pra mim foi muito bom. A linguagem não é cansativa e acredito que o que se estuda é somente o necessario. Uma coisa que não entendi é que as vezes havia algumas questões que havia a pergunta e a letra da resposta mas não havia as respostas para escolher. Acho que poderia melhorar isso ou então não entendi o conceito. |
| O professor tinha uma abordagem informal do assunto que facilitava a memorização. |
| Melhor professor de TI que já vi na área de concursos! |

| Comentário sobre o curso |
| --- |
| Existem algumas definições que só foram explicadas nos comentários dos exercícios, poderiam ser dadas antes.<br>No geral o material é muito bom. Só faltou vídeos pra ilustrar melhor alguns conceitos de arquitetura. |
| O material do curso é ótimo, mas o que faltou foi um resumo no final de cada aula com tudo que foi falado. De resto, eu gostei, foi meu primeiro curso no Estratégia e pretendo voltar a fazer outros. |
| Excelente didática do Professor. Certamente se ele lançar outro curos vou adquirir. Como sugestão o Estratégia devia fazer mais parcerias com professores de TI. Essa área é carente em cursos bons iguais a este. A procura é grande. |
| Muito bom o curso e principalmente o comprometimento do professor para com os alunos. |

| Comentário sobre o curso |
| --- |
| Muito bom o curso, contudo, senti falta das vídeo aulas complementares. |
| Gostei muito da forma com o professor respondeu as perguntas, da maioria do conetúdo apresentado. A única coisa que acredito tornar o curso perfeito são os vídeos, senti falta. O vídeo auxilia em um primeiro entendimento, para depois seguir lendo a apostila. |
| Estou iniciando no estudo de concursos para TI e gostei muito do curso. Não achei muito caro, isso me permitiu comprar outro curso no site.<br>Estão de parabéns! |
| Ótimo curso, tudo bem explicado |
| Curso de excelente qualidade.<br>Entretanto extremamente insuficiente para o concruso, mesmo com a Parte I. |

## LÓGICA DE PROGRAMAÇÃO

Vamos falar sobre Lógica de Programação! *Em primeiro lugar, por que chamamos de lógica?* Porque é necessário utilizar a lógica para resolver um problema computacional. *Como assim?* Precisamos de um encadeamento ou uma sequência de pensamentos para alcançar um determinado objetivo. Nós podemos descrever esses pensamentos como uma sequência de instruções ou passos.

*Professor, o que seria uma instrução?* É um conjunto de regras ou normas definidas para a realização ou emprego de algo, indicando ao computador uma ação elementar a ser executada. Bem, esses conceitos são muito básicos e vocês já devem estar bastante acostumados, por isso vamos ver rapidamente. Um conjunto de instruções formam um algoritmo:

> *Algoritmo: conjunto predeterminado e bem definido de passos destinados à solução de um problema, com um número finito de etapas.*

*Professor, você pode dar um exemplo?* Sim, o exemplo mais comum da bibliografia é mostrado acima: uma receita de bolo. Observem que para fazer um bolo (solucionar um problema), é necessário seguir uma sequência de passos finitos e predeterminados. No fim das contas, grosso modo, um software nada mais é do que a representação de um algoritmo.

*Professor, todos os programas que eu utilizo no meu computador são representações de algoritmos?* Sim! *Inclusive o joguinho de Paciência que eu curto?* Sim! *Mas até mesmo os apps que eu utilizo no celular?* Eles também! Todos os softwares (de desktop, notebook, smartphone, tablet, geladeira, relógio, entre outros) são representações de algoritmos.

*Então, basta que eu escreva um conjunto de passos em qualquer língua que o meu computador realiza a tarefa que eu quiser?* Claro que não! Computadores não entendem, por exemplo, português – eles entendem 0 e 1 (na verdade, eles entendem presença ou ausência de tensão elétrica), portanto é necessário representar esses algoritmos por meio de uma linguagem de programação.

```
1  // class declaration
2  public class ProgrammingExample {
3
4      // method declaration
5      public void sayHello() {
6
7          // method output
8          System.out.println("Hello World!");
9      }
10 }
```

*Como assim, professor?* Pessoal, um computador é uma grande calculadora. No entanto, ele é "burro", ele só calcula o que mandam-no calcular. As linguagens de programação surgem como uma solução para abstrair a comunicação entre seres humanos e computadores. Na imagem ao lado, a ordem do programador era: "*Computador, escreva na tela: Hello World!*".

Bem, acho que todo mundo já ouviu falar alguma vez na vida em Código-Fonte. Todo software possui um código-fonte, que é um conjunto de palavras organizado de acordo com regras específicas, formando um ou mais algoritmos. Essas palavras que formam o algoritmo são escritas utilizando uma linguagem de programação. Esse código-fonte é traduzido e posteriormente executado pelo usuário.

Pessoal... se eu não souber uma linguagem de programação, eu posso escrever um algoritmo utilizando um pseudocódigo! *O que é isso, professor?* É uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. Trata-se de um pseudo-código, logo não pode ser executado em um sistema real.

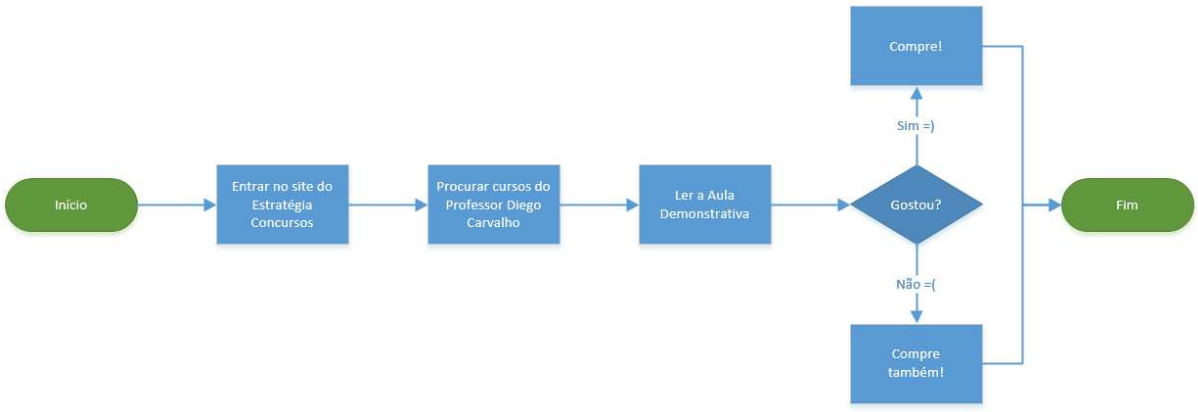
Um tipo de pseudocódigo é o Portugol (ou Português Estruturado)! Trata-se de uma simplificação extrema da língua portuguesa, limitada a pouquíssimas palavras e estruturas que têm significado pré-definido, na medida em que deve seguir um padrão. Emprega uma linguagem intermediária entre a linguagem natural e a linguagem de programação para descrever os algoritmos.

Embora o Portugol seja uma linguagem bastante simplificada, possui todos os elementos básicos e uma estrutura semelhante à de uma linguagem de programação de computadores. Portanto resolver problemas com português estruturado pode ser uma tarefa tão complexa quanto a de escrever um programa em uma linguagem de programação qualquer.

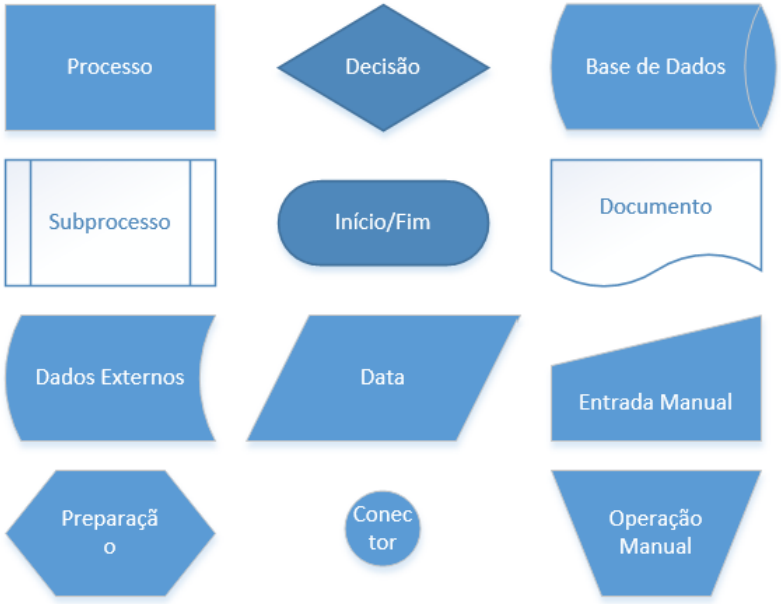
```
início
    <instruções>

    se <teste> então
        <instruções>
    senão
        <instruções>
    fim_se
fim
```

- Além do português estruturado, é possível representar um algoritmo também por meio de um Fluxograma! *O que é isso, professor?* É uma espécie de diagrama utilizado para documentar processos, ajudando o leitor a visualizá-los, compreendê-los mais facilmente e encontrar falhas ou problemas de eficiência, como mostra a imagem abaixo.



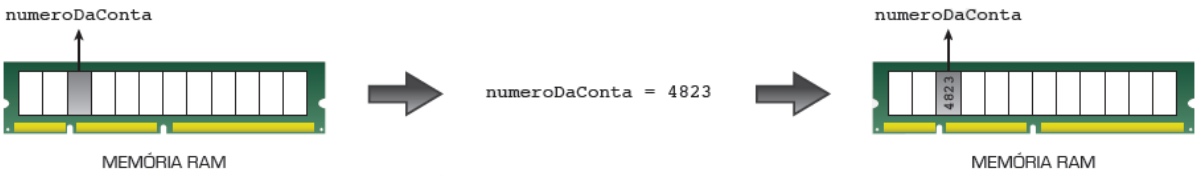

Os principais símbolos de um Fluxograma são:




Bem, nós vimos então que podemos representar algoritmos por meio de Linguagens de Programação, Pseudocódigo ou Fluxogramas! Em geral, os fluxogramas são mais utilizados para leigos; pseudocódigo para usuários um pouco mais avançados; e linguagens de programação para os avançados. Agora vamos falar de conceitos mais específicos: constantes, variáveis e atribuições!

Constantes são dados que simplesmente não variam com o tempo, i.e., possuem sempre um valor fixo invariável. Por exemplo: a constante matemática π é (sempre foi e sempre será) igual a 3.141592 – esse valor não mudará! *Professor, e o que seria uma variável?* São espaços na memória do computador reservados para guardar informações ou dados!

Em geral, variáveis são associadas a posições na Memória RAM, armazenando diversos tipos de dados que veremos com detalhes à frente! Ela possui um nome identificador que abstrai o nome do endereço na memória que ela ocupa. Observem a imagem abaixo: existe uma variável (espaço em memória) chamada **numeroDaConta** em que se armazena o valor 4823.



O conteúdo de uma variável pode ser alterado, consultado ou apagado diversas vezes durante a execução de um algoritmo, porém o valor apagado é perdido. *Bacana, mas e a atribuição?* Bem, trata-se de uma notação para associar um valor a uma variável, i.e., armazenar o conteúdo no endereço de memória específico. Cada linguagem de programação adotará uma maneira de representá-la.

Aqui, não iremos nos prender a uma linguagem de programação, vamos adotar o Português Estruturado! A notação de atribuição é representada por uma seta apontando para a esquerda do valor para o identificador, podendo ser:

```
Variável ← Constante:       dataDeNascimento ← 1988
Variável ← Variável:        endereço ← cidade
Variável ← Expressão:       idade ← (anoAtual – anoDeNascimento)
```

Vamos falar agora sobre tipos de dados! Em geral, eles se dividem em dois grupos: Dados Elementares e Dados Estruturados. Só uma informação antes de começar: os dados elementares também podem ser chamados de simples, básicos, nativos ou primitivos. Já os dados estruturados também podem ser chamados de compostos. *Bacana?* Vamos para as definições!

- Tipos Elementares: são aqueles que não podem ser decompostos. *Ora, se eu disser que o Pedrinho tem 10 anos, é possível decompor esse valor de idade?* Não, logo é um tipo elementar. Há diversos tipos elementares, dependendo da linguagem de programação utilizada. No entanto, os principais são:

    - Inteiro: também conhecido como Integer, são similares aos números inteiros da matemática, i.e., sem parte fracionária. Podem ser positivos, negativos ou nulos. Ex: -2% de crescimento do PIB; 174 km de distância; 0 ºC de Temperatura; etc.

    - Real: também conhecido como Float (Ponto Flutuante), são similares aos números reais da matemática, i.e., possuem parte fracionária. Ex: 3,141592 é a constante de PI; 9,81 $m/s^2$ de Aceleração Gravitacional; raiz quadrada de 7; etc.

    - Caractere: também conhecido como Literal ou Char, são representações de letras, dígitos e símbolos. Quando colocadas em conjunto, formam um tipo estruturado chamado String ou Cadeia de Caracteres. Ex: 'a', '$', '5', 'D', etc.

- Lógico: também conhecido como Boolean, são representações de valores lógicos – verdadeiro/falso, ligado/desligado, sim/não, etc. São extremamente importantes na programação, principalmente na verificação de condições.

- Tipos Estruturados: são aqueles que podem ser decompostos. *Ora, se eu disser que o nome da bola da copa é Brazuca, é possível decompor esse nome?* Sim, basta dividi-lo em caracteres: 'B', 'r', 'a', 'z', 'u', 'c', 'a'. Há infinitos tipos estruturados[1], pois eles são a combinação de vários outros, o mais comum é:

  - Cadeia de Caracteres: também conhecido como String, são representações de sequências de caracteres, incluindo ou não símbolos. Pode ser uma palavra, frase, código, etc, por exemplo: "O rato roeu a roupa do rei de Roma".

Pessoal, quase todas as linguagens de programação possuem instruções para Leitura e Escrita de dados. A Escrita é uma Saída de Dados (Output) que busca mostrar informações ao usuário na tela do computador (Ex: Escrever (idade)). A Leitura é uma Entrada de Dados (Input) que busca ler dados do usuário por meio teclado (Ex: Ler (idade)). Para manipular dados, utilizamos operadores:

Operadores Aritméticos: são utilizados para obter resultados numéricos, preocupando-se com a priorização[2].

| Operador | Símbolo | Prioridade |
|---|---|---|
| Multiplicação | * | 2º |
| Divisão | / | 2º |
| Adição | + | 3º |
| Subtração | - | 3º |
| Exponenciação | ^ | 1º |

Operadores Relacionais: são utilizados para comparar números e literais, retornando valores lógicos.

| Operador | Símbolo |
|---|---|
| Igual a | = |

[1] Uma música em .mp3, um texto em .pdf, uma imagem em jpg são todos tipos estruturados.
[2] Em operadores que possuem a mesma prioridade, o que aparecer primeiro deve ser priorizado! Além disso, parênteses possuem sempre a maior prioridade!

| | |
|---|---|
| Diferente de | <> ou != |
| Maior que | > |
| Menor que | < |
| Maior ou igual a | >= |
| Menor ou igual a | <= |

Operadores Lógicos: servem para combinar resultados de expressões, retornando valores lógicos (verdadeiro ou falso).

| Operador/Símbolo |
|---|
| E/And |
| Ou/Or |
| Não/Not |

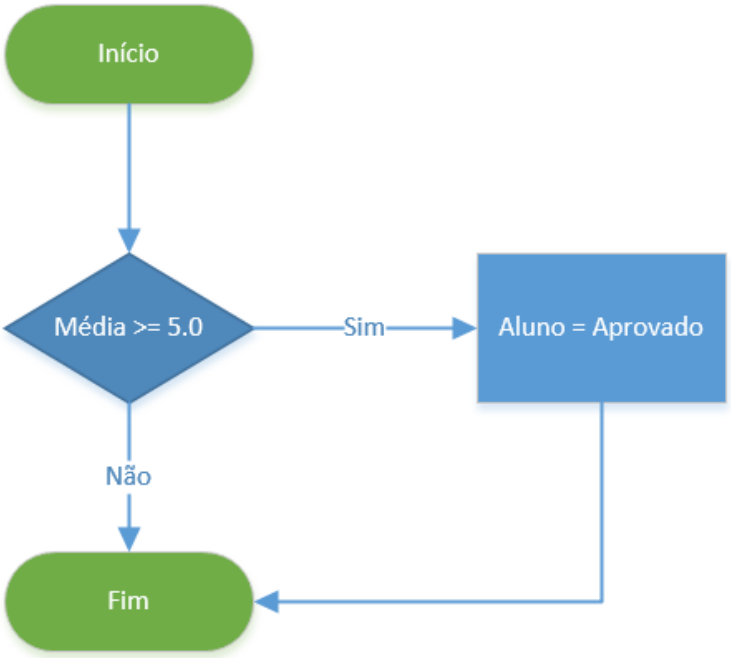
Bacana! Com isso, já podemos falar sobre Estruturas de Decisão e Repetição! Galera, as Estruturas de Decisão (também chamadas de Estruturas de Seleção, inclusive no edital) permitem interferir na sequência de instruções executadas dependendo de uma condição de teste, i.e., o algoritmo terá caminhos diferentes para decisões diferentes. O contrário dessa afirmação é a execução sequencial das instruções, sem loops ou desvios.

*Quando terminarmos, todos devem saber cantar "Hey Jude" na ordem certinha seguindo os comandos no fim da aula! Quero todos acertando!*

A melhor maneira de se visualizar uma estrutura de decisão é com fluxogramas:

## Caso 1:

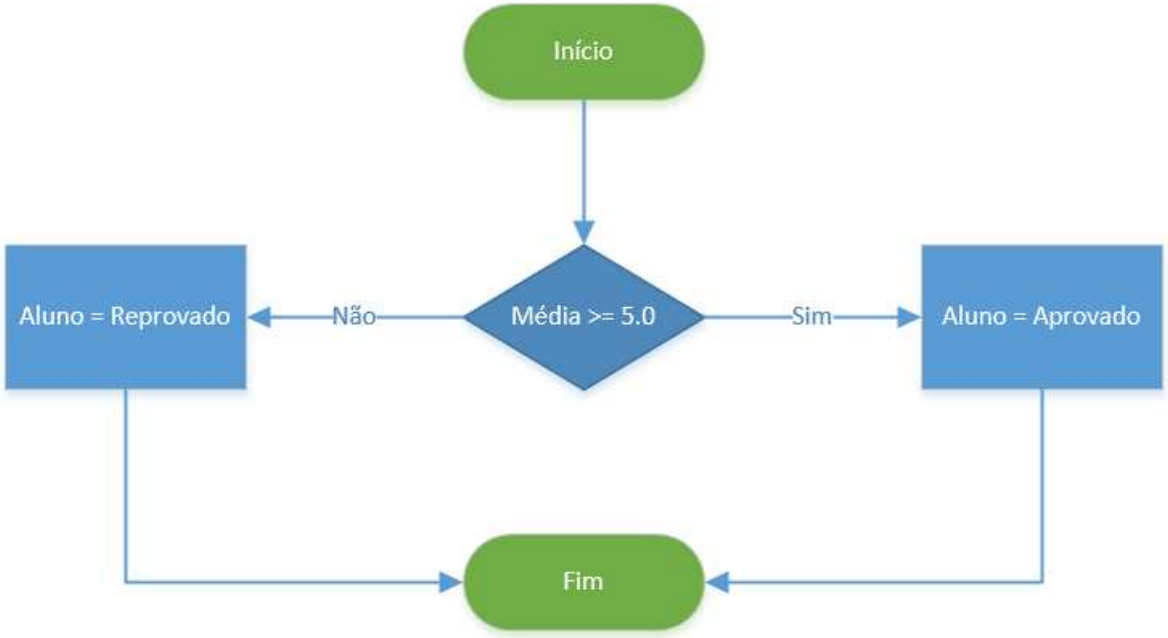
```
Se (Média >= 5.0) Então
      Aluno = Aprovado
```




O primeiro caso, e mais simples, é uma estrutura de decisão com um teste (Média >= 5.0). O programa avalia se a variável Média tem o valor maior ou igual a 5.0, no caso de o teste ser verdadeiro a instrução "Aluno = Aprovado" é executada. Em caso negativo, o programa pula a instrução "Aluno = Aprovado" e continua logo após o final do bloco de decisão. Quando uma estrutura de decisão (seleção) possui somente o bloco "Se Então", é chamada de simples. *Isso já foi cobrado em provas da FCC, atenção!* Algumas provas também cobram os nomes das estruturas em ingês, ou seja, ao invés de "Se Então", também podem cobrar como "If Then".

## Caso 2:

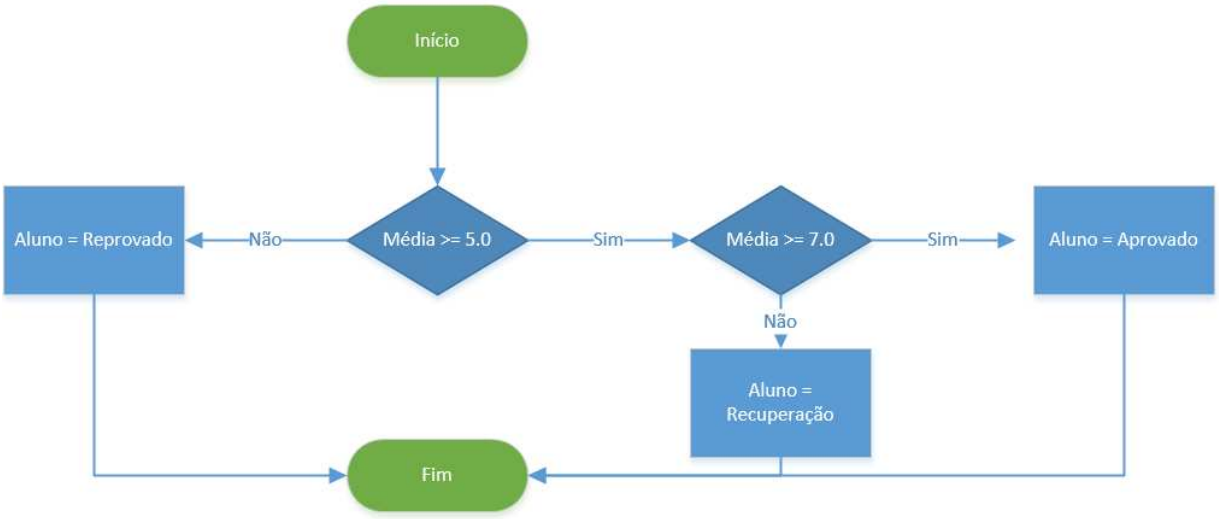
```
Se (Média >= 5.0) Então
        Aluno = Aprovado
Senão
        Aluno = Reprovado
```




O segundo caso é só um pouquinho mais complexo, onde temos dois fluxos possíveis, um que passa pelo comando "Aluno = Aprovado" e outro que passa pelo "Aluno = Reprovado". Importante ressaltar que, nesse caso, sempre alguma dessas instruções serão executadas, pois, ou Média é maior ou igual a 5.0 ou é menor, sempre. Esse exemplo de estrutra de decisão possui os blocos Se Então Senão. Quando a estrutura de decisão possui todos os blocos (Se Então Senão) ela é chamada de composta. Em inglês a estrutura composta é conhecida como "If Then Else"

## Caso 3:

```
Se (Média >= 5.0) Então
        Se (Média >= 7.0) Então
                Aluno = Aprovado
        Senão
                Aluno = Recuperação
Senão
        Aluno = Reprovado
```

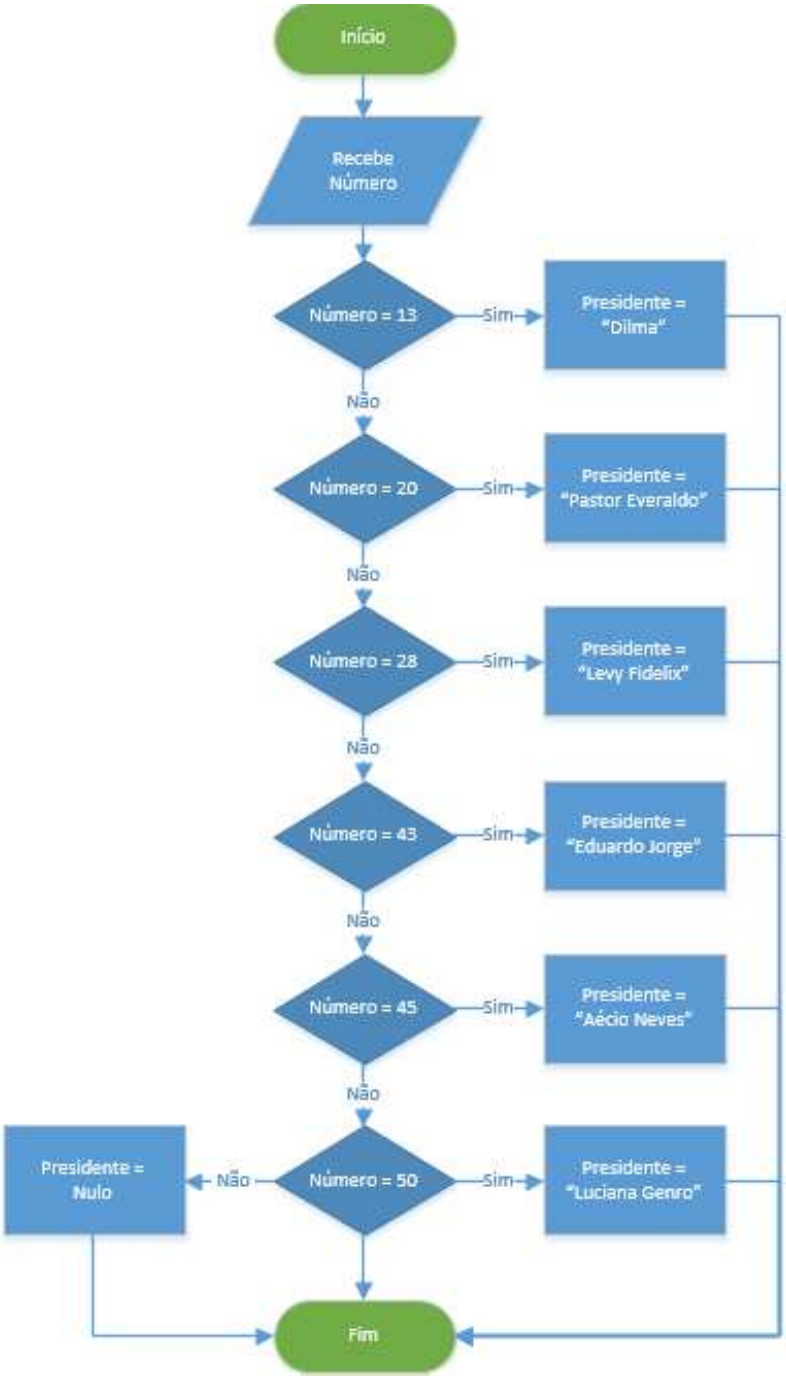



Nada nos impede de utilizar estruturas de decisão dentro de outras estruturas de decisão. Nesse caso, o programa realiza um primeiro teste (Média >= 5.0). No caso de a Média ser menor que 5.0, o comando "Aluno = Reprovado" será executado e o fluxo termina. Caso contrário, ou seja, a Média sendo maior ou igual a 5.0 executamos os comandos do bloco "Então", que, nesse caso possui mais uma estrutura de decisão. O segundo teste avalia a expressão "Média >= 7.0". Chamamos esse tipo de utilização em diferentes níveis de estrutura de decisão (seleção) aninhada.

Antes de aprendermos a nossa última estrutura de decisão (seleção), "Caso Selecione", temos que nos atentar para o fato de que a condição testada nas estruturas de decisão não precisam ter uma expressão somente. Qualquer uma que retorne um valor verdadeiro ou falso é válida como condição de teste. Expressões como (Média > 10 v Média < 5) ou ainda (Aluno == Reprovado ^ Média < 3.0 ^ NumeroFaltas > 5) poderiam ser utilizadas.

## Caso 4:

```
Selecione (Número)
       Caso 13: Presidente = “Dilma”
       Caso 20: Presidente = “Pastor Everaldo”
       Caso 28: Presidente = “Levy Fidelix”
       Caso 43: Presidente = “Eduardo Jorge”
       Caso 45: Presidente = “Aécio Neves”
       Caso 50: Presidente = “Luciana Genro”
       Caso Outro: Presidente = “Nulo”
Fim Selecione
```



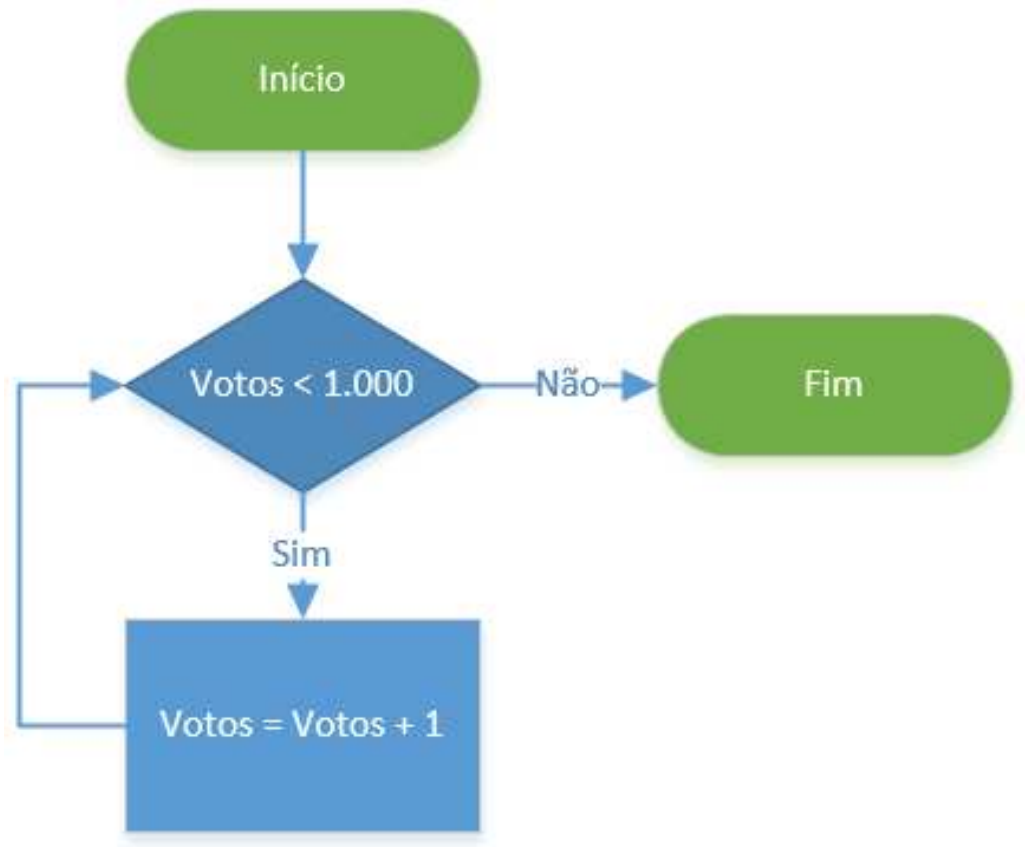

A estrutura "Caso-Selecione" também é conhecida como de decisão múltipla escolha. Diferente das outras que vimos, não parte de um teste de condição para definir para onde o fluxo deve ser desviado, mas sim avalia o valor de uma variável e dependendo de qual seja o conteúdo, encaminha para o rótulo indicado e executa as instruções ali apresentadas.

Portanto vimos quatro casos de Estrutura de Decisão (Seleção): Se-Então, Se-Então-Senão, Se-Então (Aninhados) e Caso-Selecione. Em inglês: If-Then, If-Then-Else, If-Then (nested) e Switch-Case.

Agora veremos Estruturas de Repetição! Elas são utilizadas quando se deseja que um determinado conjunto de instruções ou comandos sejam executados por mais de uma vez. A quantidade pode definida, indefinida, ou enquanto um estado se mantenha ou seja alcançado. Vejamos:

**Caso 1:** Repetição Pré-Testada (Testa-se antes de processar!)

```
Enquanto (Votos < 1.000) Faça
        Votos = Votos + 1
Fim-Enquanto
```




Esse primeiro tipo de estrutura de repetição realiza um teste antes de executar qualquer instrução dentro de seu bloco. Desse modo, as instruções podem nem ser executadas, caso o teste da condição inicial falhe. No exemplo acima, se a variável "Votos" for maior ou igual a 1.000, a instrução "Votos = Votos + 1" não é executada uma vez sequer.

A variável que está sendo testada deve sofrer alguma alteração dentro do bloco da estrutura de repetição sob pena de a execução não ter fim. Se mudássemos o código para, por exemplo:

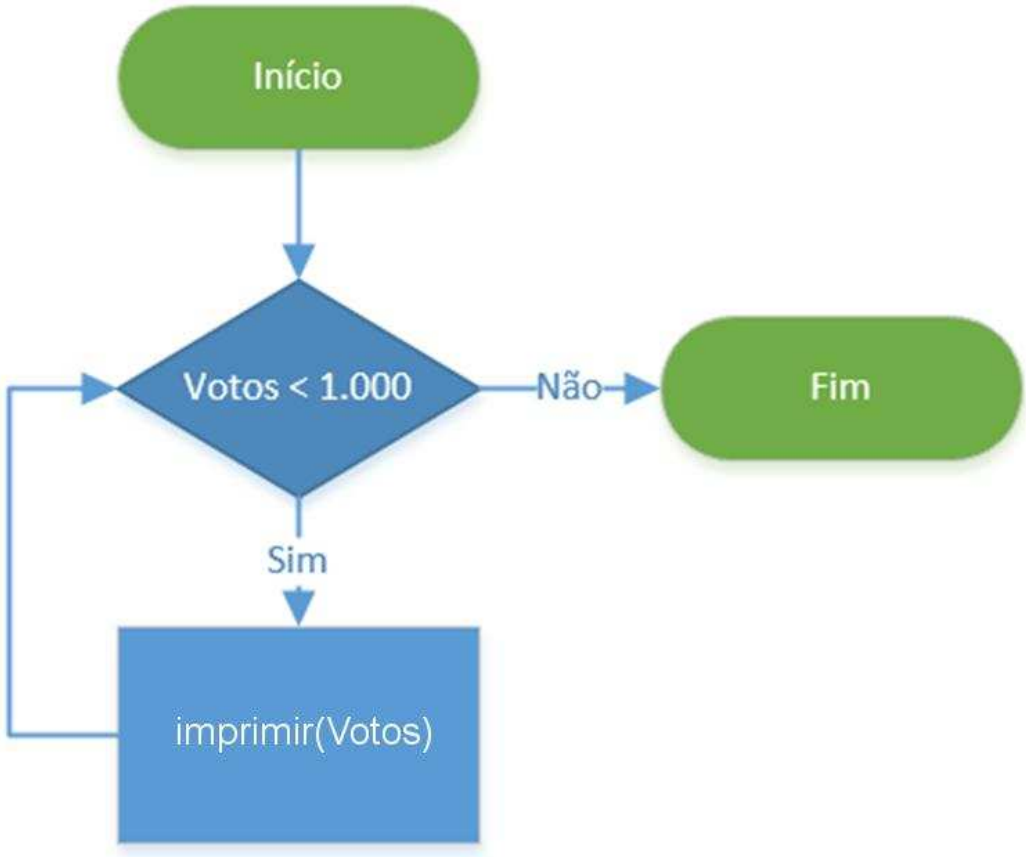
```
Votos = 500

Enquanto (Votos < 1.000) Faça
       imprimir("Eu nunca mais vou sair daqui! Muahahahaha")
Fim-Enquanto
```

A variável votos tem o valor estipulado em 500, ou seja, é menor que 1.000. Quando é executado o primeiro teste "Votos < 1000" o resultado é verdadeiro, ou seja, o fluxo é direcionado para dentro do bloco e a instrução "imprimir" é executada. Na segunda repetição, como não houve alteração da variável "Voto" vai executar novamente a instrução "imprimir" e assim por diante. Daí a obrigatoriedade de se alterar o valor da variável utilizada no teste de condição de entrada e saída da estrutura. Em inglês, a estrutura é conhecida como "While Do".

## **Caso 2:** Repetição com Variável de Controle

```
Para Votos de 1 até 1000 Faça
       imprimir(Votos)
Fim-Enquanto
```



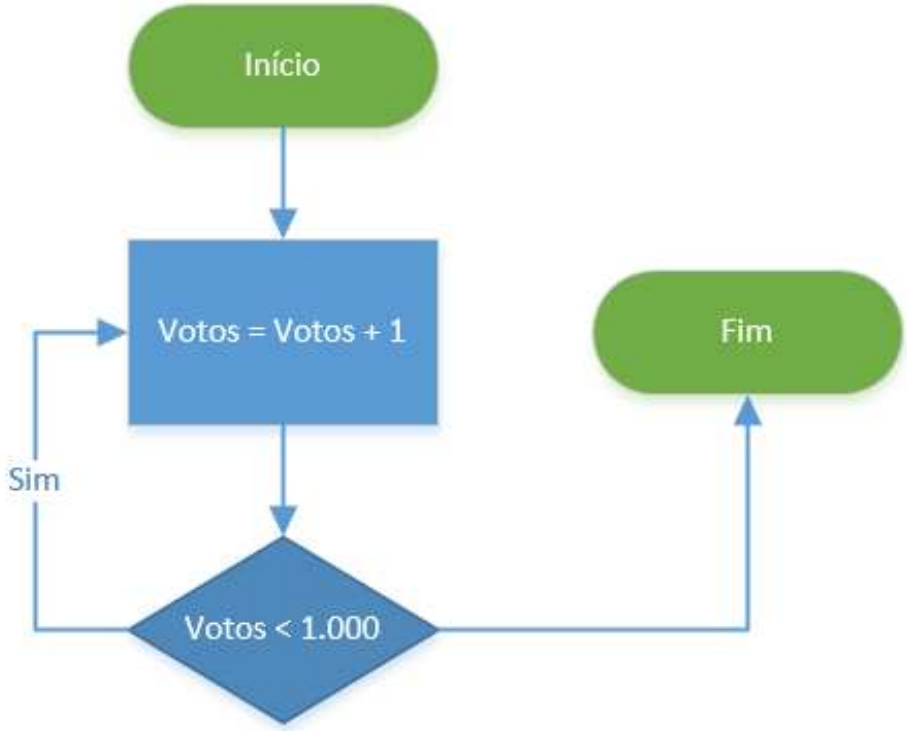
Esse tipo de estrutura é um pouco diferente da primeira, já que define de antemão quantas vezes será repetida as instruções dentro do bloco. No nosso exemplo, a instrução "imprimir(Votos) " será executada 1000 vezes, pois assim determina a expressão "Para Votos de 1 até 1000 Faça". Essa expressão é simplificada, mas realiza várias intruções internas:

- Testa se Votos é igual a 1000
- Se é igual, encerra a repetição, se é menor, soma 1 à variável de controle "Votos" e parte para mais uma repetição

Perceba que, como não há outros testes de condição de saída a não ser a contagem da variável de controle, as instruções que fazem parte do bloco de execução da estrutura de repetição não precisam alterar a variável de controle, pois a prória estrutura o faz. É uma grande diferença para os casos de estrutura de repetição pré-testada e pós-testada, que necessitam de tal alteração, como vimos e veremos adiante.

**Caso 3:** Repetição Pós-Testada (Testa-se depois de processar!)

```
Faça
       Votos = Votos + 1
Enquanto (Votos < 1.000)
```



O último caso é o pós-testado. Como sugere o nome, nessa estrutura a instrução do bloco é executada sempre ao menos uma vez! Isso porque o primeiro teste somente é feito ao final. Em inglês essa estrutura é conhecida como "Do While".

*Agora que sabemos tudo sobre estruturas de seleção (ou decisão) e de repetição, que tal tentar um exemplo de código que imprime a letra de "Hey Jude" dos Beatles. Fomos inspirados pelo fluxograma abaixo, que destaca muito bem as repetições e decisões para saber que parte da letra cantar. Vamos nessa??*

```
01 estrofe = 0
02
03 Enquanto (estrofe < 3) Repetir
04   estrofe == estrofe + 1
05   imprimir("Hey Jude, don't ")
06
07   Se (estrofe == 1) Então
08     imprimir("make it bad")
09     imprimir("take a sad song and make it better")
10   Senão Se (estrofe == 2) Então
11     imprimir("be afraid")
12     imprimir("you were made to go on and get her ")
13   Senão Se (estrofe == 3) Então
14     imprimir("let me down")
15     imprimir("you have found her, now go and get her")
16
17   imprimir("remember to let her ")
18
19   // Teste se estrofe é par ou ímpar
20   Se (estrofe % 2 == 1) Então
21     imprimir("into your heart")
22   Senão
23     imprimir("under your skin")
24
25   imprimir("then you ")
26
27   // Teste se estrofe é par ou ímpar
28   Se (estrofe % 2 == 1) Então
29     imprimir("can start")
30   Senão
31     imprimir("begin")
32
33   imprimir("to make it better")
34
35   Se (estrofe == 3) Então
36     imprimir("better better better better BETTER WAAAAAAAAAAAA")
37     Repetir
38        imprimir("NA NA NA NANANA NAAAAAAA")
39        imprimir("NANANA NAAAAAAA")
40        imprimir("Hey Jude")
41     Enquanto(Verdadeiro)
```

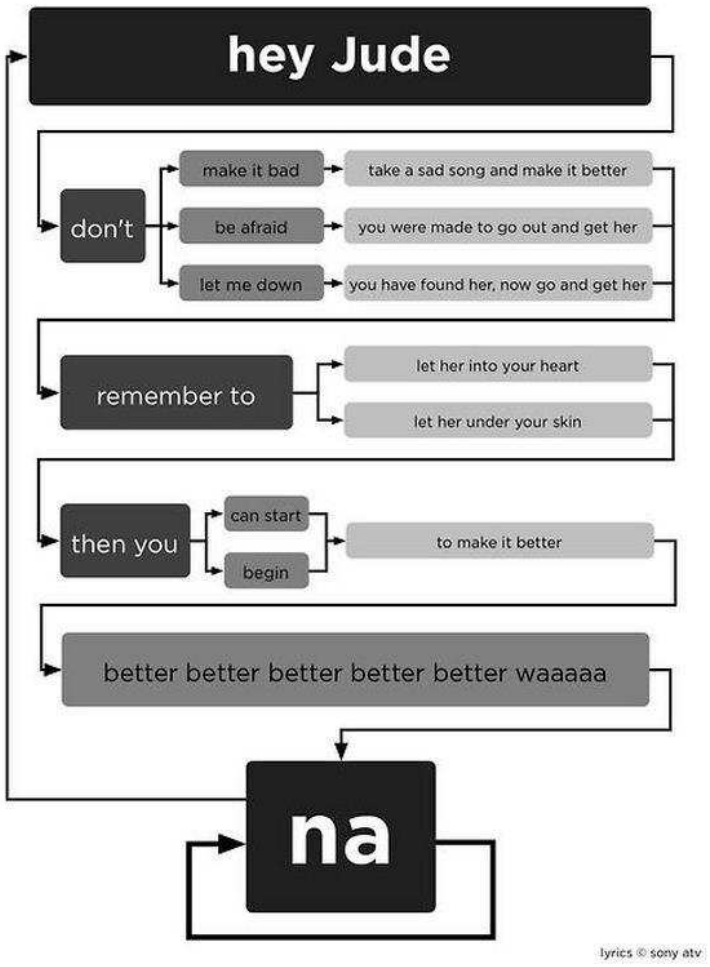
A música começa com uma estrutura que conhecemos, a repetição pré-testada (linha 03). A condição de execução é se estrofe é menor que 3, isso porque

somente temos três estrofes na música. A seguir vemos uma estrutura de decisão composta (linha 07), todas avaliando a variável "estrofe".

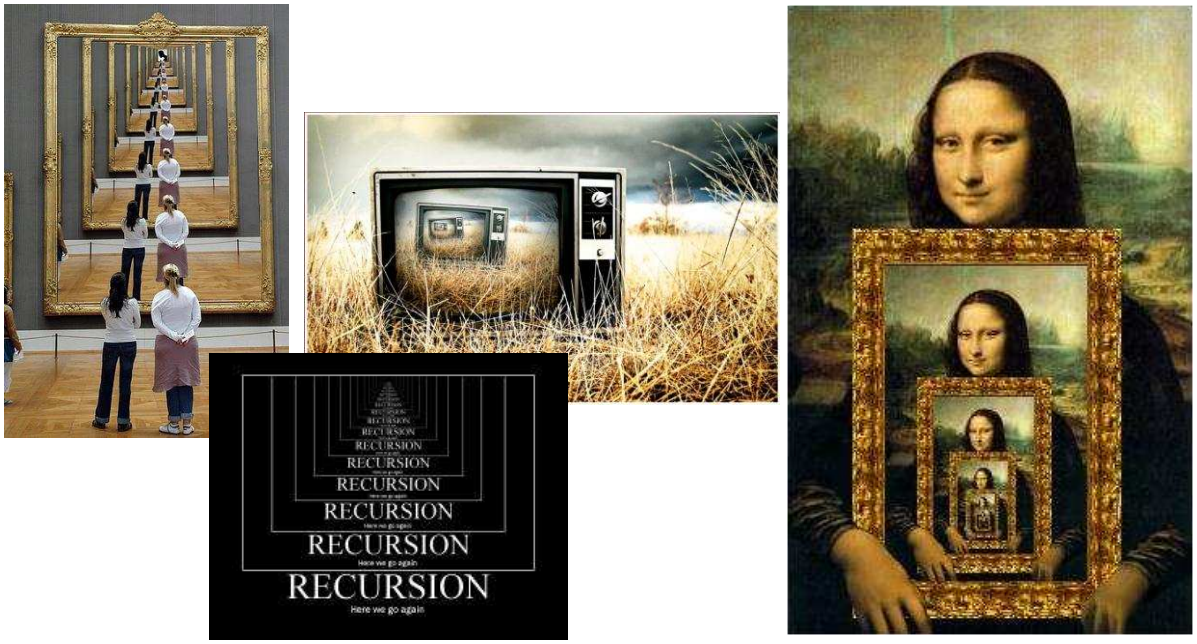
Para cada um dos valores de estrofe (1, 2 e 3) um bloco em separado é executado. Após o primeiro "Se Então Senão", temos outro, que testa se estrofe é par ou ímpar (linha 28). Essa estrutura também é composta, pois possui um bloco "Senão".

Por fim, temos uma estrutura de seleção (decisão) simples (linha 35), ou seja, somente com bloco "Se Então" e, dentro desse, encontramos uma estrutura de repetição pós-testada (linha 37). No exemplo lúdico essa repetição nunca termina (*pois ela termina em fade e parece continuar, lembram? :)*), pois a condição "Verdadeiro" sempre está satisfeita. Nos programas reais temos que lançar mão de algum escape.

*Todo mundo cantou bem alto pro vizinho ouvir? Ficou difícil? Tirei do diagrama abaixo, assim acho mais fácil de visualizar. Abraços e até a próxima.*



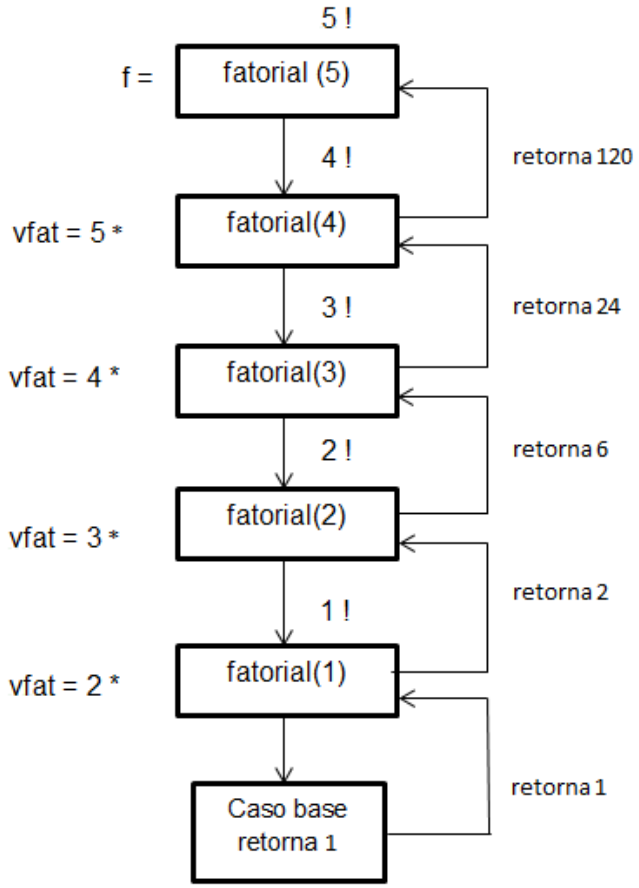
Por fim, vamos falar sobre recursividade! *O que é isso, professor?* Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que uma função chama a si mesma. Eu gosto de pensar em recursividade por meio de imagens como as apresentadas abaixo.




Nós dissemos que se trata de uma função que chama a si mesma e, daí, temos o nosso primeiro problema. *Se ela chama a si mesma sempre, não entra em um loop infinito?* Sim, por isso é necessário um caso-base (solução trivial), i.e., um caso mais simples em que é utilizada uma condição de parada, que resolve o problema. Toda recursividade é composta por um caso base e pelas chamadas recursivas.

Vantagens da recursividade: torna a escrita do código mais simples, elegante e, muitas vezes, menor. Desvantagens da recursividade: quando o loop recursivo é muito grande, é consumida muita memória nas chamadas a diversos níveis de recursão, tendo em vista que cada chamada recursiva aloca memória para os parâmetros, variáveis locais e de controle.

Em muitos casos, uma solução iterativa gasta menos memória e torna-se mais eficiente em termos de performance do que usar recursão. *Galera, vocês se lembram que estudaram fatorial na escola?* Pois é, fatorial é uma operação matemática em que um número natural (representado por $n!$) é o produto de todos os inteiros positivos menores ou iguais a $n$.




*Professor, facilita aí. Já saí da escola há muito tempo.* Ok! Fatorial(5) = 5x4x3x2x1; uma maneira diferente de escrever isso é: Fatorial(5) = 5 x Fatorial(4), que é o mesmo que dizer 5 x (4 x Fatorial(3)), que é o mesmo que 5 x (4 x (3 x Fatorial(2))), que é o mesmo que 5 x (4 x (3 x (2 x Fatorial(1))))). *E agora, professor?* Agora é a vez do caso-base! No caso do fatorial, o caso-base é: Fatorial(1) = 1. Todas essas instâncias ficaram em memória aguardando a chegada do caso-base e agora retornamos com os resultados. Dado o caso-base, temos que: 5 x (4 x (3 x (2 x Fatorial(1)))), logo 5 x (4 x (3 x (2 x 1))) = 120. Vejam ao lado a imagem da a execução de um código representando o fatorial.

Por fim, gostaria de mencionar que existem dois tipos de recursividade: direta e indireta. De modo bem simples, a recursividade direta é aquela tradicional em que uma função chama a si mesma (Ex: Função A chama a Função A); a recursividade indireta é aquela alternativa em que uma função chama outra função que chama a primeira (Ex: Função A chama a Função B, que chama a Função A).


1. (FCC - 2010 - DPE-SP - Agente de Defensoria - Analista de Sistemas) É utilizada para avaliar uma determinada expressão e definir se um bloco de código deve ou não ser executado. Essa é a definição da estrutura condicional:

   a) For
   b) If...Then...Else
   c) While
   d) Do...While
   e) Next

Comentários:

Pessoal... falou em estrutura *condicional*, trata-se de decisão! Logo, é o If-Then-Else.

Gabarito: B

---

2. (FCC - 2010 – TRT/SE - Analista de Sistemas) Objeto que se constitui parcialmente ou é definido em termos de si próprio. Nesse contexto, um tipo especial de procedimento (algoritmo) será utilizado, algumas vezes, para a solução de alguns problemas. Esse procedimento é denominado:

   a) Recursividade.
   b) Rotatividade.
   c) Repetição.
   d) Interligação.
   e) Condicionalidade.

Comentários:

> *Por fim, vamos falar sobre recursividade! O que é isso, professor? Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que*

*uma função chama a si mesma. Eu gosto de pensar em recursividade por meio de imagens como as apresentadas abaixo.*

Conforme vimos em aula, trata-se da recursividade.

Gabarito: A

---

3. (FCC - 2009 – TJ/SE - Analista de Sistemas) A recursividade na programação de computadores envolve a definição de uma função que:

a) apresenta outra função como resultado.
b) aponta para um objeto.
c) aponta para uma variável.
d) chama uma outra função.
e) pode chamar a si mesma.

Comentários:

*Por fim, vamos falar sobre recursividade! O que é isso, professor? Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que uma função chama a si mesma. Eu gosto de pensar em recursividade por meio de imagens como as apresentadas abaixo.*

Conforme vimos em aula, ela pode chamar a si mesma.

Gabarito: E

---

4. (CESPE - 2011 - TJ-ES - Técnico de Informática - Específicos) Uma estrutura de repetição possibilita executar um bloco de comando, repetidas vezes, até que seja encontrada uma dada condição que conclua a repetição.

Comentários:

Essa é uma definição perfeita da Estrutura de Repetição.

Gabarito: C

---

5. (CESPE - 2010 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Se um trecho de algoritmo tiver de ser executado repetidamente e o número

de repetições for indefinido, então é correto o uso, no início desse trecho, da estrutura de repetição Enquanto.

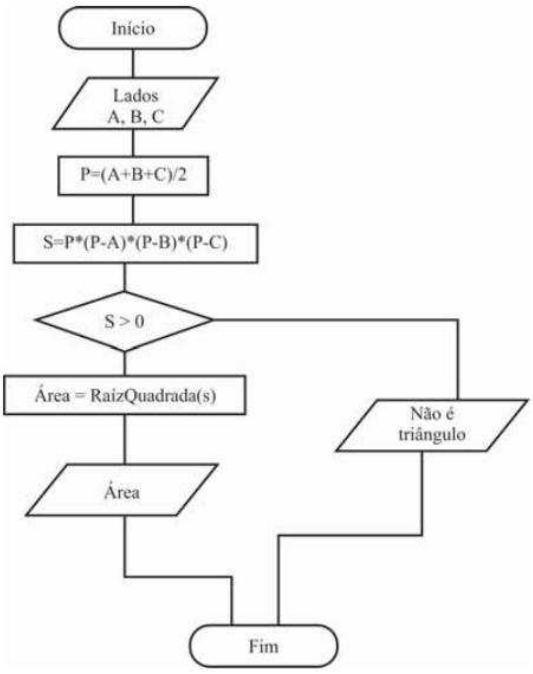
Comentários:

Perfeito! *Lembram-se que nós temos três estruturas de repetição?* Pois é, o Enquanto-Faça serve para esse propósito!

Gabarito: C

---

6. (CESPE - 2013 - CNJ - Programador de computador) No fluxograma abaixo, se A = 4, B = 4 e C = 8, o resultado que será computado para Área é igual a 32.




Comentários:

Vamos lá:

*P = (4+4+8)/2*

Lembrem-se que parênteses sempre têm prioridade:

*P = (16)/2 = 8*

Seguimos para S = P*(P-A)*(P-B)*(P-C):

*S = 8*(8-4)*(8-4)*(8-8)*

Simplificando:

*S = 8*4*4*0 = 0*

S <= 0, logo: Não é Triângulo!

Gabarito: E

7. (CESPE - 2011 - TJ-ES - Analista Judiciário - Análise de Banco de Dados - Específicos) Em uma estrutura de repetição com variável de controle, ou estrutura PARA, a verificação da condição é realizada antes da execução do corpo da sentença, o que impede a reescrita desse tipo de estrutura por meio de estrutura de repetição pós-testada.

Comentários:

Nós temos três tipos de estruturas de repetição com variável de controle: While/Enquanto, For/Para e Do-While/Faça-Enquanto - as duas primeiras pré-testadas e a última pós-testada. De fato, a estrutura For/Para é pré-testada. No entanto, é possível reescrevê-la como uma Estrutura de Repetição Pós-Testada (Do-While/Faça-Enquanto), de forma que elas sejam equivalentes.

Gabarito: E

8. (CESPE - 2010 – DETRAN/ES - Analista de Sistemas) O método de recursividade deve ser utilizado para avaliar uma expressão aritmética na qual um procedimento pode chamar a si mesmo, ou seja, a recursividade consiste em um método que, para que possa ser aplicado a uma estrutura, aplica a si mesmo para as subestruturas componentes.

Comentários:

> *Por fim, vamos falar sobre recursividade! O que é isso, professor? Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que uma função chama a si mesma. Eu gosto de pensar em recursividade por meio de imagens como as apresentadas abaixo.*

Conforme vimos em aula, a questão está correta. No entanto, sendo rigoroso, ele bem que podia ter usado um verbo diferente para evitar confusão (*"O método de recursividade pode ser utilizado..."* ou *"O método de recursividade é utilizado..."*).

Gabarito: C

9. (CESPE - 2013 – CPRM - Analista de Sistemas) Na implementação de recursividade, uma das soluções para que se evite o fenômeno de terminação do programa – que possibilita a ocorrência de um looping infinito – é definir uma função ou condição de terminação das repetições.

Comentários:

*Nós dissemos que se trata de uma função que chama a si mesma e, daí, temos o nosso primeiro problema. Se ela chama a si mesma sempre, não entra em um loop infinito? Sim, por isso é necessário um caso-base (solução trivial), i.e., um caso mais simples em que é utilizada uma condição de parada, que resolve o problema. Toda recursividade é composta por um caso base e pelas chamadas recursivas.*

Conforme vimos em aula, está perfeito novamente.

Gabarito: C

---

10. (CESPE - 2014 – ANATEL - Analista de Sistemas) A recursividade é uma técnica que pode ser utilizada na implementação de sistemas de lógica complexa, com a finalidade de minimizar riscos de ocorrência de defeitos no software.

Comentários:

*Por fim, vamos falar sobre recursividade! O que é isso, professor? Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que uma função chama a si mesma. Eu gosto de pensar em recursividade por meio de imagens como as apresentadas abaixo.*

Conforme vimos em aula, definitivamente essa não é uma finalidade da recursividade. Uma implementação simples não significa de maneira alguma mais eficiente. Em geral, um algoritmo iterativo (não-recursivo) é mais eficiente que um algoritmo recursivo (por conta da manutenção de estado, pilhas, etc). Portanto, não há essa correlação entre riscos de defeito e complexidade de implementação.

Gabarito: E

---

11. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Tanto a recursividade direta quanto a indireta necessitam de uma condição de saída ou de encerramento.

Comentários:

*Por fim, gostaria de mencionar que existem dois tipos de recursividade: direta e indireta. De modo bem simples, a recursividade direta é aquela tradicional em que uma função chama a si mesma (Ex: Função A chama a Função A); a recursividade indireta é aquela alternativa em que uma função chama outra função que chama a primeira (Ex: Função A chama a Função B, que chama a Função A).*

Conforme vimos em aula, existe recursividade direta e indireta, mas ambas precisam de um caso-base, que é uma condição de saída, para não ficar em loop infinito.

Gabarito: C

---

12. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo.

```
Atribuir 13 a X;
repetir
    atribuir X – 2 a X;
    imprimir (X);
Até que X < -1;
```

A estrutura será executada até que X seja igual ao seguinte valor
a) – 1
b) – 3

Comentários:

Vamos lá:

*X = 13*

No laço, ele afirma: X = X-2

*X = 13-2 = 11*

Segue esse loop novamente:

*X = 11-2 = 9*
*X = 9-2 = 7*
*X = 7-2 = 5*
*X = 5-2 = 3*
*X = 3-2 = 1*
*X = 1-2 = -1*
*X = -1-2 = -3 < -1, logo saímos do loop!*

Portanto, a estrutura é executada até X = -3.

Gabarito: B

13. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo, que mostra o emprego da estrutura de controle enquanto ... faça ...

```
atribuir 0 a n;
enquanto n < 7 faça
        início
          imprimir (n);
          atribuir n + 1 a n;
        fim;
```

A opção que utiliza a estrutura para ... faça ... correspondente, que gera o mesmo resultado, é:

a) Para n de 0 até 6 faça imprimir(n);
b) Para n de 0 até 7 faça imprimir(n);

Comentários:

Vejamos: N = 0 e irá até N < 7, logo N de 0 a 6.

Gabarito: A

14. (FEPESE - 2010 - SEFAZ-SC - Auditor Fiscal da Receita Estadual - Parte III - Tecnologia da Informação) Assinale a alternativa correta a respeito das variáveis e constantes, utilizadas em diversas linguagens de programação.

a) O número de constantes deve ser menor ou igual ao número de variáveis em um programa.
b) O número de constantes deve ser menor ou igual ao número de procedimentos em um programa.
c) O número de constantes deve ser igual ao número de variáveis em um programa.
d) O número de constantes independe da quantidade de variáveis em um programa.
e) O número de constantes deve ser igual ao número de procedimentos em um programa.

Comentários:

Galera, não há essa relação! O número de constantes e variáveis são independentes.

Gabarito: D

---

15. (NUCEPE - 2015 – SEDUC/PI - Analista de Sistemas) O código abaixo é usado para calcular o fatorial de números. Assinale a alternativa CORRETA sobre esse código:

```
função fatorial(n)
{
  se (n <= 1)
    retorne 1;
  senão
    retorne n * fatorial(n-1);
}
```

a) Este é um exemplo de procedimento.

b) O comando retorne pode ser retirado do código e a função terá o mesmo efeito.

c) Exemplo clássico de recursividade.

d) Não é possível chamar a função fatorial dentro dela mesma.

e) O resultado da função sempre retornará um valor elevado a ele mesmo (valor ^ valor).

Comentários:

> *Em muitos casos, uma solução iterativa gasta menos memória e torna-se mais eficiente em termos de performance do que usar recursão. Galera, vocês se lembram que estudaram fatorial na escola? Pois é, fatorial é uma operação matemática em que um número natural (representado por* ***n!****) é o produto de todos os inteiros positivos menores ou iguais a* ***n****.*

Conforme vimos em aula, trata-se de um exemplo clássico de recursividade.

Gabarito: C

---

16. (IADES - 2011 – PG/DF - Analista Jurídico - Analista de Sistemas) Os algoritmos são compostos por estruturas de controle de três tipos: sequencial, condicional e de repetição. Assinale a alternativa que apresenta apenas um tipo de estrutura de controle.

a) ...
```
escreva (“Digite seu nome: ”)
leia (nome)
escreva (“Digite sua idade: ”)
leia (idade)
limpe a tela
escreva (“Seu nome é:”, nome)
escreva (“Sua idade é:”, idade)
se (nome = “João”) entao
   se (idade > 18) entao
      escreva (nome, “ é maior de 18 anos!”)
   fim se
fim se
...
```

b) ...
```
escreva (“Pressione qualquer tecla para começar...”)
leia (tecla)
mensagem ← “Não devo acordar tarde...”
numero ← 0
enquanto (numero < 100)
      escreva (mensagem)
      numero ← (numero + 1)
fim enquanto
escreva (“Pressione qualquer tecla para
terminar...”)
leia (tecla)
escreva (“Tecla digitada: ”)
escreva (tecla)
...
```

c) ...
```
leia (nome)
escreva (“nome digitado: ”)
```

```
    escreva (nome)
    se (nome = "Wally") entao
        escreva ("Encontrado o Wally!")
    senao
        cont ← 5
        enquanto (cont > 0)
          escreva ("Não é Wally"...")
          cont ← (cont – 1)
        fim enquanto
    fim se
    ...
```

d) ...

```
    var
      nome: literal
      num: inteiro
    inicio
      escreva ("Digite seu nome: ")
      leia (nome)
      num ← 0
      se (nome = "José") entao
          num ← (num + 1)
      fim se
     escreva ("Quantidade de João encontrados:
    ")
     escreva (num)
    ...
```

e) ...

```
    var
       nome: literal
       idade: inteiro
    inicio
      escreva ("Digite seu nome: ")
      leia (nome)
      escreva ("Digite sua idade: ")
      leia (idade)
      limpe a tela
      escreva ("Seu nome é:")
      escreva (nome)
```

```
    escreva ("Sua idade é:")
    escreva (idade)
fim algoritmo
...
```

Comentários:

*Como bem definido na questão, temos estruturas sequenciais, repetição e de decisão. As estruturas sequenciais são aquelas que não tem desvios na execução, ou seja, uma é executada após a outra. Analisando os itens:*
*a) possui instruções sequenciais e também uma estrutura de decisão (uma delas aninhada), ou seja, temos dois tipos*
*b) possui instruções sequenciais e também uma estrutura de repetição.*
*c) possui instruções sequenciais, uma estrutura de decisão e uma de repetição.*
*d) possui instruções sequenciais e também uma estrutura de decisão.*
*e) não possui estruturas de decisão (seleção), nem de repetição, somente sequenciais.*

*Ou seja, resposta é a letra E*

Gabarito: E

17. (IADES - 2011 – TRE-PA - Programador de Computador)

```
VAR
N1, N2 : INTEIRO;
N1 ← 2;
N2 ← 30;
INICIO
ENQUANTO N1<N2 FAÇA
        N2 ← N2 + N1;
        N1 ← N1 * 3;
FIM ENQUANTO;
N1 ← N2 + 11;
FIM
```

Dado o algoritmo escrito em pseudocódigo, quais os valores de N1 e N2, respectivamente, ao final da execução?
a) 162 e 110.
b) 110 e 121.
c) 110 e 162.

d) 121 e 110.
e) 173 e 110.

Comentários:

*A questão traz um exemplo de estrutura de repetição pré-testada e quer saber se vocês entenderam bem a aula teórica! ☺*

*Vamos executar passo a passo para realizar os testes e mostrar como funciona cada iteração (repetição) do bloco da estrutura de repetição?*

*1ª iteração:*

*N1 é igual a 2 (atribuição da 3ª linha)*
*N2 é igual a 3 (atribuição da 4ª linha)*
*Teste: N1 < N2?? Sim, pois 2 é menor que 3!! Então entramos no loop para executar as instruções do bloco da estrutura*
*N2 recebe 30 + 2*
*N1 recebe 2 * 3*

*2ª iteração:*

*N1 é igual a 6*
*N2 é igual a 32*
*Teste: N1 < N2?? Sim, pois 6 é menor que 32!! Então entramos no loop para executar as instruções do bloco da estrutura*
*N2 recebe 32 + 6*
*N1 recebe 6 * 3*

*3ª iteração:*

*N1 é igual a 18*
*N2 é igual a 38*
*Teste: N1 < N2?? Sim, pois 18 é menor que 38!! Então entramos no loop para executar as instruções do bloco da estrutura*
*N2 recebe 38 + 18*
*N1 recebe 18 * 3*

*4ª iteração:*

*N1 é igual a 54*
*N2 é igual a 56*
*Teste: N1 < N2?? Sim, pois 54 é menor que 56!! Então entramos no loop para executar as instruções do bloco da estrutura*
*N2 recebe 56 + 54*
*N1 recebe 54 * 3*

*5ª iteração:*

*N1 é igual a 162*
*N2 é igual a 110*
*Teste: N1 < N2?? Não (finalmente!!! :D), pois 162 não é menor que 110!! Desse modo, saímos do loop e voltamos para a instrução imediatamente posterior, ou seja:*

*N1 recebe 110 + 11*

*FIM DO CÓDIGO*

*Ao final do código, N1 tem o valor de 121 e N2 de 110*

*Resposta letra D*

Gabarito: D

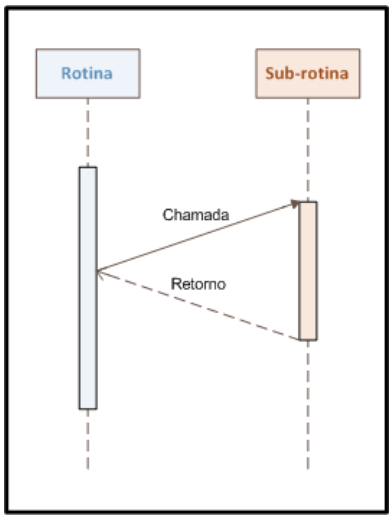
| ACERTEI | ERREI |
| --- | --- |
| | |

## FUNÇÕES E PROCEDIMENTOS

Pessoal, é importante falarmos de alguns conceitos fundamentais! *O que é uma rotina?* Uma rotina nada mais é que um conjunto de instruções – um algoritmo. Uma sub-rotina (ou subprograma) é um pedaço menor desse conjunto de instruções que, em geral, será utilizado repetidamente em diferentes locais do sistema e que resolve um problema mais específico, parte do problema maior.

Em vez de repetir um mesmo conjunto de instruções diversas vezes no código, esse conjunto pode ser agrupado em uma sub-rotina que é chamada em diferentes locais. As sub-rotinas podem vir por meio de uma função ou de um procedimento. A diferença fundamental é que, no primeiro caso, retorna-se um valor e no segundo caso, não. *Entenderam?*

Galera, essa é a ideia geral! No entanto, algumas linguagens de programação têm funções e procedimentos em que nenhum retorna valor ou ambos retornam valor, ou seja, muitas vezes, sequer há uma distinção. No contexto da programação orientada a objetos, estas sub-rotinas são encapsuladas nos próprios objetos, passando a designar-se métodos.




A criação de sub-rotinas foi histórica e permitiu fazer coisas fantásticas. Não fossem elas, estaríamos fazendo *goto* até hoje. Uma sub-rotina é um trecho de código marcado com um ponto de entrada e um ponto de saída. *Entenderam?*

Quando uma sub-rotina é chamada, ocorre um desvio do programa para o ponto de entrada, e quando a sub-rotina atinge seu ponto de saída, o programa desaloca a sub-rotina, e volta para o mesmo ponto de onde tinha saído. Vejam a imagem ao lado!

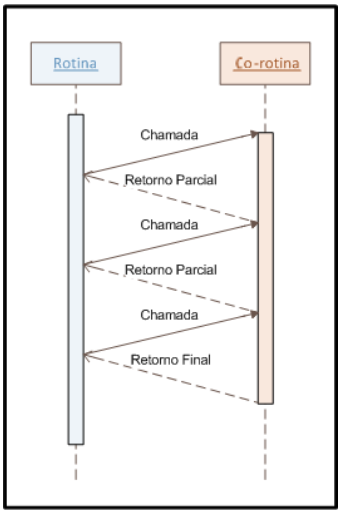
Algumas das vantagens na utilização de sub-rotinas durante a programação são: redução de código duplicado; possibilidade de reutilizar o mesmo código sem grandes alterações em outros programas; decomposição de problemas grandes em pequenas partes; melhorar a interpretação visual; esconder ou regular uma parte de um programa; reduzir o acoplamento; entre outros.

Quando uma sub-rotina termina, ela é desalocada das estruturas internas do programa, i.e., seu tempo de vida termina! É praticamente como se ela nunca tivesse existido. Podemos chamá-la de novo, isso é claro, mas será uma nova encarnação da sub-rotina: o programa novamente começará executando desde seu ponto de entrada. Uma co-rotina não funciona assim!

As co-rotinas são parecidas com as sub-rotinas, diferindo apenas na forma de transferência de controle, realizada na chamada e no retorno. Elas possuem um ponto de entrada e podemos dizer que são mais genéricas que as sub-rotinas. Na co-rotina, o trecho de código trabalha conjuntamente com o código chamador até que sua tarefa seja terminada.

O controle do programa é passado da co-rotina para o chamador e de volta para a co-rotina até que a tarefa da co-rotina termine. Numa co-rotina, existe o ponto de entrada (que é por onde se inicia o processamento), um ponto de saída (pela qual o tempo de vida da co-rotina termina), e um ponto de retorno parcial. Esse retorno parcial não termina com a vida da co-rotina.

Pelo contrário, apenas passa o controle do programa para a rotina chamadora, e marca um ponto de reentrada. Quando a rotina chamadora devolver o controle para a co-rotina, o processamento começa a partir do último ponto de retorno parcial.




Na prática, uma co-rotina permite retornar valores diversas vezes antes de terminar o seu tempo de vida. Enquanto o tempo de vida das sub-rotinas é ditado pela pilha de execução, o tempo de vida das co-rotinas é ditado por seu uso e necessidade.


1. (CONSULPLAN - 2012 - TSE – Técnico – Programação de Sistemas)

Analise o pseudocódigo, que ilustra o uso de uma função recursiva.

```
programa PPRRGG;
variáveis
   VERDE, AZUL : numérica;
função FF(AUX:numérica):numérica;
início
   atribuir VERDE+1 a VERDE;
   se AUX <= 2
   então atribuir 5 a FF
   senão atribuir AUX*FF(AUX-1) a FF;
fim; { fim da função FF }
início
  atribuir 0 a VERDE;
  atribuir FF(4) a AZUL;
  escrever(VERDE,AZUL);
fim.
```

O valor de retorno de FF e a quantidade de vezes que a função será executada serão, respectivamente,
(A) 5 e 1.
(B) 15 e 2.
(C) 60 e 3.
(D) 300 e 4.

Comentários:

Galera, como vimos, as sub-rotinas que chamam a si mesmas são chamadas de recursivas. Na questão temos a função FF, que na última linha faz uma chamada a ela mesma.

Vamos fazer o passo a passo para entender como funciona a função, lembrando que a recursividade requer um critério de parada a partir de algum teste de um valor da variável usada como controle.

Segundo o início do código temos:

VERDE tem o valor 0
AZUL tem o valor retornado por FF(4)

### 1ª chamada da função FF

AUX tem o valor 4 (passagem de parâmetro na chamada do programa principal)
VERDE RECEBE VERDE + 1, ou seja, VERDE RECEBE 1
AUX <= 2? Ou ainda, 4 <= 2? Não! Bloco senão é executado
FF retorna AUX * FF(AUX-1), ou seja, FF retorna 4 * FF(3)

### 2a chamada da função FF

AUX tem o valor 3 (passagem de parâmetro na chamada recursiva)
VERDE RECEBE VERDE + 1, ou seja, VERDE RECEBE 2

*/* Opa, opa, professor, porque VERDE continua com o valor que recebeu na 1a chamada de FF? Ora, meu caro padawan, porque as variáveis VERDE e AZUL foram declaradas fora da função num escopo mais geral, ou seja, as alterações dentro da função valem fora dela e para todas as suas chamadas recursivas. Agora de volta ao código :) */*

AUX <= 2? Ou ainda, 3 <= 2? Não! Bloco senão é executado
FF retorna AUX * FF(AUX-1), ou seja, FF retorna 3 * FF(2)

### 3a chamada da função FF

AUX tem o valor 2 (passagem de parâmetro na chamada recursiva)

VERDE RECEBE VERDE + 1, ou seja, VERDE RECEBE 3
AUX <= 2? Ou ainda, 3 <= 3? Sim! Bloco então é executado
FF retorna 5

*/* Agora é hora de voltar nas chamadas recursivas, do último para o primeiro
Isso lembra alguma coisa?? LIFO?? Pilhas! São utilizadas nas chamadas
recursivas para voltar a execução na hierarquia das chamadas */*

### Voltando à 2a chamada de FF

FF retorna 3 * 5, ou seja, FF retorna 15

Voltando à 1a chamada de FF

FF retorna 4 * FF(3), ou seja, FF retorna 4 * 15, FF retorna 60

Ou seja, o valor de FF(4) é 60 e a função FF é chamada 3 vezes.

Gabarito: C

| ACERTEI | ERREI |
| --- | --- |
| | |

## COMPLEXIDADE DE ALGORITMOS

*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[3] implementado.

[3] 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.

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 (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.

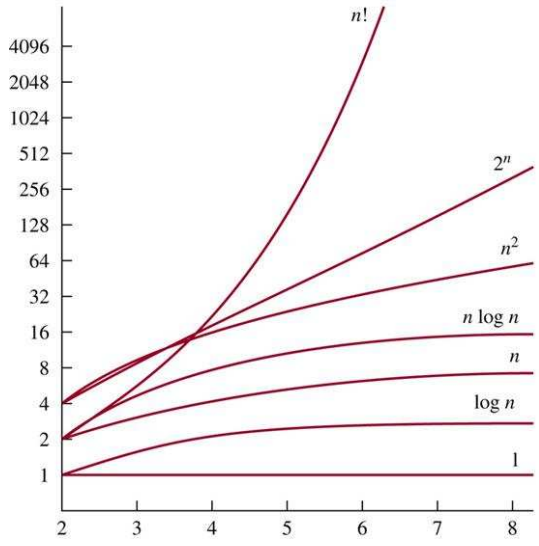
E trata da complexidade porque se é necessário 1 segundo para ordenar 10.000 elementos, quanto tempo levará para ordenar 1.000.000? A complexidade, nesse exemplo particular, é a medida relativa para alguma coisa. Vamos ver isso por meio de um exemplo: soma de dois inteiros! A soma é uma operação ou um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


1. (VUNESP – 2012 – TJ/SP - Analista Judiciário - Tecnologia da Informação) Considerando o conceito de Complexidade de Algoritmos, representado por O(função), assinale a alternativa que apresenta, de forma crescente, as complexidades de algoritmos.

a) $O(2^n)$; $O(n^3)$; $O(n^2)$; $O(\log_2 n)$; $O(n.\log_2 n)$.
b) $O(n^2)$; $O(n^3)$; $O(2^n)$; $O(\log_2 n)$; $O(n.\log_2 n)$.
c) $O(n^3)$; $O(n^2)$; $O(2^n)$; $O(n.\log_2 n)$; $O(\log_2 n)$.
d) $O(\log_2 n)$; $O(n.\log_2 n)$; $O(n^2)$; $O(n^3)$; $O(2^n)$.
e) $O(n.\log_2 n)$; $O(\log_2 n)$; $O(2^n)$; $O(n^3)$; $O(n^2)$.

Comentários:

| 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 |

Conforme vimos em aula, basta consultar a tabelinha!

Gabarito: D

---

2. (FCC - 2010 - TRT - 8ª Região (PA e AP) - Analista Judiciário - Tecnologia da Informação) Numa competição de programação, ganhava mais pontos o time que apresentasse o algoritmo mais eficiente para resolver o pior caso de um

determinado problema. A complexidade assintótica (notação Big O) dos algoritmos elaborados está ilustrada na tabela abaixo.

| Time | Complexidade |
|---|---|
| Branco | $O(n^{20})$ |
| Amarelo | O(nlogn) |
| Azul | O(1) |
| Verde | O(n!) |
| Vermelho | $O(2^n)$ |

O time que obteve a medalha de prata (2º algoritmo mais eficiente) é o:

a) Branco.
b) Amarelo.
c) Azul.
d) Verde.
e) Vermelho.

Comentários:

| 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 |

Conforme vimos em aula, basta consultar a tabelinha! O primeiro é o time Azul ($O(1)$) e o segundo é o time Amarelo ($O(n \log n)$).

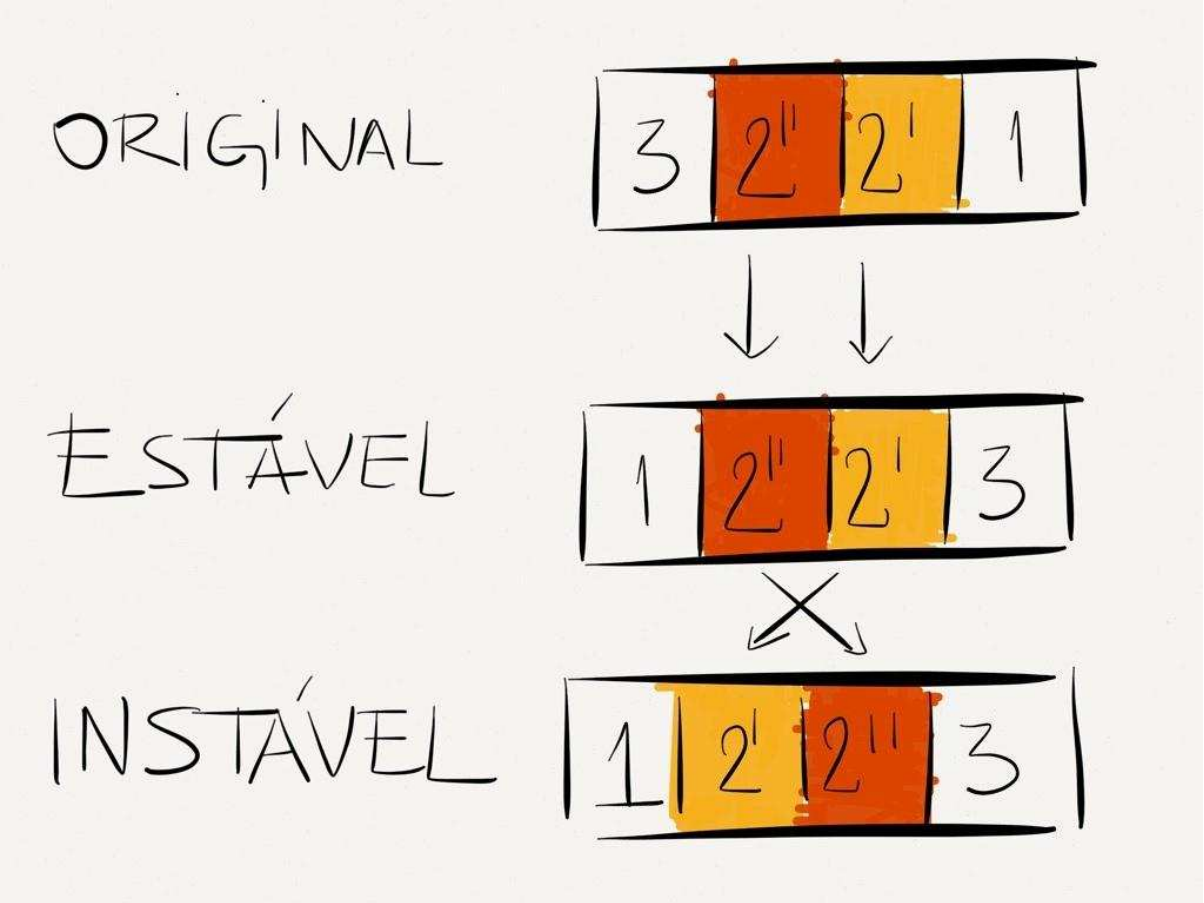
Gabarito: B

| ACERTEI | ERREI |
|---|---|
| | |

## MÉTODOS DE ORDENAÇÃO

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.

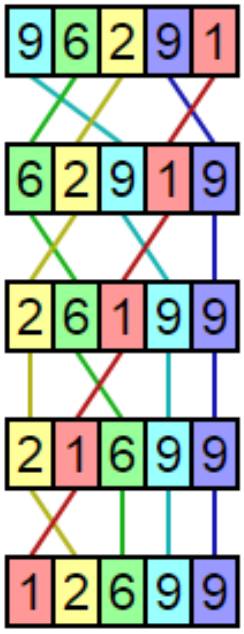
Antes de iniciar, vamos falar sobre o conceito de Estabilidade! Um método estável é aquele em que os itens com chaves iguais mantêm-se com a posição inalterada durante o processo de ordenação, i.e., preserva-se a ordem relativa dos itens com chaves duplicadas ou iguais. Métodos Estáveis: Bubble, Insertion 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).

## BubbleSort (Troca)




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}
```

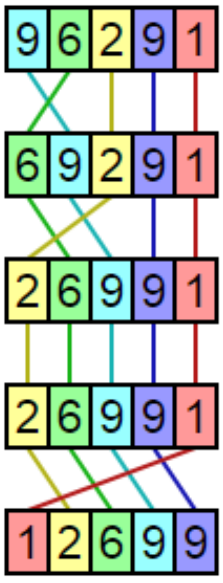
| Melhor Caso | Caso Médio | Pior Caso |
|---|---|---|
| $\Omega(n)$ | $\Omega(n^2)$ | $\Omega(n^2)$ |

## InsertionSort (Inserção)

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}
```

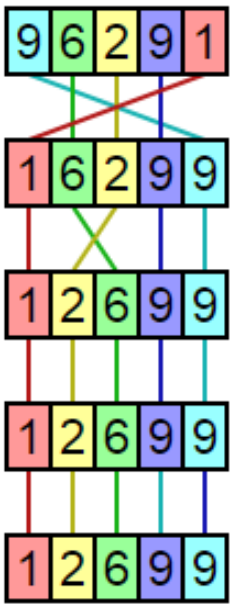



Nesse método, consideramos que a lista está dividida em parte esquerda, já ordenada, e parte direita, em possível desordem. Inicialmente, a parte esquerda contém apenas o primeiro elemento da lista. Cada iteração consiste em inserir o primeiro elemento da parte direta (pivô) na posição adequada da parte 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 |
|---|---|---|
| $\Omega(n)$ | $\Omega(n^2)$ | $\Omega(n^2)$ |

## SelectionSort (Selecão)




Esse algoritmo consiste em selecionar o menor 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.

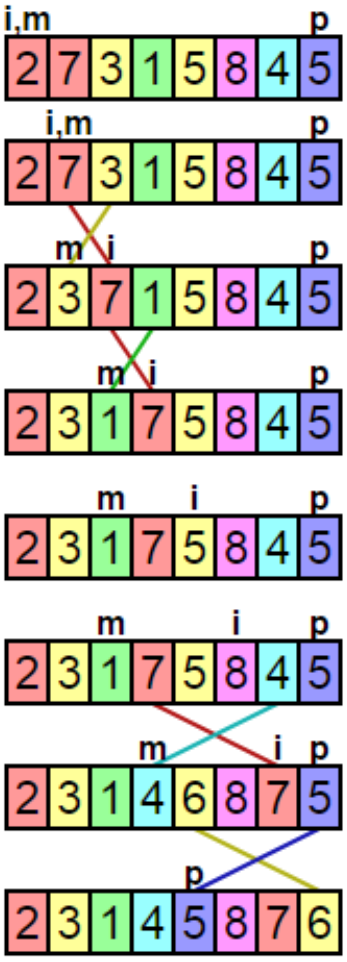
Assim, a parte esquerda aumenta, visto que passa a incluir o pivô, e a parte direita diminui. Note que o pivô é maior que todos os demais elementos da parte esquerda, portanto a parte esquerda continua ordenada. Ademais, o pivô era o menor elemento da direita, logo todos os elementos da esquerda continuam 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 |
|---|---|---|
| $\Omega(n^2)$ | $\Theta(n^2)$ | $O(n^2)$ |

## QuickSort (Troca)




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 |
|---|---|---|
| $\Omega(n \log n)$ | $\Theta(n \log n)$ | $O(n^2)$ |

## ShellSort (Inserção)

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.

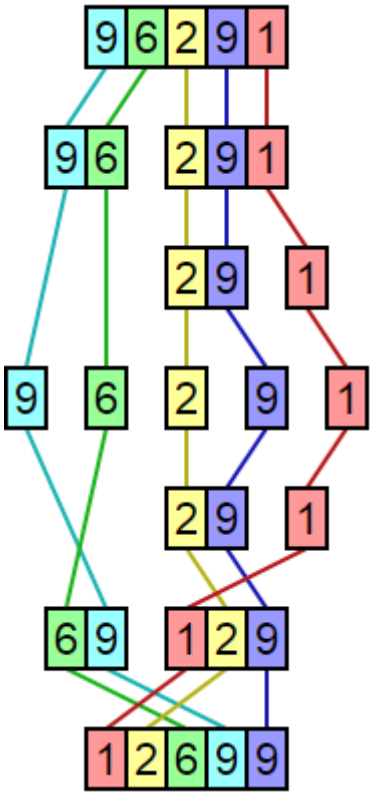
É o algoritmo mais eficiente dentre os de ordem quadrática. Nesse método, as comparações e as trocas são feitas conforme determinada distância *(gap)* entre dois elementos, de modo que, se gap = 6, há comparação entre o 1º e 7º elementos ou entre o 2º e 8º elementos e assim sucessivamente, repetindo 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* | $O(n^2)$ |

## MergeSort (Intercalação)




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.

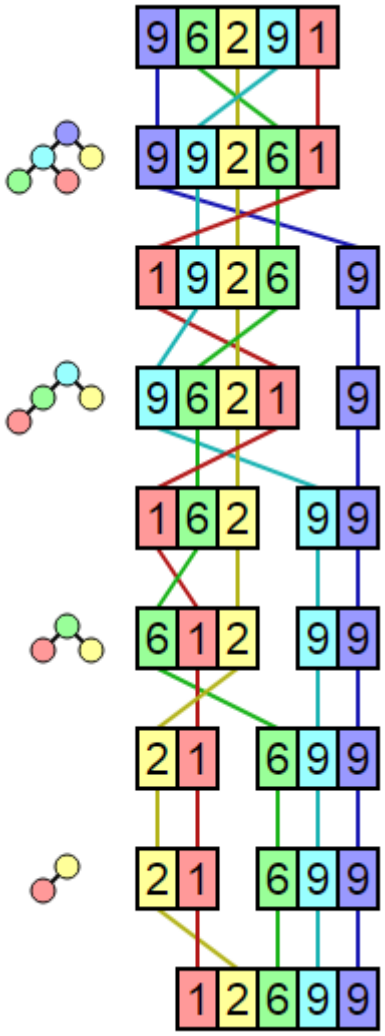
Além disso, deve ser sempre possível utilizar as soluções obtidas com a resolução das instâncias menores para chegar a uma solução da instância original. No MergeSort, divide-se a lista em duas metades. Essas metades são ordenadas recursivamente (usando o próprio MergeSort) e depois são intercaladas.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

## HeapSort (Seleção)




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) |

## RESUMÃO DE 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) |


1. (Instituto Cidades - 2012 - TCM-GO - Auditor de Controle Externo - Informática) São exemplos de algoritmos de ordenação, exceto:

   a) Bubble Sort
   b) Select Sort
   c) Shell Sort
   d) Busca Sequencial;
   e) Quick Sort;

Comentários:

Conforme vimos em aula, a Busca Sequencial não é um algoritmo de ordenação! Na verdade, ele é um método de pesquisa sobre estruturas de dados.

Gabarito: D

---

2. (FUMARC - 2012 - TJ-MG - Técnico Judiciário - Analista de Sistemas – I) Quicksort divide um conjunto de itens em conjuntos menores, que são ordenados de forma independe, e depois os resultados são combinados para produzir a solução de ordenação do conjunto maior.

Comentários:

Conforme vimos em aula, está perfeito! Sendo um algoritmo do tipo Dividir Para Conquistar, ele reparte o conjunto de dados em conjuntos menores, que são ordenados independentemente e depois combinados em uma solução maior.

Gabarito: C

---

3. (CESPE - 2012 - MPE-PI - Analista Ministerial - Informática - Cargo 6) O heapsort é um algoritmo de ordenação em que a quantidade de elementos armazenada fora do arranjo de entrada é constante durante toda a sua execução.

Comentários:

*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.*

Essa questão é polêmica. O arranjo tem tamanho constante, mas a quantidade de elementos é variável. Diferente de outros algoritmos de ordenação que tem uma estrutura auxiliar de tamanho variável (assim como seus elementos), o Heap Sort tem uma estrutura auxiliar de tamanho fixo (porém a quantidade de elementos é variável). Como é dito por Neil Dale: "*A heapsort is just as efficient in terms of space; only one array is used to store the data. The heap sort requires only constante extra space*". No entanto, a questão foi dada como correta!

Gabarito: C

---

4. (CESPE - 2010 - ABIN - Oficial Técnico de Inteligência - Área de Suporte a Rede de Dados) A eficácia do método de ordenação rápida (quicksort) depende da escolha do pivô mais adequado ao conjunto de dados que se deseja ordenar. A situação ótima ocorre quando o pivô escolhido é igual ao valor máximo ou ao valor mínimo do conjunto de dados.

Comentários:

*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.*

Conforme vimos em aula, a questão se refere ao pior caso!

Gabarito: E

---

5. (CESPE - 2010 - ABIN - Oficial Técnico de Inteligência - Área de Suporte a Rede de Dados) A estabilidade de um método de ordenação é importante quando o conjunto de dados já está parcialmente ordenado.

Comentários:

> *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).*

Conforme vimos em aula, a estabilidade é irrelevante com dados parcialmente ordenados ou não! A estabilidade é importante quando se deseja ordenar um conjunto de dados por mais de um critério (Ex: primeiro pelas chaves e segundo por índices). Se esse não for o caso (e a questão não disse que era!), a estabilidade "não fede nem cheira". O fato de os dados estarem parcialmente ordenados não fará diferença em termos de ordenação – ambos serão ordenados da mesma maneira.

Gabarito: E

---

6. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Administração de Dados) A classificação interna por inserção é um método que realiza a ordenação de um vetor por meio da inserção de cada elemento em sua posição correta dentro de um subvetor classificado.

Comentários:

Conforme vimos em aula, trata-se do InsertionSort!

Gabarito: C

---

7. (FCC - 2009 - TRT - 15ª Região - Analista Judiciário - Tecnologia da Informação) São algoritmos de classificação por trocas apenas os métodos:

a) SelectionSort e InsertionSort.
b) MergeSort e BubbleSort.
c) QuickSort e SelectionSort.
d) BubbleSort e QuickSort.

e) InsertionSort e MergeSort.

Comentários:

Conforme vimos em aula, BubbleSort e QuickSort são métodos de troca; InsertionSort é um método de Inserção; SelectionSort e HeapSort são métodos de Seleção; e MergeSort é um método de Intercalação.

Gabarito: D

---

8. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – I) O tempo de pior caso do algoritmo QuickSort é de ordem menor que o tempo médio do algoritmo Bubblesort.

Comentários:

| 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 | $\Omega(n \log n)$ | $\Theta(n \log n)$ | $O(n \log n)$ |

Conforme vimos em aula, está incorreto! São iguais: $O(n^2)$.

Gabarito: E

---

9. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – II) O tempo médio do QuickSort é $O(n\log_2 n)$, pois ele usa como estrutura básica uma árvore de prioridades.

Comentários:

| 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 | Θ(n log n) | Θ(n log n) | Θ(n log n) |
| HeapSort | Θ(n log n) | Θ(n log n) | Θ(n log n) |

Conforme vimos em aula, de fato, ele tem tempo médio O(n log n), mas ele usa como estrutura básica uma lista ou um vetor!

Gabarito: E

---

10. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – III) O tempo médio do QuickSort é de ordem igual ao tempo médio do MergeSort.

Comentários:

| Algoritmo | Melhor Caso | Caso Médio | Pior Caso |
|---|---|---|---|
| BubbleSort | Θ(n) | Θ($n^2$) | Θ($n^2$) |
| InsertionSort | Θ(n) | Θ($n^2$) | Θ($n^2$) |
| SelectionSort | Θ($n^2$) | Θ($n^2$) | Θ($n^2$) |
| QuickSort | Θ(n log n) | Θ(n log n) | Θ($n^2$) |
| ShellSort | Θ(n log n) | Depende do *gap* | Θ($n^2$) |
| MergeSort | Θ(n log n) | Θ(n log n) | Θ(n log n) |
| HeapSort | Θ(n log n) | Θ(n log n) | Θ(n log n) |

Conforme vimos em aula, ambos têm tempo O(n log n).

Gabarito: C

---

11. (CESGRANRIO - 2012 - CMB – Analista de Sistemas – III) Em uma reunião de análise de desempenho de um sistema WEB, um programador apontou corretamente que a complexidade de tempo do algoritmo bubblesort, no pior caso, é:

a) O(1)
b) O(log n)
c) O(n)
d) O(n log n)
e) O($n^2$)

Comentários:

| 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 | $\Omega(n \log n)$ | $\Theta(n \log n)$ | $O(n \log n)$ |

Conforme vimos em aula, trata-se de $O(n^2)$!

Gabarito: E

---

12. (CESPE - 2010 – INMETRO – Analista de Sistemas) Se f é uma função de complexidade para um algoritmo F, então O(f) é considerada a complexidade assintótica ou o comportamento assintótico do algoritmo F. Assinale a opção que apresenta somente algoritmos que possuem complexidade assintótica quando f(n) = O(n log n).

a) HeapSort e BubbleSort
b) QuickSort e InsertionSort
c) MergeSort e BubbleSort
d) InsertionSort
e) HeapSort, QuickSort e MergeSort

Comentários:

| 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 | $\Omega(n \log n)$ | $\Theta(n \log n)$ | $O(n \log n)$ |

Conforme vimos em aula, trata-se da última opção. Vejam que ele não utilizou, nessa questão, o Pior Caso.

Gabarito: E

---

13. **(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

Comentários:

| 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 | $\Omega(n \log n)$ | $\Theta(n \log n)$ | $O(n \log n)$ |

Conforme vimos em aula, trata-se da segunda opção.

Gabarito: B

---

14. **(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

Comentários:

| 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) |

Conforme vimos em aula, trata-se da primeira opção!

Gabarito: A

---

15. (CESPE - 2011 – FUB – Analista de Sistemas) Os métodos de ordenação podem ser classificados como estáveis ou não estáveis. O método é estável se preserva a ordem relativa de dois valores idênticos. Alguns métodos eficientes como shellsort ou quicksort não são estáveis, enquanto alguns métodos pouco eficientes, como o método da bolha, são estáveis.

Comentários:

Métodos Estáveis: BubbleSort, InsertionSort e MergeSort; Métodos Instáveis: SelectionSort, QuickSort, HeapSort e ShellSort. Portanto, item perfeito!

Gabarito: C

---

16. (CESPE - 2012 – BASA – Analista de Sistemas) O método de classificação Shellsort iguala-se ao método Quicksort em termos de complexidade temporal, porém é mais eficiente para quantidades pequenas a moderadas de dados.

Comentários:

| 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) |

Não, a complexidade temporal é completamente diferente! A complexidade que nós estudamos em aula se trata do comportamento do algoritmo em termos de custo - ela fornece uma resposta digamos que "genérica". *Por que?* Porque são ignorados constantes, parâmetros, etc - perceba que nós dizemos, por exemplo, que o algoritmo x tem complexidade $O(n^2)$. *O que isso significa?* Significa apenas que esse algoritmo tem um custo polinomial, mas nós sabemos que funções polinomiais têm o formato $ax^2 + bx + c$. Onde está o a, bx, c? Nós ignoramos e ficamos apenas com o parâmetro mais importante. Além disso tudo, ainda temos que considerar que existe o pior caso, melhor caso e caso médio.

Para saber a complexidade temporal de um algoritmo de ordenação, nós temos que nos focar em diversos parâmetros. Além das constantes e parâmetros da função que representa a complexidade, nós temos que levar em consideração aspectos práticos (Por exemplo: o hardware em que será rodado o programa). Resumindo nosso papo: não há correlação direta entre complexidade de custo e complexidade temporal, logo eu não posso usar a função de um para calcular o outro.

Gabarito: E

---

17. (CESPE - 2012 – BASA – Analista de Sistemas) O método de classificação Quicksort é estável e executado em tempo linearmente dependente da quantidade de dados que estão sendo classificados.

Comentários:

| 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 | $\Omega(n \log n)$ | $\Theta(n \log n)$ | $O(n \log n)$ |

Conforme vimos em aula, QuickSort é instável e não possui complexidade temporal linear!

Gabarito: E

---

18. (CESPE - 2012 – BASA – Analista de Sistemas) No método de ordenamento denominado shellsort, as comparações e as trocas são feitas conforme determinada distância entre dois elementos, de modo que, uma distância igual a 6 seria a comparação entre o primeiro elemento e o sétimo, ou entre o segundo elemento e o oitavo, e assim sucessivamente, repetindo-se esse processo até que as últimas comparações e trocas tenham sido efetuadas e a distância tenha diminuído até chegar a 1.

Comentários:

*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.*

Conforme vimos em aula, a questão está perfeita.

Gabarito: C

---

19. (FGV - 2008 – PETROBRÁS – Analista de Sistemas) Sobre o algoritmo de ordenação heapsort, assinale a afirmação correta.

a) Utiliza ordenação por árvore de decisão, ao invés de ordenação por comparação.

b) A estrutura de dados que utiliza, chamada heap, pode ser interpretada como uma árvore binária.

c) Seu desempenho de pior caso é pior do que o do algoritmo quicksort.

d) Seu desempenho de pior caso é o mesmo da ordenação por inserção.

e) Seu desempenho de pior caso é menor do que o da ordenação por intercalação.

Comentários:

(a) Utiliza ordenação por seleção; (b) Perfeito; (c) Não, é melhor que o QuickSort; (d) Não, é melhor que o InsertionSort; (e) Não, é idêntico ao MergeSort.

Gabarito: B

---

20. (CESGRANRIO – 2009 – BASA – Analista de Sistemas) Com relação aos algoritmos quicksort e mergsort, o tempo de execução para o:

a) pior caso do quicksort é (n lg n).
b) pior caso do mergesort é ($n^2$).
c) pior caso do mergesort é (n lg n).
d) caso médio do mergesort é O(lg n).
e) caso médio do quicksort é O($n^2$).

Comentários:

| 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 | $\Omega(n \log n)$ | $\Theta(n \log n)$ | $O(n \log n)$ |

Conforme vimos em aula, trata-se da terceira opção.

Gabarito: C

---

21. (CESPE – 2009 – UNIPAMPA – Analista de Sistemas) O algoritmo quicksort, que divide uma instrução em quatro blocos diferentes de busca, é um exemplo de estrutura de ordenação de dados.

Comentários:

Conforme vimos em aula, são dois blocos diferentes de busca.

Gabarito: E

---

22. (CESPE - 2013 – CPRM – Analista de Sistemas) No algoritmo de ordenação denominado quicksort, escolhe-se um ponto de referência, denominado pivô, e separam-se os elementos em dois grupos: à esquerda, ficam os elementos menores que o pivô, e à direita ficam os maiores. Repete-se esse processo para os grupos de elementos formados (esquerda e direita) até que todos os elementos estejam ordenados.

Comentários:

*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.*

*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!*

Conforme vimos em aula, é exatamente assim!

Gabarito: C

---

23. (CESPE - 2013 – MPU – Analista de Sistemas) Entre os algoritmos de ordenação e pesquisa bubble sort, quicksort e heapsort, o quicksort é considerado o mais eficiente, pois se caracteriza como um algoritmo de dividir-para-conquistar, utilizando operações de particionamento.

Comentários:

Conforme vimos em aula, o HeapSort é o mais eficiente no pior caso!

Gabarito: E

---

24. (CESPE - 2013 – TRT/9 – Analista de Sistemas) No método Quicksort, o pivô é responsável pelo número de partições em que o vetor é dividido. Como o pivô não pode ser um elemento que esteja repetido no vetor, o Quicksort não funciona quando há elementos repetidos.

Comentários:

O pivô não é responsável pelo número de partições em que o vetor é dividido. Ademais, ele pode sim ser um elemento que esteja repetido no vetor!

Gabarito: E

---

25. (FCC - 2011 - TRT - 14ª Região (RO e AC) - Analista Judiciário - Tecnologia da Informação) NÃO se trata de um método de ordenação (algoritmo):

a) inserção direta.
b) seleção direta.
c) inserção por meio de incrementos decrescentes.
d) direta em cadeias.
e) particionamento.

Comentários:

(a) Trata-se do InsertionSort; (b) Trata-se do SelectionSort; (c) Trata-se do ShellSort; (d) Trata-se de um método de busca; (e) Trata-se do QuickSort. Portanto, é a quarta opção!

Gabarito: D

---

26. (FCC - 2016 - TRT - 23ª REGIÃO (MT) - Analista Judiciário - Tecnologia da Informação)

Considere o método de ordenação abaixo.

```
void ordena(int m,int x[]) {
  int aux,j,i;
  for(i=0;i<m-1;i++) {
    for(j=0;j<m-i-1;j++)
      if (x[j] > x[j+1]) {
        aux=x[j];
        x[j]=x[j+1];
        x[j+1]=aux;
      }
  }
}
```

Utilizando este algoritmo de ordenação, percorre-se a lista dada da esquerda para a direita, comparando pares de elementos consecutivos, trocando de

lugar os que estão fora da ordem. Em cada troca, o maior elemento é deslocado uma posição para a direita. Trata-se de um algoritmo de ordenação

a) Select Sort.
b) Insert Sort.
c) Bubble Sort.
d) Shell Sort.
e) Quick Sort.

Comentários:

O algoritmo realiza trocas de dois em dois, percorrendo todos os elementos. Como vimos, esse algoritmo é o Bubble Sort.

Gabarito: C

| ACERTEI | ERREI |
| --- | --- |
| | |

# LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
## LÓGICA DE PROGRAMAÇÃO

1. (FCC - 2010 - DPE-SP - Agente de Defensoria - Analista de Sistemas) É utilizada para avaliar uma determinada expressão e definir se um bloco de código deve ou não ser executado. Essa é a definição da estrutura condicional:

   a) For
   b) If...Then...Else
   c) While
   d) Do...While
   e) Next

2. (FCC - 2010 – TRT/SE - Analista de Sistemas) Objeto que se constitui parcialmente ou é definido em termos de si próprio. Nesse contexto, um tipo especial de procedimento (algoritmo) será utilizado, algumas vezes, para a solução de alguns problemas. Esse procedimento é denominado:

   a) Recursividade.
   b) Rotatividade.
   c) Repetição.
   d) Interligação.
   e) Condicionalidade.

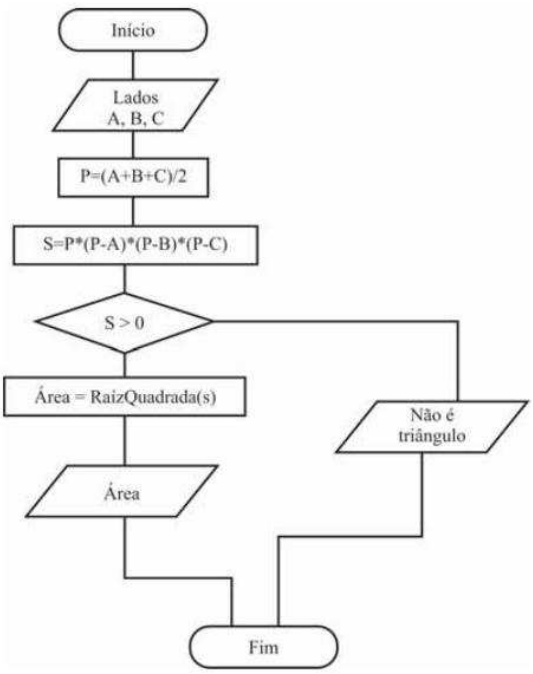
3. (FCC - 2009 – TJ/SE - Analista de Sistemas) A recursividade na programação de computadores envolve a definição de uma função que:

   a) apresenta outra função como resultado.
   b) aponta para um objeto.
   c) aponta para uma variável.
   d) chama uma outra função.
   e) pode chamar a si mesma.

4. (CESPE - 2011 - TJ-ES - Técnico de Informática - Específicos) Uma estrutura de repetição possibilita executar um bloco de comando, repetidas vezes, até que seja encontrada uma dada condição que conclua a repetição.

5. (CESPE - 2010 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Se um trecho de algoritmo tiver de ser executado repetidamente e o número de repetições for indefinido, então é correto o uso, no início desse trecho, da estrutura de repetição Enquanto.

6. (CESPE - 2013 - CNJ - Programador de computador) No fluxograma abaixo, se A = 4, B = 4 e C = 8, o resultado que será computado para Área é igual a 32.




7. (CESPE - 2011 - TJ-ES - Analista Judiciário - Análise de Banco de Dados - Específicos) Em uma estrutura de repetição com variável de controle, ou estrutura PARA, a verificação da condição é realizada antes da execução do corpo da sentença, o que impede a reescrita desse tipo de estrutura por meio de estrutura de repetição pós-testada.

8. (CESPE - 2010 – DETRAN/ES - Analista de Sistemas) O método de recursividade deve ser utilizado para avaliar uma expressão aritmética na qual um procedimento pode chamar a si mesmo, ou seja, a recursividade consiste em um método que, para que possa ser aplicado a uma estrutura, aplica a si mesmo para as subestruturas componentes.

9. (CESPE - 2013 – CPRM - Analista de Sistemas) Na implementação de recursividade, uma das soluções para que se evite o fenômeno de terminação do programa – que possibilita a ocorrência de um looping infinito – é definir uma função ou condição de terminação das repetições.

10. (CESPE - 2014 – ANATEL - Analista de Sistemas) A recursividade é uma técnica que pode ser utilizada na implementação de sistemas de lógica complexa, com a finalidade de minimizar riscos de ocorrência de defeitos no software.

11. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Tanto a recursividade direta quanto a indireta necessitam de uma condição de saída ou de encerramento.

12. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo.

```
Atribuir 13 a X;
repetir
    atribuir X – 2 a X;
    imprimir (X);
Até que X < -1;
```

A estrutura será executada até que X seja igual ao seguinte valor
a) – 1
b) – 3

13. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo, que mostra o emprego da estrutura de controle enquanto ... faça ...

```
atribuir 0 a n;
enquanto n < 7 faça
          início
            imprimir (n);
            atribuir n + 1 a n;
          fim;
```

A opção que utiliza a estrutura para ... faça ... correspondente, que gera o mesmo resultado, é:

c) Para n de 0 até 6 faça imprimir(n);
d) Para n de 0 até 7 faça imprimir(n);

14. (FEPESE - 2010 - SEFAZ-SC - Auditor Fiscal da Receita Estadual - Parte III - Tecnologia da Informação) Assinale a alternativa correta a respeito das variáveis e constantes, utilizadas em diversas linguagens de programação.

a) O número de constantes deve ser menor ou igual ao número de variáveis em um programa.

b) O número de constantes deve ser menor ou igual ao número de procedimentos em um programa.
c) O número de constantes deve ser igual ao número de variáveis em um programa.
d) O número de constantes independe da quantidade de variáveis em um programa.
e) O número de constantes deve ser igual ao número de procedimentos em um programa.

15. (NUCEPE - 2015 – SEDUC/PI - Analista de Sistemas) O código abaixo é usado para calcular o fatorial de números. Assinale a alternativa CORRETA sobre esse código:

```
função fatorial(n)
{
  se (n <= 1)
    retorne 1;
  senão
    retorne n * fatorial(n-1);
}
```

a) Este é um exemplo de procedimento.

b) O comando retorne pode ser retirado do código e a função terá o mesmo efeito.

c) Exemplo clássico de recursividade.

d) Não é possível chamar a função fatorial dentro dela mesma.

e) O resultado da função sempre retornará um valor elevado a ele mesmo (valor ^ valor).

16. (IADES - 2011 – PG/DF - Analista Jurídico - Analista de Sistemas) Os algoritmos são compostos por estruturas de controle de três tipos: sequencial, condicional e de repetição. Assinale a alternativa que apresenta apenas um tipo de estrutura de controle.

a) ...
escreva (“Digite seu nome: ”)
leia (nome)
escreva (“Digite sua idade: ”)

```
    leia (idade)
    limpe a tela
    escreva ("Seu nome é:", nome)
    escreva ("Sua idade é:", idade)
    se (nome = "João") entao
      se (idade > 18) entao
         escreva (nome, " é maior de 18 anos!")
      fim se
   fim se
   ...
```

b) ...

```
    escreva ("Pressione qualquer tecla para começar...")
    leia (tecla)
    mensagem ← "Não devo acordar tarde..."
    numero ← 0
    enquanto (numero < 100)
         escreva (mensagem)
         numero ← (numero + 1)
    fim enquanto
    escreva ("Pressione qualquer tecla para
    terminar...")
    leia (tecla)
    escreva ("Tecla digitada: ")
    escreva (tecla)
    ...
```

c) ...

```
    leia (nome)
    escreva ("nome digitado: ")
    escreva (nome)
    se (nome = "Wally") entao
        escreva ("Encontrado o Wally!")
    senao
        cont ← 5
        enquanto (cont > 0)
          escreva ("Não é Wally"...")
          cont ← (cont – 1)
        fim enquanto
    fim se
```

```
...
```

d) ...

```
var
   nome: literal
   num: inteiro
inicio
   escreva ("Digite seu nome: ")
   leia (nome)
   num ← 0
   se (nome = "José") entao
      num ← (num + 1)
   fim se
  escreva ("Quantidade de João encontrados:
")
  escreva (num)
...
```

e) ...

```
var
    nome: literal
    idade: inteiro
inicio
   escreva ("Digite seu nome: ")
   leia (nome)
   escreva ("Digite sua idade: ")
   leia (idade)
   limpe a tela
   escreva ("Seu nome é:")
   escreva (nome)
   escreva ("Sua idade é:")
   escreva (idade)
fim algoritmo
...
```

17. (IADES - 2011 – TRE-PA - Programador de Computador)

```
VAR
N1, N2 : INTEIRO;
N1 ← 2;
N2 ← 30;
INICIO
ENQUANTO N1<N2 FAÇA
        N2 ← N2 + N1;
        N1 ← N1 * 3;
FIM ENQUANTO;
N1 ← N2 + 11;
FIM
```

Dado o algoritmo escrito em pseudocódigo, quais os valores de N1 e N2, respectivamente, ao final da execução?

a) 162 e 110.
b) 110 e 121.
c) 110 e 162.
d) 121 e 110.
e) 173 e 110.

# LISTA DE EXERCÍCIOS COMENTADOS (CONSULPLAN)
## FUNÇÕES E PROCEDIMENTOS

1. (CONSULPLAN - 2012 - TSE – Técnico – Programação de Sistemas)

Analise o pseudocódigo, que ilustra o uso de uma função recursiva.

```
programa PPRRGG;
variáveis
   VERDE, AZUL : numérica;
função FF(AUX:numérica):numérica;
início
   atribuir VERDE+1 a VERDE;
   se AUX <= 2
   então atribuir 5 a FF
   senão atribuir AUX*FF(AUX-1) a FF;
fim; { fim da função FF }
início
  atribuir 0 a VERDE;
  atribuir FF(4) a AZUL;
  escrever(VERDE,AZUL);
fim.
```

O valor de retorno de FF e a quantidade de vezes que a função será executada serão, respectivamente,
(A) 5 e 1.
(B) 15 e 2.
(C) 60 e 3.
(D) 300 e 4.

# LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
## COMPLEXIDADE DE ALGORITMOS

1. (VUNESP – 2012 – TJ/SP - Analista Judiciário - Tecnologia da Informação) Considerando o conceito de Complexidade de Algoritmos, representado por O(função), assinale a alternativa que apresenta, de forma crescente, as complexidades de algoritmos.

   a) $O(2^n)$; $O(n^3)$; $O(n^2)$; $O(\log_2 n)$; $O(n.\log_2 n)$.
   b) $O(n^2)$; $O(n^3)$; $O(2^n)$; $O(\log_2 n)$; $O(n.\log_2 n)$.
   c) $O(n^3)$; $O(n^2)$; $O(2^n)$; $O(n.\log_2 n)$; $O(\log_2 n)$.
   d) $O(\log_2 n)$; $O(n.\log_2 n)$; $O(n^2)$; $O(n^3)$; $O(2^n)$.
   e) $O(n.\log_2 n)$; $O(\log_2 n)$; $O(2^n)$; $O(n^3)$; $O(n^2)$.

2. (FCC - 2010 - TRT - 8ª Região (PA e AP) - Analista Judiciário - Tecnologia da Informação) Numa competição de programação, ganhava mais pontos o time que apresentasse o algoritmo mais eficiente para resolver o pior caso de um determinado problema. A complexidade assintótica (notação Big O) dos algoritmos elaborados está ilustrada na tabela abaixo.

| Time | Complexidade |
|---|---|
| Branco | $O(n^{20})$ |
| Amarelo | O(nlogn) |
| Azul | O(1) |
| Verde | O(n!) |
| Vermelho | $O(2^n)$ |

   O time que obteve a medalha de prata (2º algoritmo mais eficiente) é o:

   a) Branco.
   b) Amarelo.
   c) Azul.
   d) Verde.
   e) Vermelho.

# LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
## MÉTODOS DE ORDENAÇÃO

1. (Instituto Cidades - 2012 - TCM-GO - Auditor de Controle Externo - Informática) São exemplos de algoritmos de ordenação, exceto:

   a) Bubble Sort
   b) Select Sort
   c) Shell Sort
   d) Busca Sequencial;
   e) Quick Sort;

2. (FUMARC - 2012 - TJ-MG - Técnico Judiciário - Analista de Sistemas – I) Quicksort divide um conjunto de itens em conjuntos menores, que são ordenados de forma independe, e depois os resultados são combinados para produzir a solução de ordenação do conjunto maior.

3. (CESPE - 2012 - MPE-PI - Analista Ministerial - Informática - Cargo 6) O heapsort é um algoritmo de ordenação em que a quantidade de elementos armazenada fora do arranjo de entrada é constante durante toda a sua execução.

4. (CESPE - 2010 - ABIN - Oficial Técnico de Inteligência - Área de Suporte a Rede de Dados) A eficácia do método de ordenação rápida (quicksort) depende da escolha do pivô mais adequado ao conjunto de dados que se deseja ordenar. A situação ótima ocorre quando o pivô escolhido é igual ao valor máximo ou ao valor mínimo do conjunto de dados.

5. (CESPE - 2010 - ABIN - Oficial Técnico de Inteligência - Área de Suporte a Rede de Dados) A estabilidade de um método de ordenação é importante quando o conjunto de dados já está parcialmente ordenado.

6. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Administração de Dados) A classificação interna por inserção é um método que realiza a ordenação de um vetor por meio da inserção de cada elemento em sua posição correta dentro de um subvetor classificado.

7. (FCC - 2009 - TRT - 15ª Região - Analista Judiciário - Tecnologia da Informação) São algoritmos de classificação por trocas apenas os métodos:

   a) SelectionSort e InsertionSort.
   b) MergeSort e BubbleSort.
   c) QuickSort e SelectionSort.
   d) BubbleSort e QuickSort.
   e) InsertionSort e MergeSort.

8. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – I) O tempo de pior caso do algoritmo QuickSort é de ordem menor que o tempo médio do algoritmo Bubblesort.

9. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – II) O tempo médio do QuickSort é $O(n\log_2 n)$, pois ele usa como estrutura básica uma árvore de prioridades.

10. (CESGRANRIO - 2011 - PETROBRÁS – Analista de Sistemas – III) O tempo médio do QuickSort é de ordem igual ao tempo médio do MergeSort.

11. (CESGRANRIO - 2012 - CMB – Analista de Sistemas – III) Em uma reunião de análise de desempenho de um sistema WEB, um programador apontou corretamente que a complexidade de tempo do algoritmo bubblesort, no pior caso, é:

    a) O(1)
    b) O(log n)
    c) O(n)
    d) O(n log n)
    e) $O(n^2)$

12. (CESPE - 2010 – INMETRO – Analista de Sistemas) Se f é uma função de complexidade para um algoritmo F, então O(f) é considerada a complexidade assintótica ou o comportamento assintótico do algoritmo F. Assinale a opção que apresenta somente algoritmos que possuem complexidade assintótica quando f(n) = O(n log n).

    a) HeapSort e BubbleSort
    b) QuickSort e InsertionSort
    c) MergeSort e BubbleSort

d) InsertionSort
e) HeapSort, QuickSort e MergeSort

13. (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

14. (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

15. (CESPE - 2011 – FUB – Analista de Sistemas) Os métodos de ordenação podem ser classificados como estáveis ou não estáveis. O método é estável se preserva a ordem relativa de dois valores idênticos. Alguns métodos eficientes como shellsort ou quicksort não são estáveis, enquanto alguns métodos pouco eficientes, como o método da bolha, são estáveis.

16. (CESPE - 2012 – BASA – Analista de Sistemas) O método de classificação Shellsort iguala-se ao método Quicksort em termos de complexidade temporal, porém é mais eficiente para quantidades pequenas a moderadas de dados.

17. (CESPE - 2012 – BASA – Analista de Sistemas) O método de classificação Quicksort é estável e executado em tempo linearmente dependente da quantidade de dados que estão sendo classificados.

18. (CESPE - 2012 – BASA – Analista de Sistemas) No método de ordenamento denominado shellsort, as comparações e as trocas são feitas conforme determinada distância entre dois elementos, de modo que, uma distância igual a 6 seria a comparação entre o primeiro elemento e o sétimo, ou entre o segundo elemento e o oitavo, e assim sucessivamente, repetindo-se esse processo até que as últimas comparações e trocas tenham sido efetuadas e a distância tenha diminuído até chegar a 1.

19. (FGV - 2008 – PETROBRÁS – Analista de Sistemas) Sobre o algoritmo de ordenação heapsort, assinale a afirmação correta.

a) Utiliza ordenação por árvore de decisão, ao invés de ordenação por comparação.

b) A estrutura de dados que utiliza, chamada heap, pode ser interpretada como uma árvore binária.

c) Seu desempenho de pior caso é pior do que o do algoritmo quicksort.

d) Seu desempenho de pior caso é o mesmo da ordenação por inserção.

e) Seu desempenho de pior caso é menor do que o da ordenação por intercalação.

20. (CESGRANRIO – 2009 – BASA – Analista de Sistemas) Com relação aos algoritmos quicksort e mergsort, o tempo de execução para o:

a) pior caso do quicksort é (n lg n).
b) pior caso do mergesort é ($n^2$).
c) pior caso do mergesort é (n lg n).
d) caso médio do mergesort é O(lg n).
e) caso médio do quicksort é O($n^2$).

21. (CESPE – 2009 – UNIPAMPA – Analista de Sistemas) O algoritmo quicksort, que divide uma instrução em quatro blocos diferentes de busca, é um exemplo de estrutura de ordenação de dados.

22. (CESPE - 2013 – CPRM – Analista de Sistemas) No algoritmo de ordenação denominado quicksort, escolhe-se um ponto de referência, denominado pivô, e separam-se os elementos em dois grupos: à esquerda, ficam os elementos

menores que o pivô, e à direita ficam os maiores. Repete-se esse processo para os grupos de elementos formados (esquerda e direita) até que todos os elementos estejam ordenados.

23. (CESPE - 2013 – MPU – Analista de Sistemas) Entre os algoritmos de ordenação e pesquisa bubble sort, quicksort e heapsort, o quicksort é considerado o mais eficiente, pois se caracteriza como um algoritmo de dividir-para-conquistar, utilizando operações de particionamento.

24. (CESPE - 2013 – TRT/9 – Analista de Sistemas) No método Quicksort, o pivô é responsável pelo número de partições em que o vetor é dividido. Como o pivô não pode ser um elemento que esteja repetido no vetor, o Quicksort não funciona quando há elementos repetidos.

25. (FCC - 2011 - TRT - 14ª Região (RO e AC) - Analista Judiciário - Tecnologia da Informação) NÃO se trata de um método de ordenação (algoritmo):

a) inserção direta.
b) seleção direta.
c) inserção por meio de incrementos decrescentes.
d) direta em cadeias.
e) particionamento.

26. (FCC - 2016 - TRT - 23ª REGIÃO (MT) - Analista Judiciário - Tecnologia da Informação)

Considere o método de ordenação abaixo.

```
void ordena(int m,int x[]) {
  int aux,j,i;
  for(i=0;i<m-1;i++) {
    for(j=0;j<m-i-1;j++)
      if (x[j] > x[j+1]) {
        aux=x[j];
        x[j]=x[j+1];
        x[j+1]=aux;
      }
  }
}
```

Utilizando este algoritmo de ordenação, percorre-se a lista dada da esquerda para a direita, comparando pares de elementos consecutivos, trocando de

lugar os que estão fora da ordem. Em cada troca, o maior elemento é deslocado uma posição para a direita. Trata-se de um algoritmo de ordenação

a) Select Sort.
b) Insert Sort.
c) Bubble Sort.
d) Shell Sort.
e) Quick Sort.

# GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)

## LÓGICA DE PROGRAMAÇÃO

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| B | A | E | C | C | E | E | C | C | E |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| C | B | A | D | C | E | D | | | |

# GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)

## FUNÇÕES DE PROCEDIMENTOS

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| C | | | | | | | | | |

# GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)

## COMPLEXIDADE DE ALGORITMOS

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| D | B | | | | | | | | |

# GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)

## MÉTODOS DE ORDENAÇÃO

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| D | C | C | E | E | C | D | E | E | C |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| E | E | B | A | C | E | E | C | B | C |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| E | C | E | E | D | C | | | | |

# 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.**