PROVAS DE TI
TUDO PARA VOCÊ PASSAR

ITnerante

Algoritmos e Estrutura de dados

Rodrigo Adur
rodrigoadurti@gmail.com

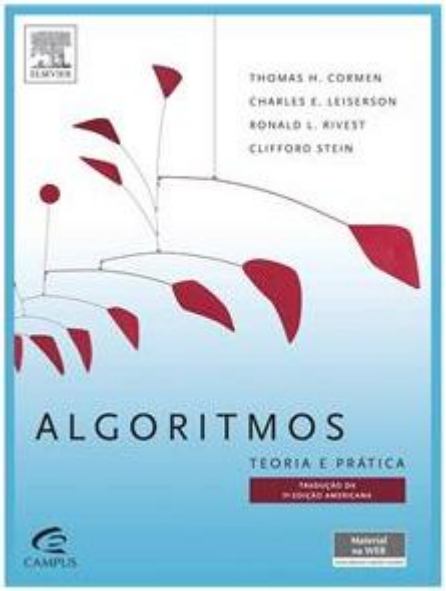
Apresentação

- **Professor Rodrigo Adur**
  - **Formação**
    - Bacharelado em Sistemas de Informações (FILC)
    - Especialista em Sistemas de Informações (NCE/UFRJ)
  - **Experiência Profissional**
    - Desenvolvimento de Sistemas
      - Desenvolvimento de biblioteca de componentes
      - Desenvolvimento de framework
    - Analista de Sistemas do SERPRO
      - Arquiteto
      - Líder técnico
      - Programador

Estrutura do Módulo

- **Módulo I**
  - Conceitos Básicos
  - Complexidade de Algoritmos
  - Arrays
  - Recursividade
  - Pesquisas em Vetores
    - Busca Sequencial
    - Busca Binária
  - Algoritmos de Ordenação
    - Bubble Sort
    - Selection Sort
    - Insertion Sort (Inserção Direta)
    - Heap Sort
    - Merge Sort
    - Quick Sort

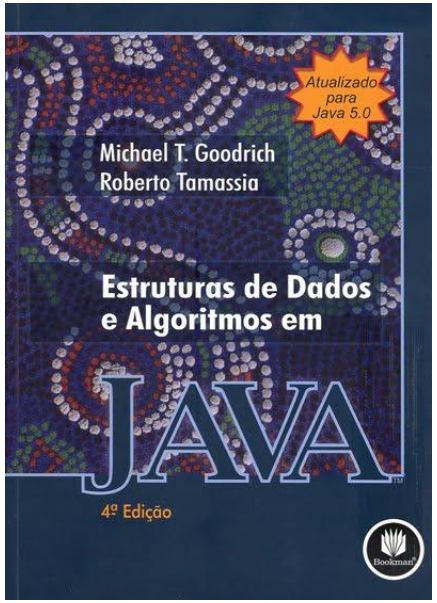
Bibliografia



Algoritmos – Teoria e prática
Thomas Cormen
3ª Edição

Estrutura de Dados e seus Algoritmos
Jayme Luiz
3ª Edição

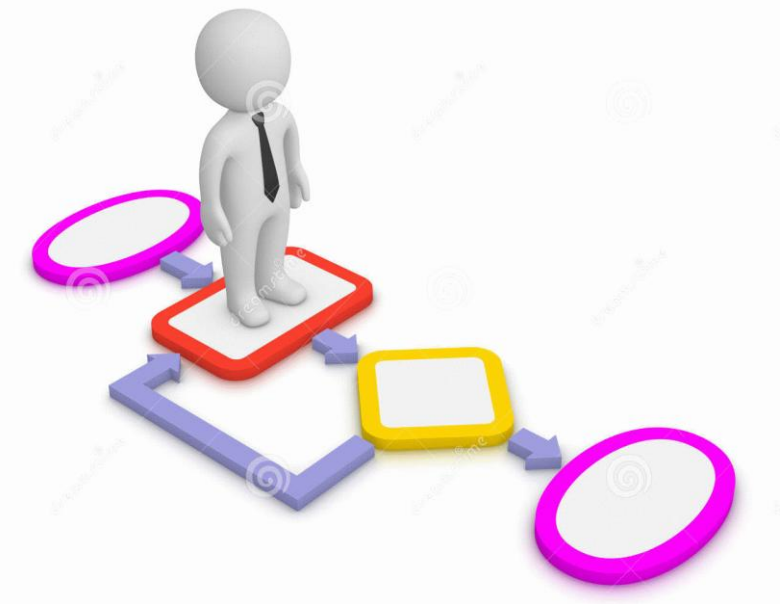
Bibliografia



Estrutura de Dados e Algoritmos em Java

Michael Goodrich

4ª Edição

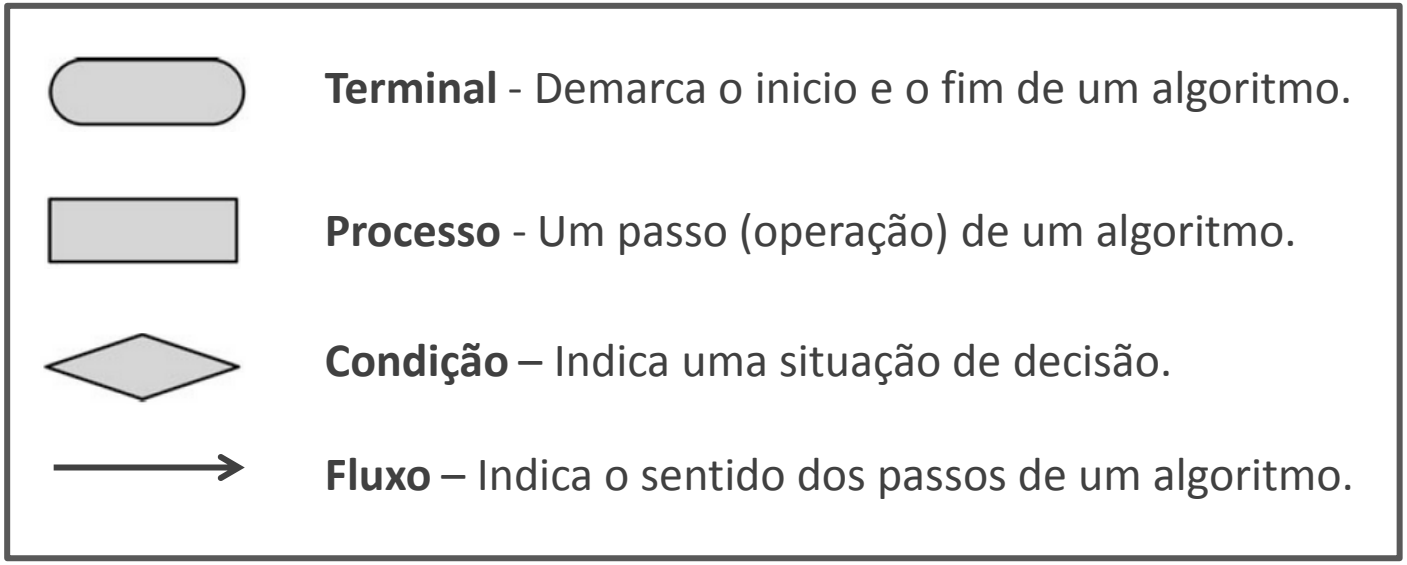


Conceitos Básicos

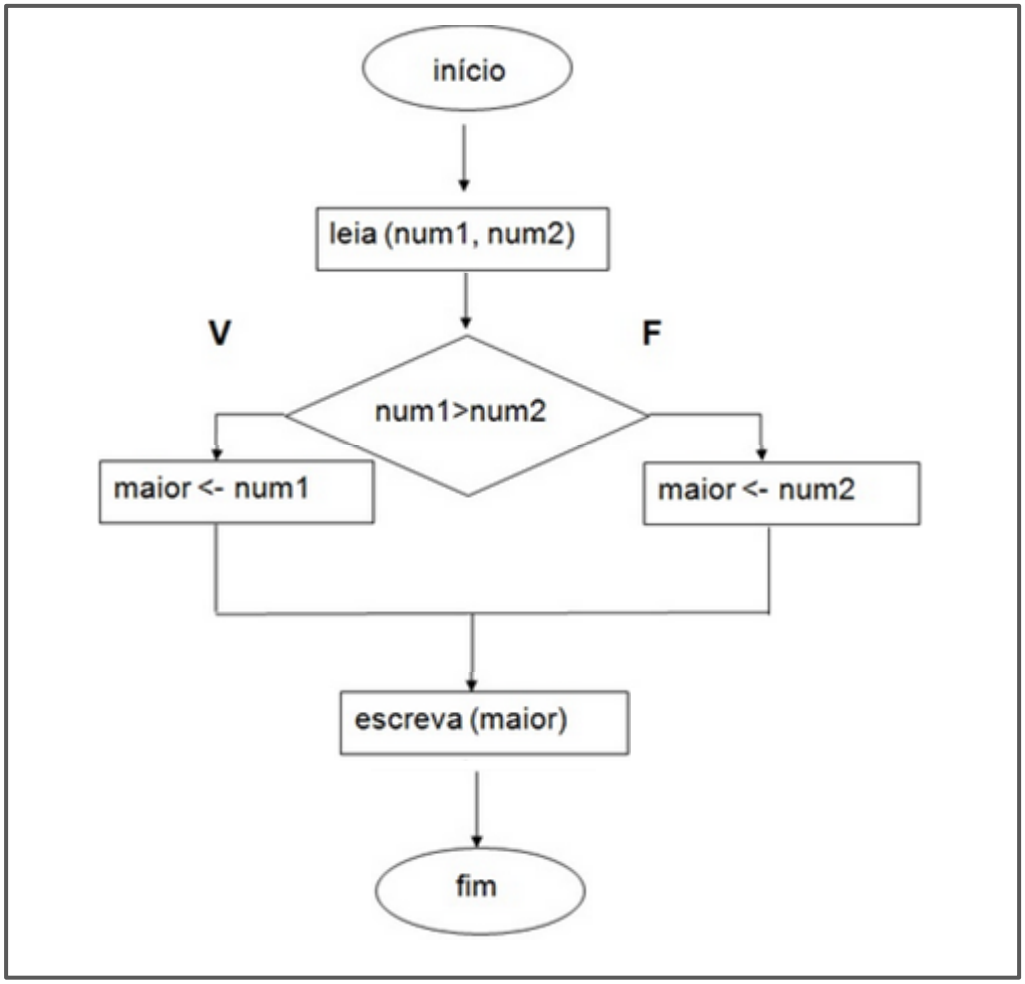
- **Estrutura de Dados**
  - Uma estrutura de dados é um modo particular de armazenamento e organização dos dados que possibilite o uso dos mesmos de forma eficiente.
  - As estruturas de dados diferem umas das outras pela disposição ou manipulação de seus dados. A disposição dos dados em uma estrutura obedece a condição preestabelecida e caracteriza a estrutura.
  - Nenhuma estrutura de dados única funciona bem para todas as finalidades. Dessa forma, é importante conhecer seus pontos fortes e suas limitações.

- **Algoritmos**
  - Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.
  - Um algoritmo é uma sequencia de etapas computacionais que transformam a entrada na saída.

- **Fluxograma**
  - Trata-se de uma representação esquemática de um processo ou uma sequencia de passos, feito através de gráficos que ilustram de forma descomplicada a transição de informações entre os elementos que o compõe.



- Fluxograma


início
leia (num1, num2)
V
F
num1>num2
maior <- num1
maior <- num2
escreva (maior)
fim

- **Pseudocódigo / Pseudolinguagem**
  - O pseudocódigo permite que o programador possa se concentrar na lógica e nas estruturas de controle e não com as regras de uma linguagem específica.
  - Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo. O algoritmo deve ser fácil de se interpretar e fácil de codificar, porém, sem ambiguidades.

Questão 01

[2011 - CESPE - EBC - Analista]

Acerca de pseudocódigo, descrição narrativa e fluxograma, que são usados para a descrição de algoritmos, julgue os próximos itens.

Para especificar os passos de um algoritmo, o pseudocódigo utiliza uma linguagem natural com característica pouco formal, o que pode causar ambiguidade e propiciar interpretações errôneas.

Certo Errado

- **TAD – Tipo Abstrato de Dados**
  - A abstração enfatiza apenas as características essenciais de um objeto, levando em consideração um cenário específico. O conjunto de características resultante da abstração é materializado na forma de um TAD – Tipo Abstrato de Dados.

  - Um tipo abstrato de dados agrupa a estrutura de dados juntamente com as operações que podem ser feitas sobre esses dados.

- **TAD – Tipo Abstrato de Dados**
  - O TAD encapsula a estrutura de dados de forma que os usuários só tenham acesso a algumas operações disponibilizadas sobre esses dados.

Questão 02

ITnerante

[2010 - CESPE - TRT - 21ª Região (RN) - Tecnologia da Informação]

Julgue os itens seguintes, referentes às estruturas de dados.

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

Certo Errado

Questão 03

ITnerante

[2009 - FCC - TRE-PI - Programação de Sistemas]

Em relação a tipos abstratos de dados, é correto afirmar que:

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

(B) Algumas pilhas admitem serem declaradas como tipos abstratos de dados.

(C) Filas não permitem declaração como tipos abstratos de dados.

(D) Os tipos abstratos de dados podem ser formados pela união de tipos de dados primitivos, mas não por outros tipos abstratos de dados.

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

Questão 04

[2010 - CESPE - Banco da Amazônia - Tecnologia da Informação]

Em relação à classificação de dados e tipos abstratos de dados (TADs), julgue os itens subsequentes.

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

Certo Errado

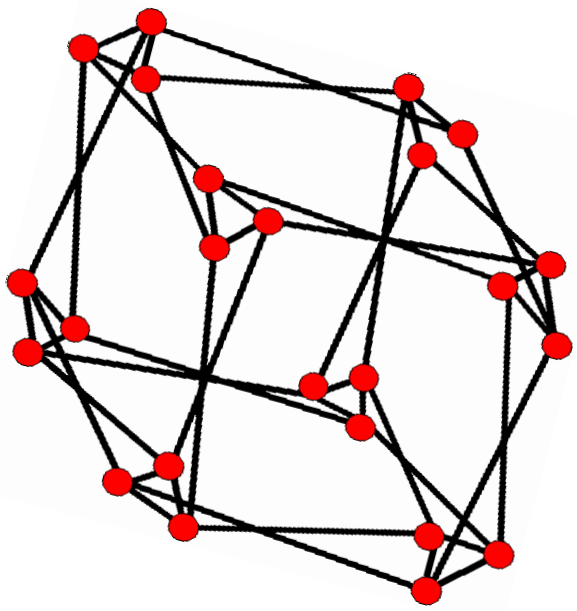
Gabarito

01 – Errado

02 – Certo

03 – E

04 – Errado



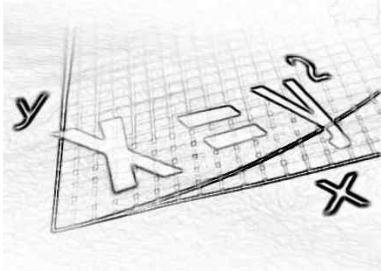
Complexidade de Algoritmos

- **Principais características**
  - Para ser capaz de classificar um algoritmo como sendo bom, são necessárias formas de analisar o mesmo. Analisar um algoritmo significa prever os recursos de que o algoritmo necessita.
  - Em geral, pela análise de vários algoritmos candidatos para um problema, pode-se identificar o mais eficiente.

Complexidade de algoritmos

ITnerante

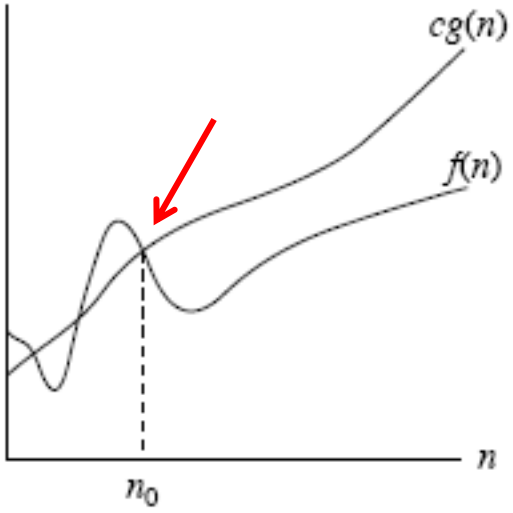
- **Principais características**
  - O tempo de execução de um algoritmo pode ser determinado por:
    - Métodos empíricos - obter o tempo de execução através da execução propriamente dita do algoritmo, considerando-se entradas diversas.
    - Métodos analíticos - O objetivo dos métodos analíticos é determinar uma expressão matemática que traduza o comportamento de tempo de um algoritmo. Esse método visa aferir o tempo de execução de forma independente das condições locais de processamento.



- **Notação Assintótica**
  - Quando observamos tamanhos de entradas suficientemente grandes para tornar relevante apenas a ordem de crescimento do tempo de execução de um algoritmo, estamos estudando a **eficiência assintótica**.
  - Em geral, um algoritmo que é assintoticamente mais eficiente será a melhor escolha para todas as entradas, exceto as muito pequenas.

- **Notação Assintótica**
  - A eficiência assintótica observa apenas as entradas grandes o suficiente para tornar relevante apenas a ordem de crescimento do tempo de execução.
  - Não serão consideradas constantes aditivas ou multiplicativas na expressão matemática obtida.
  - Depois de simplificar a expressão, ficaremos apenas com a parte da função de maior complexidade.
  - Por exemplo:
    - Um valor de número de passos igual a 3n será aproximado para n.
    - Um valor de número de passos igual $n^2 + 2$ será aproximado para n^2.

- **Notação O (ômicron)**
  - A notação O (Big O Notation) é útil para descrever limites superiores de complexidade.
  - **Definição:** Sejam f e g funções positivas. Diz-se que f esta na ordem O de g, escrevendo-se f = O(g), quando existir uma constante c > 0 e um valor n suficientemente grande, tal que:
    $f(n) \leq c.g(n)$
  - A função g atua como um limite superior para valores assintóticos.
  - Exemplo:
    - $5n^2 + 3n\log n + 2$ é $O(n^2)$ — n^2, 1
    - $5n^2 + 3n\log n + 2$ é $O(n^3)$ — n^2, 1

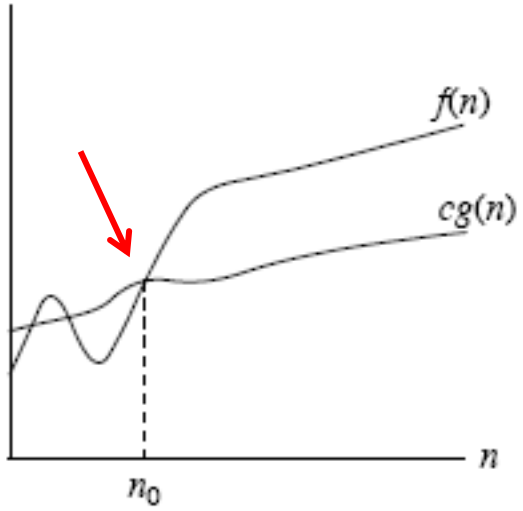


- **Notação θ (theta)**
  - A notação θ é útil para exprimir limites superiores justos.
  - **Definição:** Sejam f e g funções positivas. Diz-se que f é θ(g), escrevendo-se f = θ(g), quando ambas as condições f = O(g) e g = O(f) forem verificadas. A notação θ exprime o fato de que duas funções possuem a mesma ordem de grandeza assintótica.
  - A notação θ permite dizer que duas funções crescem à mesma taxa, até fatores constantes.
  - Exemplo:

$5n^3 + 3n^2 + 2000$ é $\theta(n^3)$

Notação Ω (ômega)

- A notação Ω, descrita a seguir, é útil para descrever limites inferiores assintóticos.
- **Definição:** Sejam f e g funções positivas. Diz-se que f é Ω(g), escrevendo-se f = Ω(g) quando existir uma constante c > 0 e um valor n suficientemente grande, tal que:
  $f(n) \geq c.g(n)$.
- A notação Ω fornece uma maneira assintótica de dizer que uma função cresce a uma taxa que é "maior ou igual" a de uma outra função.
- Exemplo: nlogn 1
  - $3n\log n + 2$ é $\Omega(n^3)$



- **Notação Assintótica**
  - A complexidade de tempo de pior caso corresponde ao número de passos que o algoritmo efetua no seu pior caso de execução, isto é, para a entrada mais desfavorável. A complexidade de pior caso fornece um limite superior para o número de passos que o algoritmo pode efetuar, em qualquer caso.

- **Funções mais usadas na análise de algoritmos**
  - Para saber a complexidade de um algoritmo, divide-se o mesmo em classes de problemas, de acordo com o parâmetro que afeta o algoritmo de forma mais significativa.

Aumento da complexidade (+)

constante	logaritmo	linear	nlogn	quadrática	cúbica	exponencial
1	logn	n	nlogn	n^2	n^3	a^n

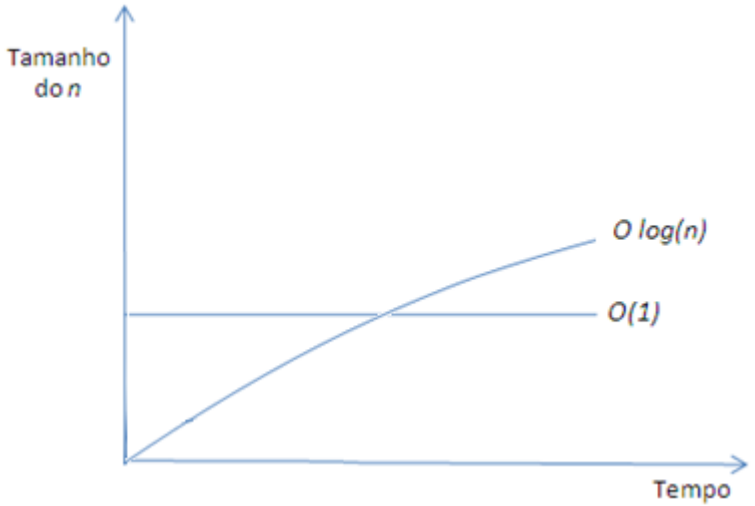
(–) Diminuição da complexidade

- **Funções mais usadas na análise de algoritmos**
  - **A função constante**

    Caracteriza algoritmos de complexidade *1*, independentemente do tamanho n de entradas. É o único caso onde as instruções dos algoritmos são executadas num tamanho fixo de vezes.

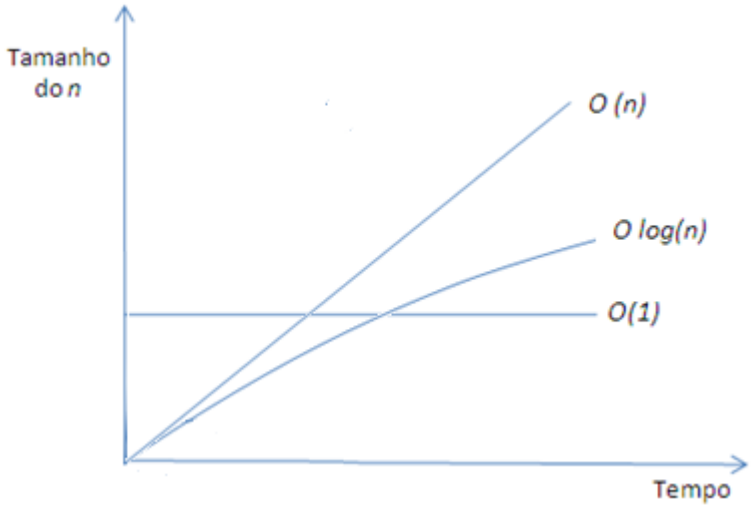
- **Funções mais usadas na análise de algoritmos**
  - **A função logaritmo**

    Caracteriza algoritmos de complexidade *logn*. Ocorre tipicamente em algoritmos que dividem problemas em problemas menores.



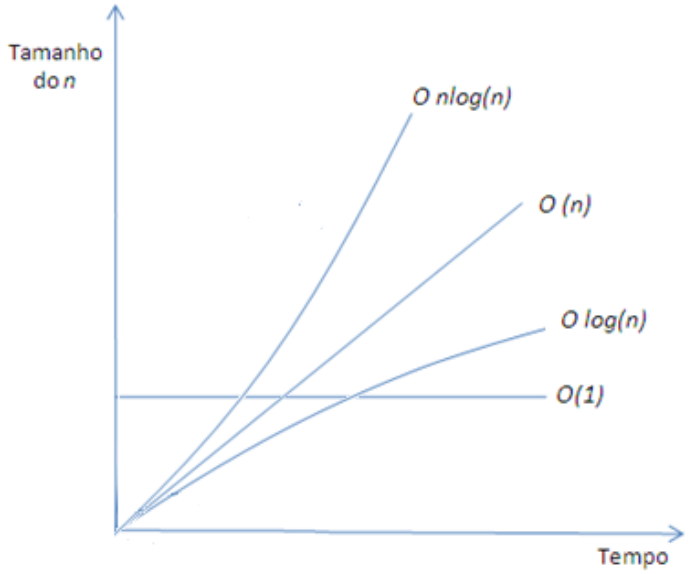
- **Funções mais usadas na análise de algoritmos**
  - **A função linear**

    Caracteriza algoritmos de complexidade *n*. Uma operação básica é executada em cada elemento de entrada do algoritmo.



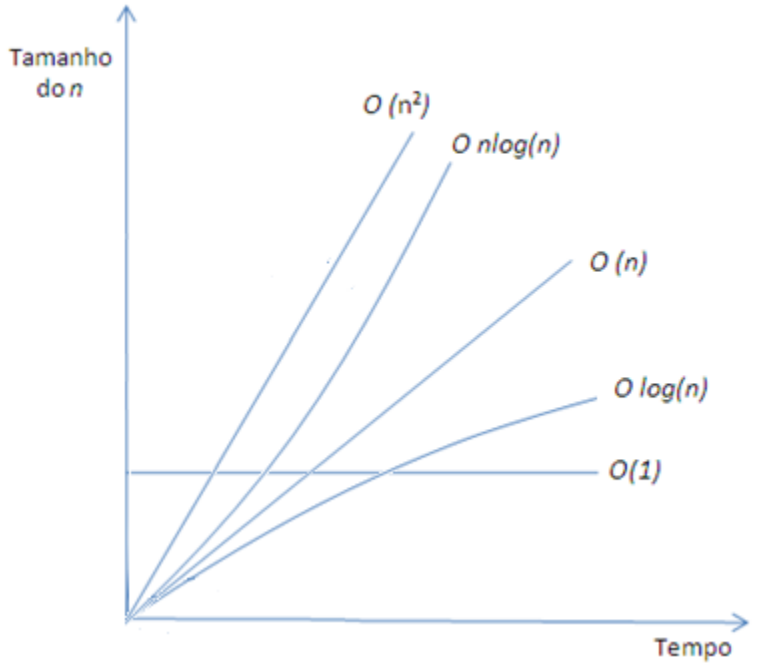
- **Funções mais usadas na análise de algoritmos**
  - **A função nLogn**

    São algoritmos de complexidade *nLogn*. Ocorre tipicamente em algoritmos que dividem problemas em problemas menores, porém juntando posteriormente a solução dos problemas menores.



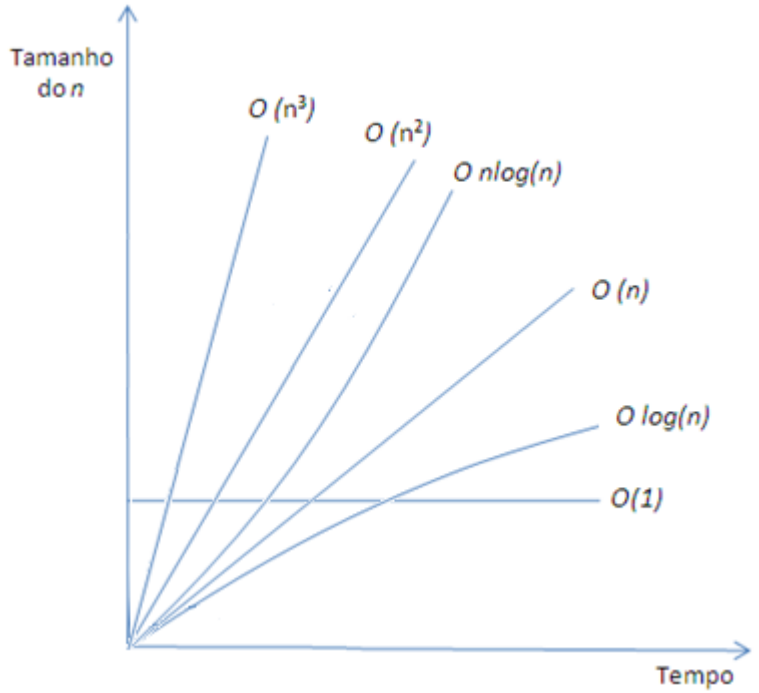
- **Funções mais usadas na análise de algoritmos**
  - **A função quadrática**

    São os algoritmos de complexidade n^2. Nesses algoritmos, os itens são processados aos pares, com laços aninhados.



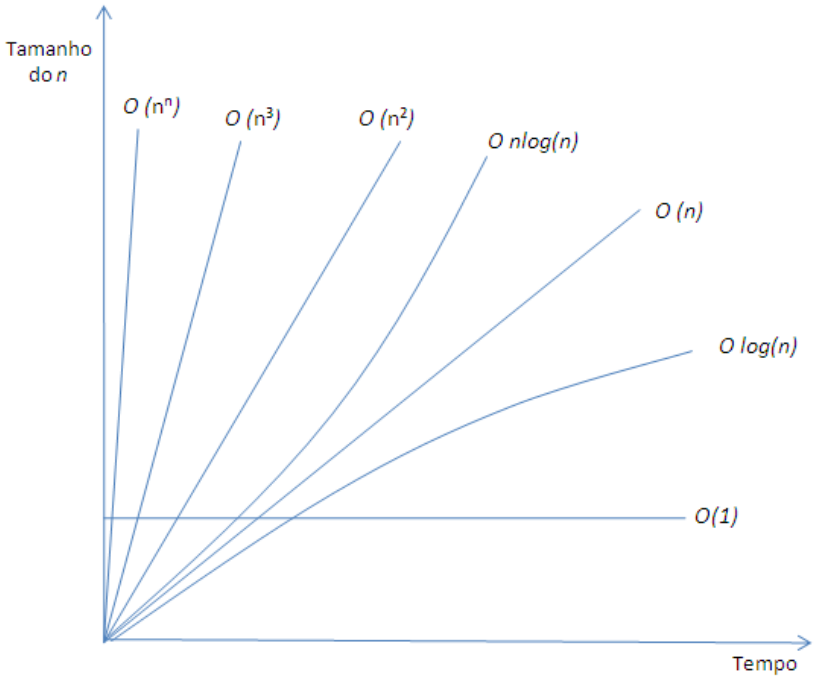
- **Funções mais usadas na análise de algoritmos**
  - **A função cúbica**

    São algoritmos de complexidade n^3. Nesses algoritmos, os Itens são processados três a três, com três laços aninhados.



- **Funções mais usadas na análise de algoritmos**
  - **A função exponencial**

    São algoritmos de complexidade a^n. Por se tratar de algoritmos muito custosos, possuem pouca aplicação prática.



- **Funções mais usadas na análise de algoritmos**

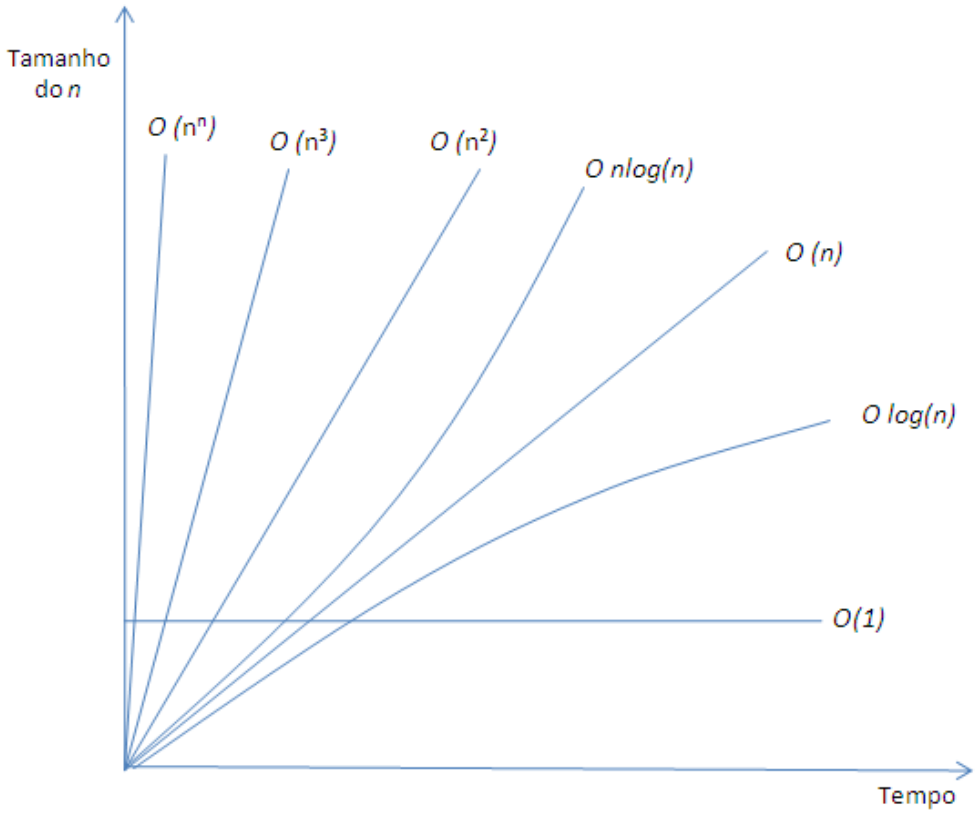


Gráfico comparativo entre as complexidades. Consideramos um algoritmo mais eficiente que outro se o tempo de execução do seu pior caso apresenta uma ordem de crescimento mais baixa. Quanto mais complexo, menos eficiente.

Questão 05

ITnerante

[2009 - CESGRANRIO - BNDES - Análise de Sistemas - Desenvolvimento]

Seja n o tamanho da entrada de um algoritmo para um problema P. Cada alternativa, que corresponde a um algoritmo distinto, apresenta o número de operações necessárias para resolver P. Considerando-se a análise assintótica (Big O notation), qual algoritmo possui menor complexidade?

(A) $2 + 10\log n$
(B) $3n^2 + n$
(C) $1000 + 2n^3$
(D) $5n + 128$
(E) 4^n

Questão 06

ITnerante

[2011 - FCC - TRT - 19ª Região (AL) - Tecnologia da Informação]

Considere os seguintes algoritmos e suas complexidades na notação Big O:

- Algoritmo A: O(log n)
- Algoritmo B: $O(n^2)$
- Algoritmo C: O(n . log n)

Considerando-se o pior caso de execução destes algoritmos, é correto afirmar que o algoritmo:

(A) A é o menos eficiente.
(B) C é o menos eficiente.
(C) A não é o mais eficiente nem o menos eficiente.
(D) B é o menos eficiente.
(E) C é o mais eficiente.

Questão 07

ITnerante

[2009 - CESGRANRIO - Casa da Moeda - Desenvolvimento de Sistemas]

No desenvolvimento de um sistema de análise financeira, um programador utilizou um algoritmo cuja complexidade de tempo, no pior caso, é igual a O(n).
Outro programador aponta um algoritmo de melhor complexidade igual a

(A) O(log n).
(B) O(n log n).
(C) $O(n^2)$
(D) $O(2^n)$
(E) O(n!)

Questão 08

ITnerante

[2008 - CESGRANRIO - BNDES - Análise de Sistemas - Desenvolvimento]

Observe o algoritmo em JAVA.

```
public void algoritmo(int[] v) {
        int m;
        int tmp;
        for (int i=0; i<v.length; i++) {
            m = i;
            for (int j=i+1; j<v.length; j++) {
               if (v[j]<v[m]) {
                  m=j;
               }
            }
            if (m!=i) {
              tmp = v[m];
              v[m] = v[i];
              v[i] = tmp;
            }
        }
}
```

A complexidade de tempo desse algoritmo, no pior caso, em que n corresponde ao número de elementos do vetor v, é:

(A) Θ(n).
(B) O (n log n).
(C) O (n^2).
(D) Θ(n log n).
(E) Ω(n2 log n).

Questão 09

[2008 - CESGRANRIO - BNDES - Análise de Sistemas - Desenvolvimento]

Se a complexidade de tempo de um algoritmo é da ordem de (n log n), é correto afirmar que esse algoritmo também é

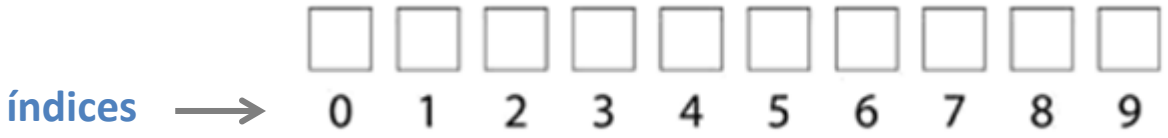
(A) Θ (n).
(B) $\Omega(n^2)$.
(C) Ω(n log n).
(D) O (log n).
(E) O (n).

Gabarito

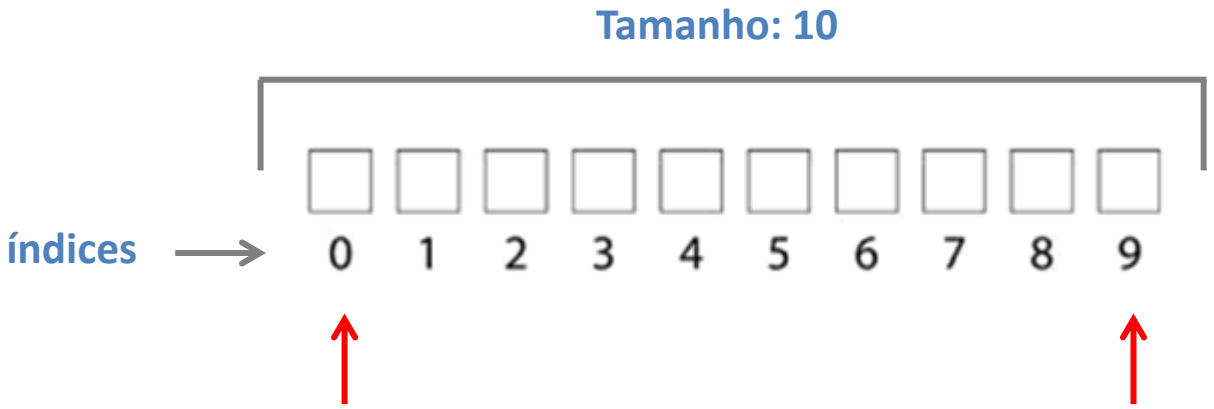
05 – A
06 – D
07 – A
08 – C
09 – C

Arrays

- **Principais características**
  - Um array é uma estrutura de dados que armazena um conjunto de elementos de forma que os mesmos são acessados por um índice.
  - Os arrays podem ser unidimensionais (vetores) ou multidimensionais (matrizes).
  - Os arrays são agregados homogêneos, ou seja, todos os seus elementos são de um mesmo tipo.

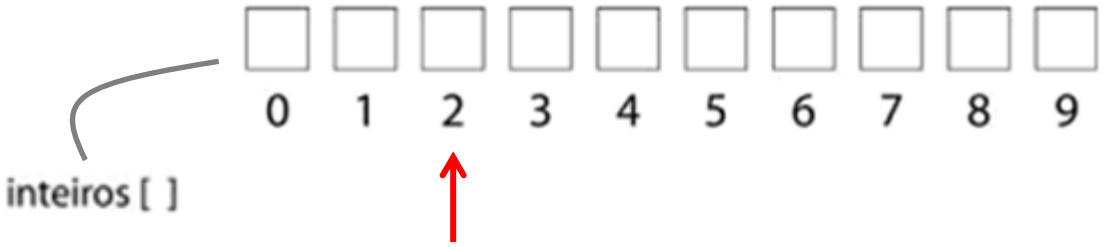


- **Principais características**
  - Os vetores possuem tamanho fixo e suas posições são contíguas na memória.
  - O índice inicial de um vetor é zero. Dessa forma, ao se realizar uma varredura em um array, sempre devemos considerar o intervalo de 0 ao tamanho do vetor - 1;



ITnerante

- **Manipulando vetores**



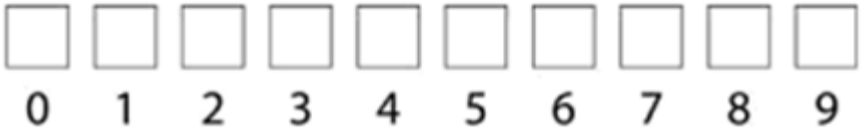
- Definindo vetores:

```
int[] inteiros = new int[10];
```

- Acessando elementos em vetores:

```
int terceiroElemento = inteiros[2];
```

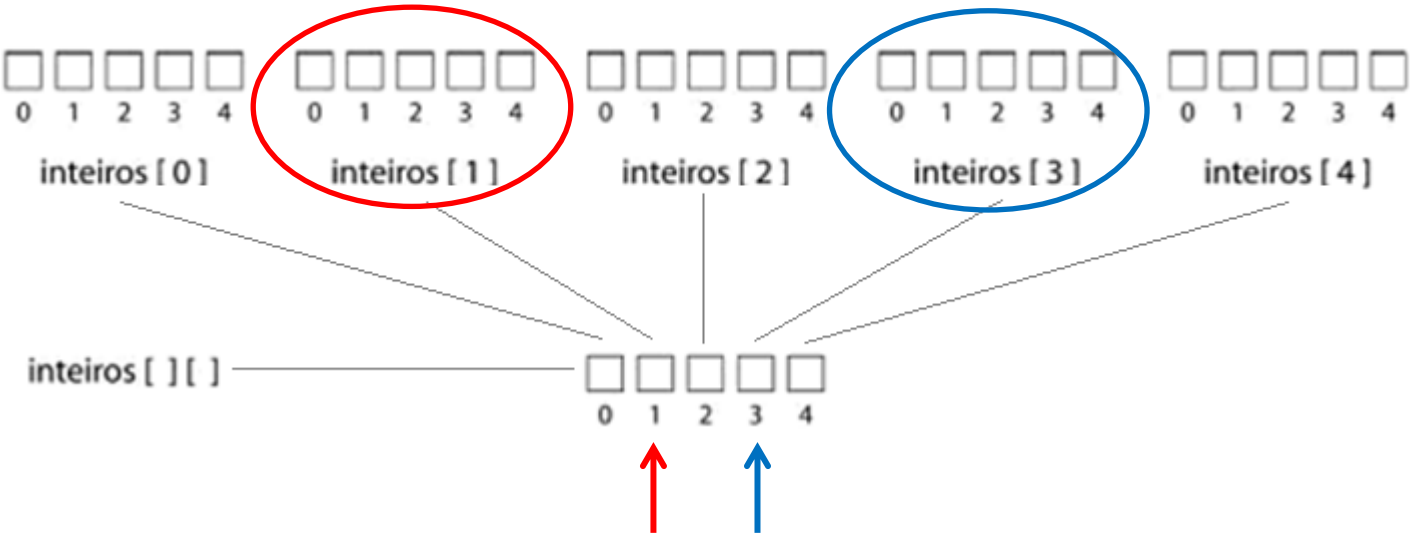
- **Manipulando vetores**



- Percorrendo todos os elementos de um vetor

```
for (int indice = 0; indice < inteiros.length; indice++) {
    int elemento = inteiros[indice];
    // ...
}
```

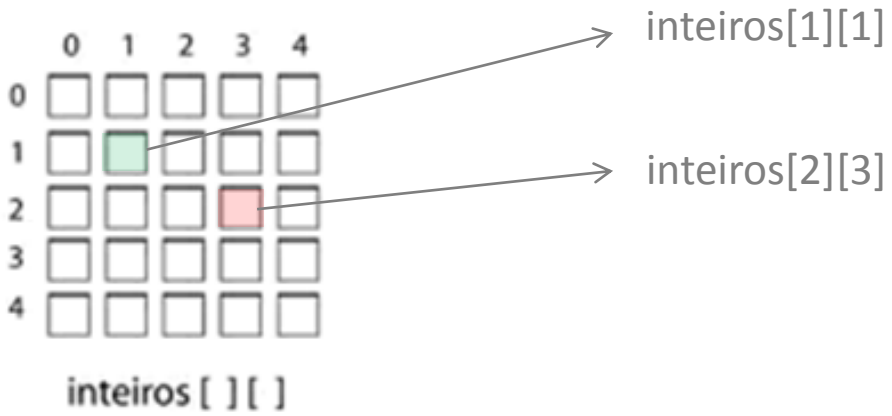
- **Manipulando matrizes**



- Definindo matrizes em java:

```
int inteiros[][] = new int[5][5];
```

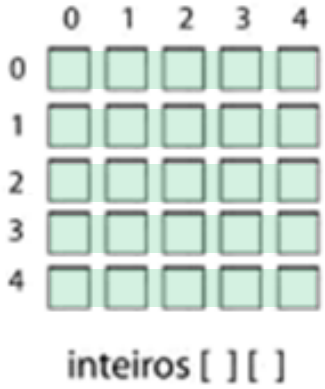
- **Manipulando matrizes**



- Acessando elementos em matrizes:

```
int elemento = inteiros[2][3];
```

- **Manipulando matrizes**



- Percorrendo todos os elementos de uma matriz

```
for (int linha = 0; linha < 5; linha++) {
    for (int coluna = 0; coluna < 5; coluna++) {
        int elemento = inteiros[linha][coluna];
        // ...
    }
}
```

Questão 10

ITnerante

[2009 - CESPE - ANAC - Tecnologia da Informação]

Um array é um agregado, possivelmente heterogêneo, de elementos de dados. Nele, um elemento individual é identificado por sua posição em relação ao primeiro.

Certo Errado

Questão 11

ITnerante

[2011 - CESPE – EBC - Analista]

Acerca das estruturas de dados estáticas e homogêneas (vetores e matrizes) e das estruturas de dados dinâmicas (listas, pilhas, filas), utilizadas para armazenar conjuntos de valores, julgue os itens a seguir.

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

Certo Errado

Questão 12

ITnerante

[2010 - CESPE - TRE-BA - Análise de Sistemas]

Acerca de estruturas de dados do tipo vetor em linguagens estruturadas, julgue os itens a seguir.

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

Certo Errado

Questão 13

ITnerante

[2010 - FCC - TRT - 9ª REGIÃO (PR) - Analista Tecnologia da Informação]

É uma estrutura de dados dividida em linhas e colunas. Desta forma, pode-se armazenar diversos valores dentro dela. Para obter um valor é necessário identificá-lo por meio do número da linha e da coluna onde está armazenado. Trata-se de

(A) árvore.
(B) matriz.
(C) pilha.
(D) fita.
(E) deque.

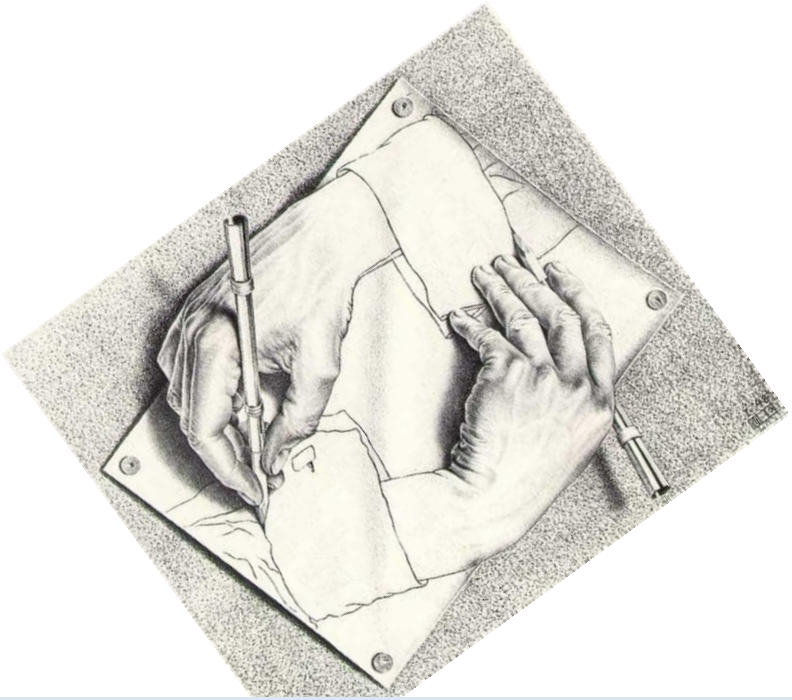
Gabarito

10 – Errado

11 – Errado

12 – Certo

13 - B



Recursividade

- **Principais características**
  - Um método recursivo é aquele que possui uma ou mais chamadas para si mesmo.
  - Todo método recursivo pode ser implementado de forma não recursiva, tal que execute exatamente a mesma computação.
  - Frequentemente, os procedimentos recursivos são mais concisos do que um não recursivo correspondente.
  - Todo método recursivo deve possuir um ou mais casos recursivos e pelo menos um caso base.

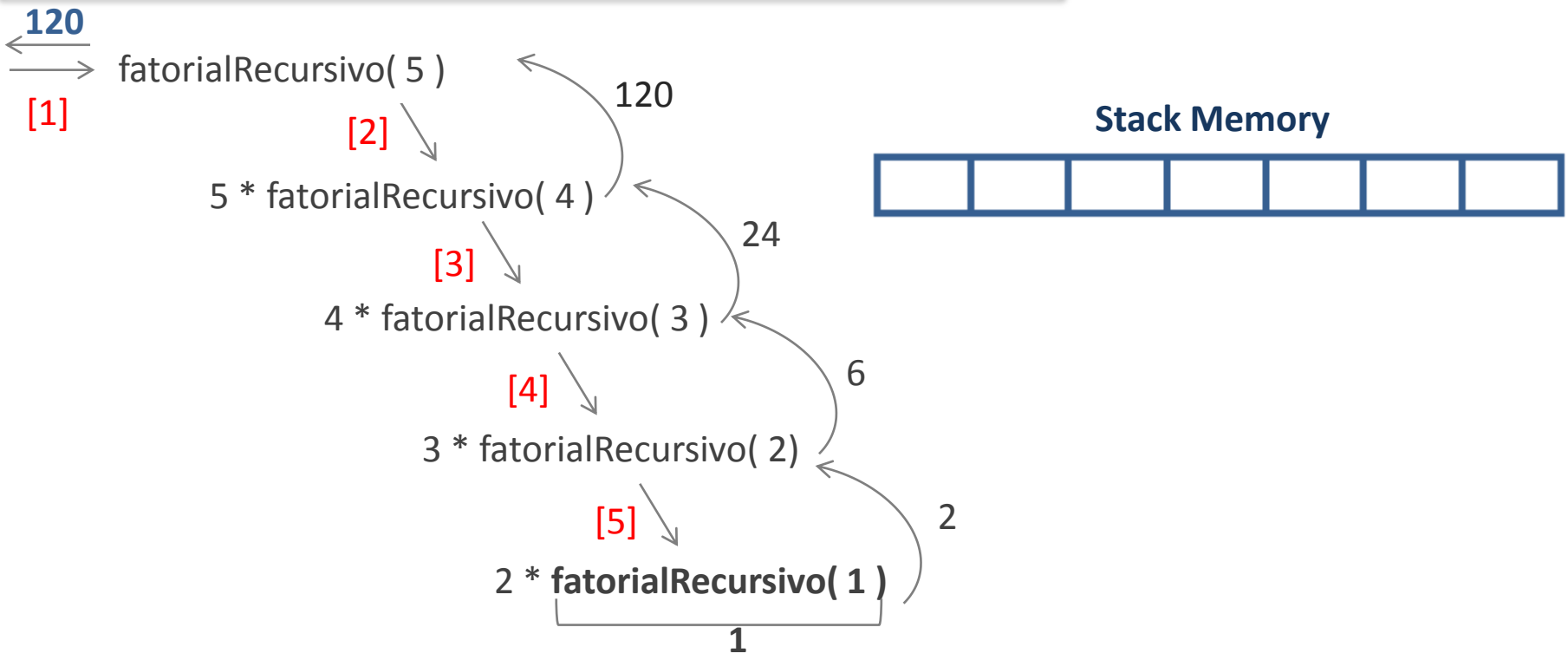
- **Principais características**
  - Um algoritmo não recursivo equivalente a um recursivo pode ser mais eficiente.
  - A depuração de métodos recursivos pode ser mais complexa.
  - Durante o processamento do método recursivo, os dados vão para uma área da memória chamada stack (pilha).

- **Exemplo de recursão**
  - O fatorial de um inteiro positivo *n* é definido como sendo o produto dos inteiros de *n* até *1*.
  - Por convenção, o fatorial de *0* é *1* e o fatorial de *1* é *1*.
  - De maneira formal, para n>1 temos: n! = n x (n-1) x (n-2) ... x 2 x 1
  - Implementação do fatorial de forma recursiva:

```
1    public int fatorialRecursivo(int numero) {
2        // caso base
3        if (numero == 0 || numero == 1) {
4            return 1;
5        } else {
6            // caso recursivo
7            return numero * fatorialRecursivo(numero - 1);
8        }
9    }
```

Recursividade

```
1    public int fatorialRecursivo(int numero) {
2        // caso base
3        if (numero == 0 || numero == 1) {
4            return 1;
5        } else {
6            // caso recursivo
7            return numero * fatorialRecursivo(numero - 1);
8        }
9    }
```



- Implementação do fatorial de forma não recursiva

```
1      public int fatorialNaoRecursivo(int numero) {
2          int valorFactorial = 1;
3          for (int i=numero; i>0; i--) {
4              valorFactorial = valorFactorial * i;
5          }
6          return valorFactorial;
7      }
```

Questão 14

ITnerante

[2009- FCC - TJ – Programação 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.

Questão 15

ITnerante

[2010 - CESPE - Banco da Amazônia - Tecnologia da Informação]

```
1  função func(var M[], A, B: inteiro): inteiro
2  início
3     se (A = B) então
4        retorne M[A]
5     senão
6        retorne M[A] + func(M,A+1,B)
7     fimse
8  fim
```

Considere o algoritmo acima, em que M[ ] seja um vetor de valores inteiros e A e B sejam dois valores inteiros e o primeiro elemento do vetor M esteja localizado na posição 1, sendo os argumentos passados por referência. Com base nessas informações, julgue os itens a seguir.

É possível implementar uma função que gere o mesmo resultado, utilizando a mesma lista de parâmetros e substituindo o uso de recursividade por uma estrutura de repetição do tipo para.

Certo Errado

Questão 16

ITnerante

[2010 - CESPE - Banco da Amazônia - Tecnologia da Informação]

```
1  função func(var M[], A, B: inteiro): inteiro
2  início
3     se (A = B) então
4        retorne M[A]
5     senão
6        retorne M[A] + func(M,A+1,B)
7     fimse
8  fim
```

Considere o algoritmo acima, em que M[ ] seja um vetor de valores inteiros e A e B sejam dois valores inteiros e o primeiro elemento do vetor M esteja localizado na posição 1, sendo os argumentos passados por referência. Com base nessas informações, julgue os itens a seguir.

Se X for um vetor com os elementos [1, 2, 4, 8, 16, 32, 64, 128], a chamada da função func(X,2,6) retornará o valor 62.

Certo Errado

Gabarito

14 – E

15 – Certo

16 – Certo

Pesquisa em Vetores

Pesquisa Sequencial

- **Principais características**
  - Método de busca mais simples. Percorre o vetor sequencialmente, desde o seu primeiro elemento, até que o valor seja encontrado ou que os elementos do vetor se esgotem.
  - A ideia básica do algoritmo seria folhear, por exemplo, uma lista telefônica página a página, até encontrar o nome desejado ou constatar que o mesmo não existe.
  - Algoritmo possui complexidade no pior caso O(n).

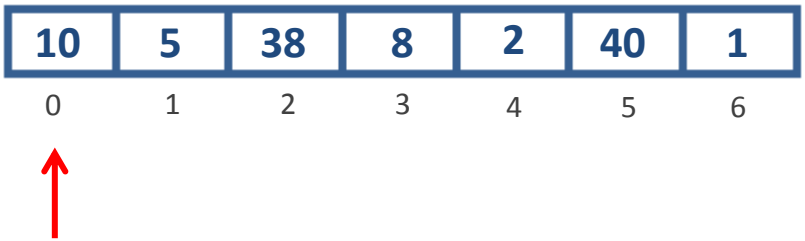
Pesquisa Sequencial

ITnerante

- **Implementação**

```
1    public static int buscaLinear(int[] vetor, int valorProcurado) {
2        for (int i = 0; i < vetor.length; i++) {
3            if (vetor[i]==valorProcurado) {
4                return i;
5            }
6        }
7        return -1;
8    }
```

- Como buscar o número 40 nesse vetor?

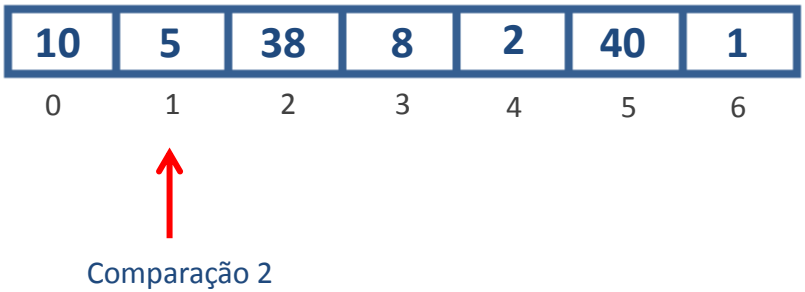


Pesquisa Sequencial

- **Implementação**

```
1     public static int buscaLinear(int[] vetor, int valorProcurado) {
2         for (int i = 0; i < vetor.length; i++) {
3             if (vetor[i]==valorProcurado) {
4                 return i;
5             }
6         }
7         return -1;
8     }
```

- Como buscar o número 40 nesse vetor?



Pesquisa Sequencial

ITnerante

- **Implementação**

```
1     public static int buscaLinear(int[] vetor, int valorProcurado) {
2         for (int i = 0; i < vetor.length; i++) {
3             if (vetor[i]==valorProcurado) {
4                 return i;
5             }
6         }
7         return -1;
8     }
```

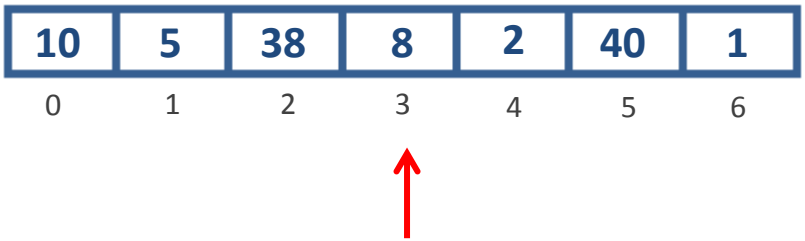
- Como buscar o número 40 nesse vetor?

Pesquisa Sequencial

- **Implementação**

```
1     public static int buscaLinear(int[] vetor, int valorProcurado) {
2         for (int i = 0; i < vetor.length; i++) {
3             if (vetor[i]==valorProcurado) {
4                 return i;
5             }
6         }
7         return -1;
8     }
```

- Como buscar o número 40 nesse vetor?

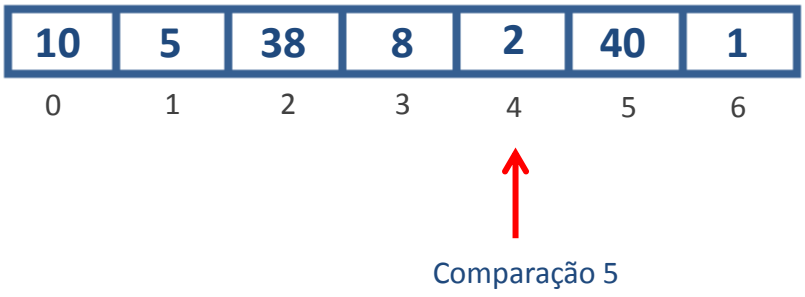


Pesquisa Sequencial

- **Implementação**

```
1     public static int buscaLinear(int[] vetor, int valorProcurado) {
2         for (int i = 0; i < vetor.length; i++) {
3             if (vetor[i]==valorProcurado) {
4                 return i;
5             }
6         }
7         return -1;
8     }
```

- Como buscar o número 40 nesse vetor?



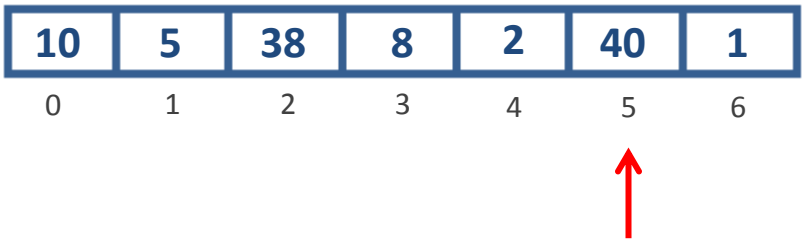
Pesquisa Sequencial

ITnerante

- **Implementação**

```
1     public static int buscaLinear(int[] vetor, int valorProcurado) {
2         for (int i = 0; i < vetor.length; i++) {
3             if (vetor[i]==valorProcurado) {
4                 return i;
5             }
6         }
7         return -1;
8     }
```

- Como buscar o número 40 nesse vetor?



Pesquisa Binária

- **Principais características**
  - A ideia básica do algoritmo é percorrer o vetor como se folheia, por exemplo, uma lista telefônica. Abandonando-se as partes do catálogo onde o nome procurado, com certeza, não será encontrado.
  - Para a realização desse tipo de busca, o vetor deve estar ordenado.
  - Esse método exige acesso aleatório aos elementos do conjunto.
  - Algoritmo possui complexidade no pior caso O(logn).
  - O pior caso ocorre quando o elemento procurado é o último a ser verificado, ou mesmo não é encontrado.

Pesquisa Binária

- **Funcionamento do Algoritmo**
  - O vetor é dividido ao meio.
  - É verificado se o valor procurado é igual ao valor que corresponde à linha de divisão.
    - Caso elemento encontrado, fim da busca.
  - Caso valor não seja o procurado, é verificado se esta acima ou abaixo da linha de divisão.
    - Caso esteja abaixo, a parte superior é descartada, caso contrário, a parte inferior é descartada.
  - Isso é feito sistematicamente até que o valor seja encontrado ou até que seja identificado que o valor não esta na lista. A cada comparação metade dos elementos da lista são descartados.

Pesquisa Binária

ITnerante

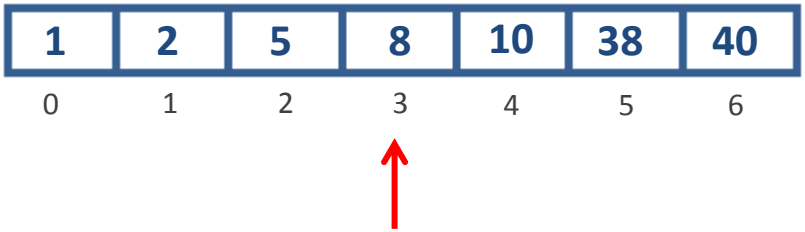
- Implementação iterativa

```
1     public int buscaBinaria( int[] array, int valorProcurado ) {
2         int indiceEsq = 0;
3         int indiceDir = array.length - 1;
4         int indiceMeio;
5
6         while ( indiceEsq <= indiceDir ) {
7                 indiceMeio = (indiceEsq + indiceDir) / 2;
8                 if ( array[indiceMeio] < valorProcurado ) {
9                         indiceEsq = indiceMeio + 1;
10                } else if( array[indiceMeio] > valorProcurado ) {
11                        indiceDir = indiceMeio - 1;
12                } else {
13                        return indiceMeio;
14                }
15        }
16        return -1;
17 }
```

Pesquisa Binária

```
1     public int buscaBinaria( int[] array, int valorProcurado ) {
2         int indiceEsq = 0;
3         int indiceDir = array.length - 1;
4         int indiceMeio;
5
6         while ( indiceEsq <= indiceDir ) {
7                 indiceMeio = (indiceEsq + indiceDir) / 2;
8                 if ( array[indiceMeio] < valorProcurado ) {
9                         indiceEsq = indiceMeio + 1;
10                } else if( array[indiceMeio] > valorProcurado ) {
11                        indiceDir = indiceMeio - 1;
12                } else {
13                        return indiceMeio;
14                }
15        }
16        return -1;
17 }
```

- Como buscar o número 40 nesse vetor?



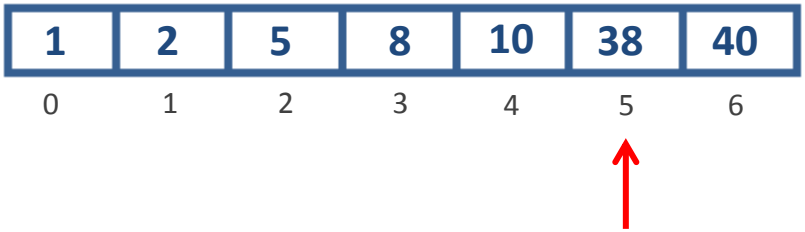
indiceEsq = 0

indiceDir = 6

Pesquisa Binária

```
1   public int buscaBinaria( int[] array, int valorProcurado ) {
2       int indiceEsq = 0;
3       int indiceDir = array.length - 1;
4       int indiceMeio;
5
6       while ( indiceEsq <= indiceDir ) {
7               indiceMeio = (indiceEsq + indiceDir) / 2;
8               if ( array[indiceMeio] < valorProcurado ) {
9                       indiceEsq = indiceMeio + 1;
10              } else if( array[indiceMeio] > valorProcurado ) {
11                      indiceDir = indiceMeio - 1;
12              } else {
13                      return indiceMeio;
14              }
15      }
16      return -1;
17  }
```

- Como buscar o número 40 nesse vetor?



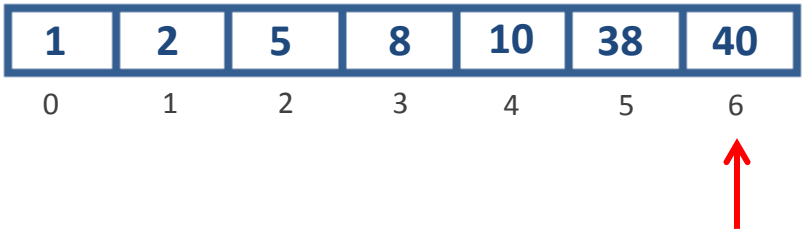
indiceEsq = 0 → 4

indiceDir = 6

Pesquisa Binária

```
1     public int buscaBinaria( int[] array, int valorProcurado ) {
2         int indiceEsq = 0;
3         int indiceDir = array.length - 1;
4         int indiceMeio;
5
6         while ( indiceEsq <= indiceDir ) {
7                 indiceMeio = (indiceEsq + indiceDir) / 2;
8                 if ( array[indiceMeio] < valorProcurado ) {
9                         indiceEsq = indiceMeio + 1;
10                } else if( array[indiceMeio] > valorProcurado ) {
11                        indiceDir = indiceMeio - 1;
12                } else {
13                        return indiceMeio;
14                }
15        }
16        return -1;
17    }
```

- Como buscar o número 40 nesse vetor?



indiceEsq = 0 → 4 → 6

indiceDir = 6

Pesquisa Binária

ITnerante

- ## Implementação recursiva

```
1  private int buscaBinariaRecursiva(int[] array, int indiceEsq, int indiceDir, int valorProcurado) {
2      int indiceMeio = (indiceDir + indiceEsq) / 2;
3
4      if (indiceEsq > indiceDir)
5          return -1;
6
7      else if (array[indiceMeio] == valorProcurado)
8          return indiceMeio;
9
10     else if (array[indiceMeio] < valorProcurado)
11         return buscaBinariaRecursiva(array, indiceMeio + 1, indiceDir, valorProcurado);
12
13     else
14         return buscaBinariaRecursiva(array, indiceEsq, indiceMeio - 1, valorProcurado);
15 }
```

Questão 17

ITnerante

[2010 - CESPE - Banco da Amazônia - Tecnologia da Informação]

Acerca de pesquisa de dados e de operações básicas sobre estruturas, julgue os itens que se seguem.

A pesquisa sequencial é aplicável em estruturas não ordenadas.
Certo Errado

Questão 18

ITnerante

[CESPE - 2010 - Banco da Amazônia - Tecnologia da Informação]

Acerca de pesquisa de dados e de operações básicas sobre estruturas, julgue os itens que se seguem.

Quando um algoritmo recursivo recebe como parâmetro o trecho do vetor no qual deve ser realizada a pesquisa, então essa pesquisa é do tipo sequencial.
Certo Errado

Questão 19

ITnerante

[FCC - 2012 - TJ-RJ - Análise de Sistemas]

O algoritmo conhecido como busca binária é um algoritmo de desempenho ótimo para encontrar a posição de um item em:
(A) uma árvore B.
(B) uma lista ligada ordenada.
(C) uma árvore de busca binária.
(D) um heap binário.
(E) um vetor ordenado.

Questão 20

ITnerante

[CESPE - 2012 - Banco da Amazônia - Administração de Dados]

Acerca de pesquisa de dados e de operações básicas sobre estruturas, julgue os itens que se seguem.

A busca binária é realizada em um grupo de dados previamente ordenado.
Certo Errado

Questão 21

ITnerante

[CESPE - 2010 - Banco da Amazônia - Tecnologia da Informação]

Acerca de pesquisa de dados e de operações básicas sobre estruturas, julgue os itens que se seguem.

Na pesquisa binária, realiza-se a varredura de uma estrutura de dados desde o seu início até o final dessa estrutura, ou até que uma informação desejada seja encontrada.

Certo Errado

Questão 22

ITnerante

[2012 - CESPE - TRE-RJ - Programação de Sistemas]

Julgue os itens a seguir, referentes a estrutura de dados e organização de arquivos.

Uma das formas mais simples e rápida de busca em uma estrutura de dados ordenada é o método de pesquisa binária, que segue o paradigma de divisão e conquista. Se o item pesquisado estiver no meio do vetor, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior e, se vier depois, a busca continua na metade anterior do vetor.

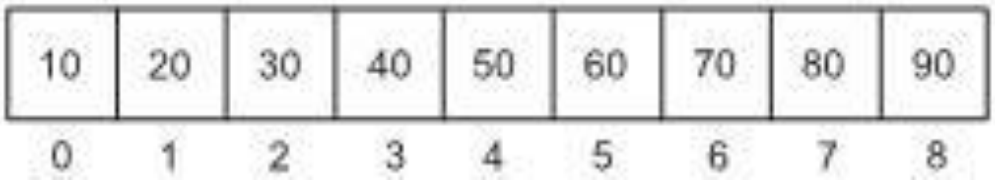
Certo Errado

Questão 23

ITnerante

[CESGRANRIO - 2011 - FINEP - Analista - Desenvolvimento de Sistemas]

Seja o seguinte vetor, ordenado de forma ascendente:



Caso se utilize um algoritmo de busca binária, quantas iterações serão necessárias para que o valor 80 seja encontrado?

(A) 2
(B) 3
(C) 4
(D) 8
(E) 9

Gabarito

17 – Certo

18 – Errado

19 – E

20 – Certo

21 – Errado

22 – Certo

23 – B

Algoritmos de Ordenação

- **Principais características**
  - Os algoritmos de ordenação são de fundamental importância na ciência da computação.
  - O melhor algoritmo para determinada aplicação depende, entre outros fatores, do número de itens a ordenar e do grau de ordenação já apresentado por esses itens.
  - Ao se ordenar um conjunto de valores, uma questão relevante é o tratamento de elementos que possuem o mesmo valor. Um algoritmo de ordenação é **estável** se não altera a posição relativa de elementos que têm um mesmo valor.

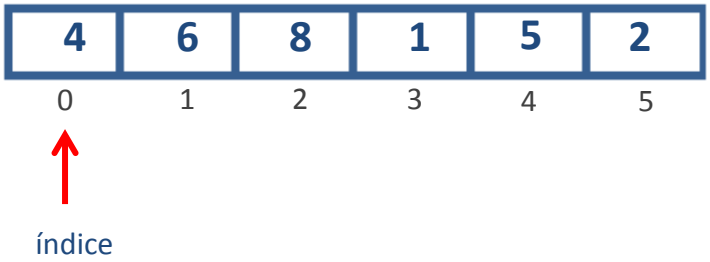
- **Principais características**
  - Considerando um vetor desordenado com os elementos abaixo:
    [44 , 55, 22_1, 66, 77, 33, 11, 22_2, 88]
  - Quando o vetor é ordenado por um algoritmo estável, a ordem relativa dos itens com chaves iguais mantém-se inalterada após a ordenação.
    Vetor ordenado com algoritmo estável
    [11, 22_1, 22_2, 33, 44, 55, 66, 77, 88]
  - Quando o vetor é ordenado por um algoritmo não estável, a ordem relativa dos itens com chaves iguais é alterada após a ordenação.
    Vetor ordenado com algoritmo não estável
    [11, 22_2, 22_1, 33, 44, 55, 66, 77, 88]

- **Selection Sort**
  - É um tipo de ordenação por seleção que consiste em trocar o menor elemento de uma lista com o elemento posicionado no inicio da lista, depois o segundo menor elemento para a segunda posição e assim sucessivamente.
  - Essa algoritmo admite uma variação onde a troca é baseada no maior elemento que é posicionado no final da lista.
  - Algoritmos de ordenação não estável.
  - Complexidade $O(n^2)$.

Implementação do Selection Sort

```
1   public void selectionSort(int[] valores){
2       int menorValor;
3
4       for ( int indice=0; indice<valores.length-1; indice++ ){
5           menorValor = indice;
6
7           for (int j=indice+1; j<valores.length; j++){
8               if (valores[menorValor]>valores[j]) {
9                   menorValor = j;
10              }
11          }
12
13          if (menorValor!=indice){
14              int aux = valores[indice];
15              valores[indice] = valores[menorValor];
16              valores[menorValor] = aux;
17          }
18
19      }
20  }
```

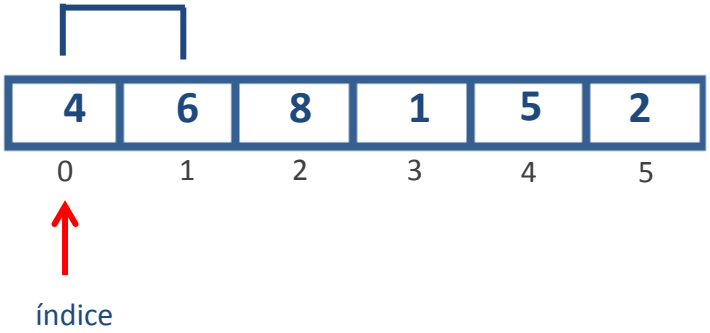
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 1 / inicio]

Posição menor valor: 0

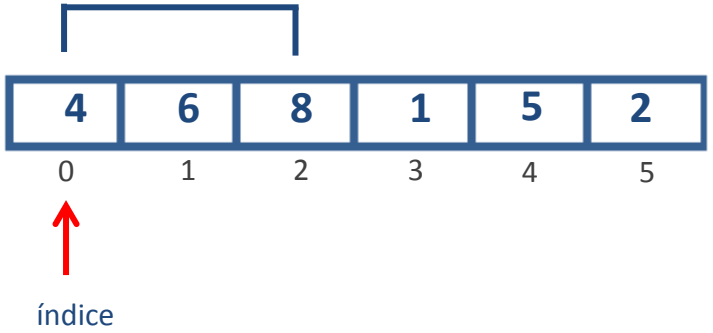
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 1 / comparação 1]

Posição menor valor: 0

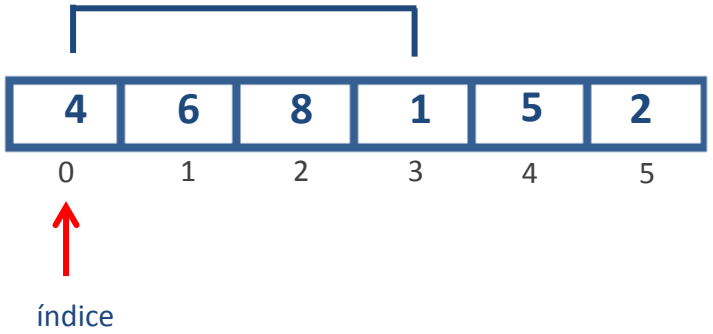
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 1 / comparação 2]

Posição menor valor: 0

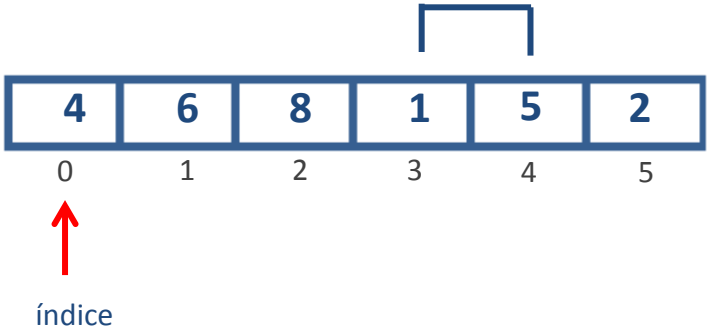
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 1 / comparação 3]

Posição menor valor: 3

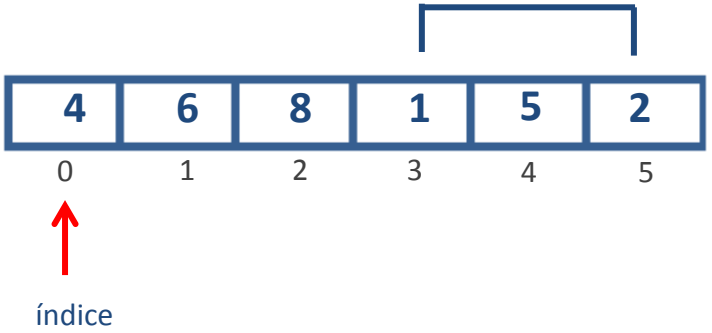
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 1 / comparação 4]

Posição menor valor: 3

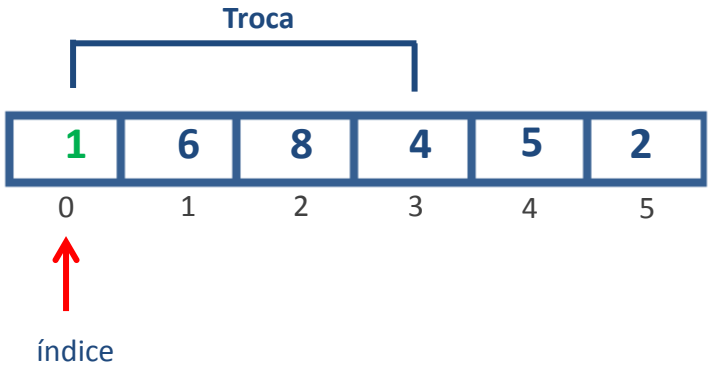
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 1 / comparação 5]

Posição menor valor: 3

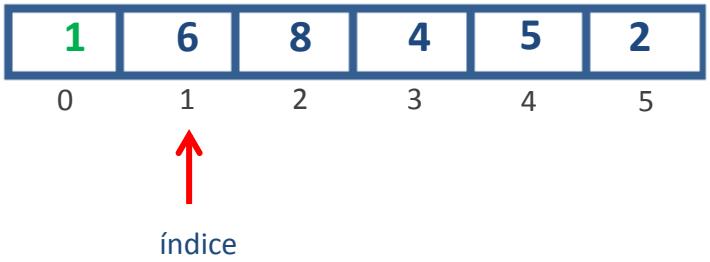
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 1 / final]

Posição menor valor: 3

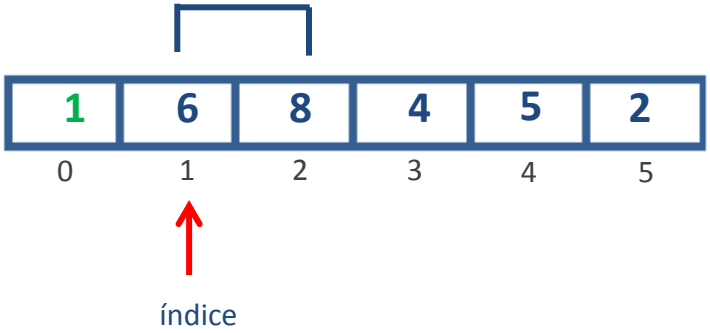
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 2 / inicio]

Posição menor valor: 1

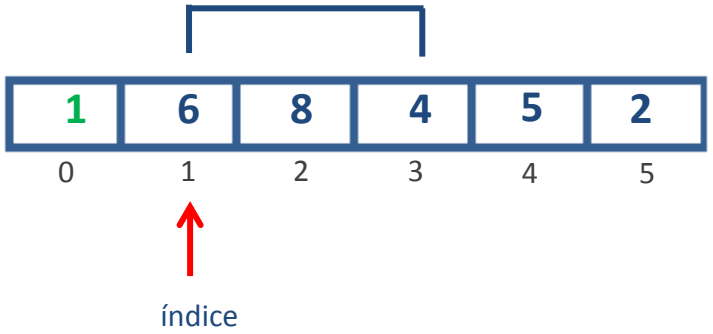
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 2 / comparação 1]

Posição menor valor: 1

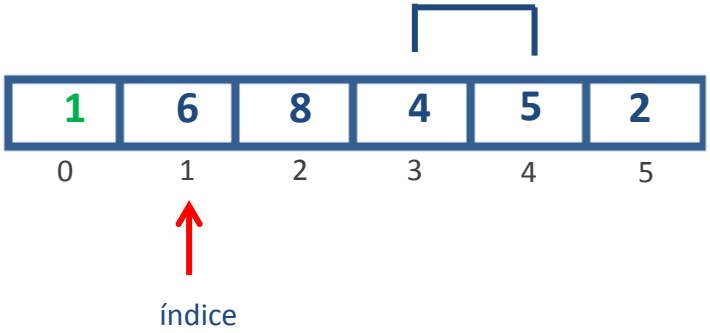
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 2 / comparação 2]

Posição menor valor: 3

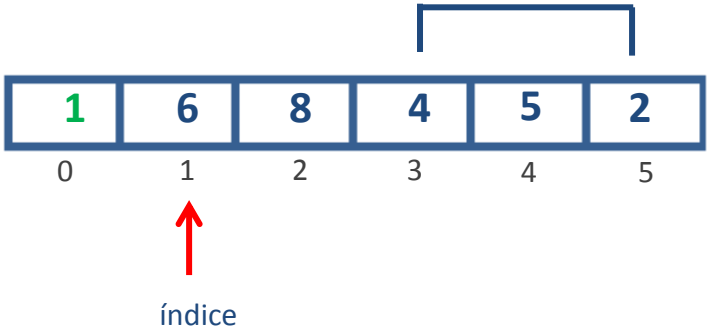
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 2 / comparação 3]

Posição menor valor: 3

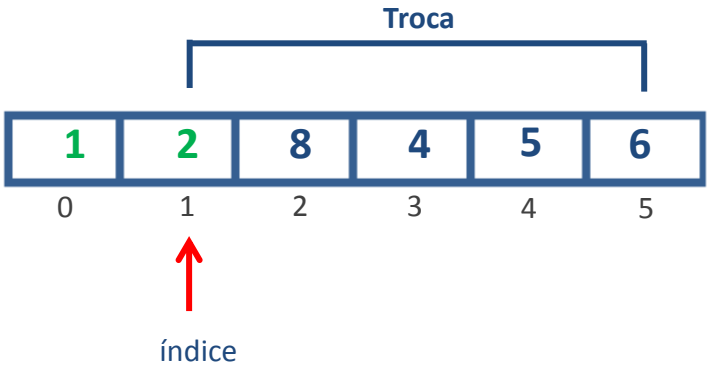
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 2 / comparação 4]

Posição menor valor: 5

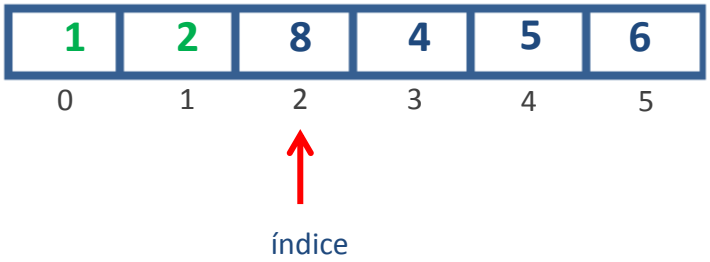
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 2 / final]

Posição menor valor: 5

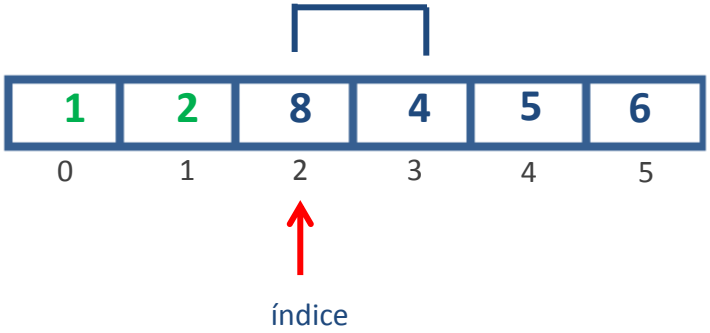
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 3 / inicio]

Posição menor valor: 2

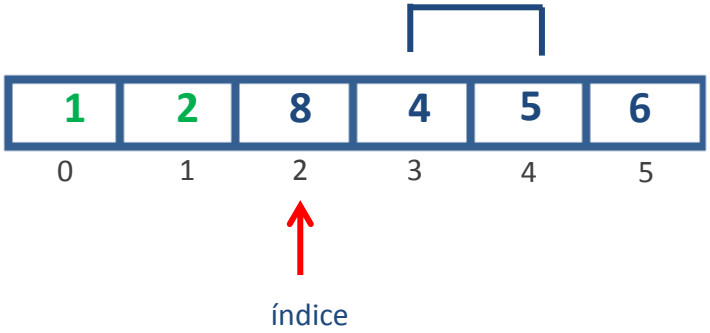
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 3 / comparação 1]

Posição menor valor: 3

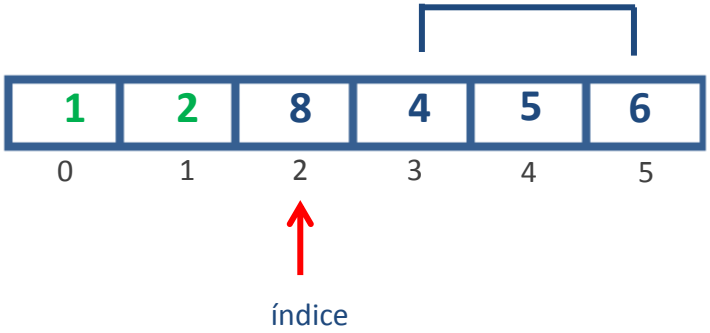
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 3 / comparação 2]

Posição menor valor: 3

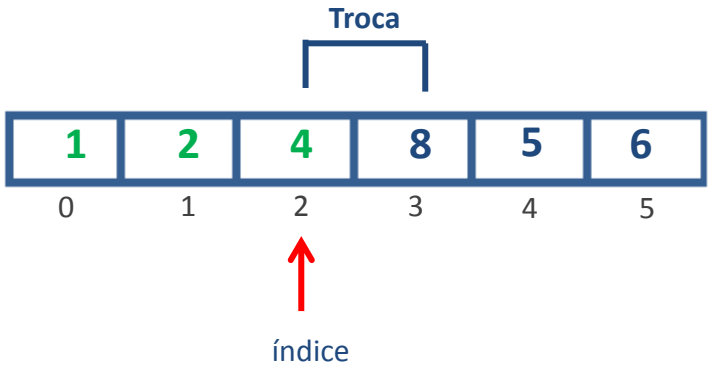
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 3 / comparação 3]

Posição menor valor: 3

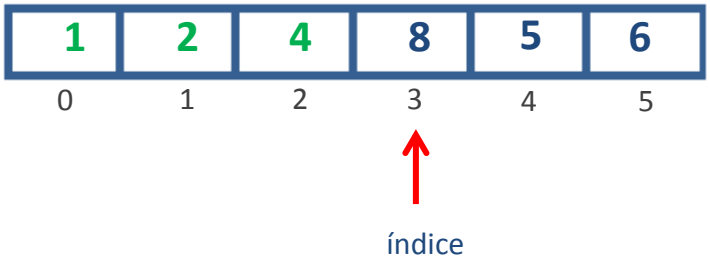
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 3 / final]

Posição menor valor: 3

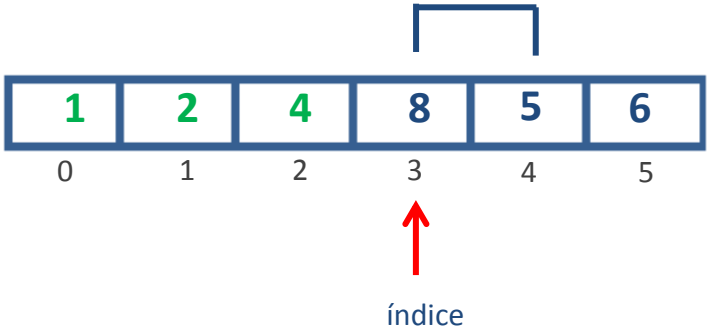
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 4 / inicio]

Posição menor valor: 3

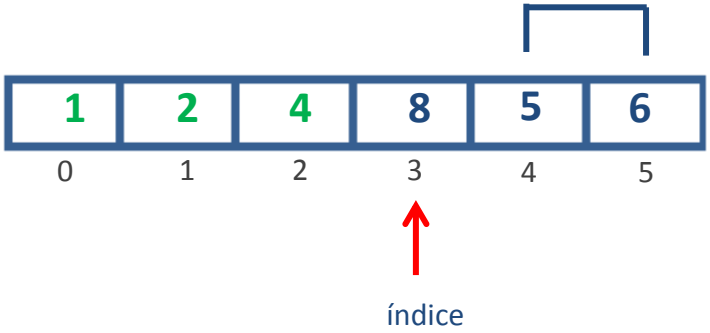
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 4 / comparação 1]

Posição menor valor: 4

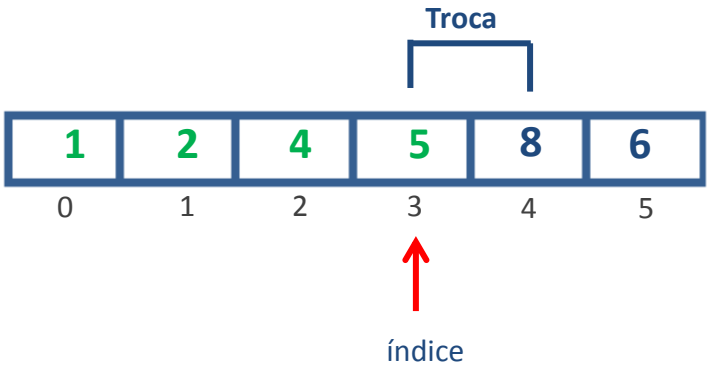
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 4 / comparação 2]

Posição menor valor: 4

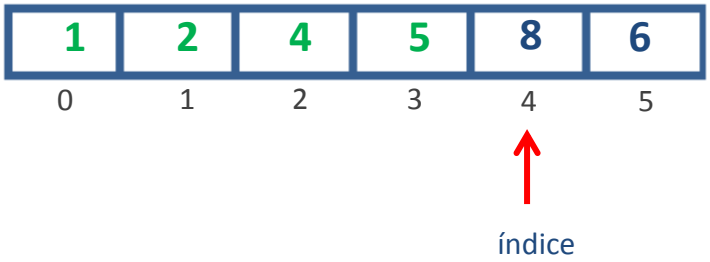
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 4 / final]

Posição menor valor: 4

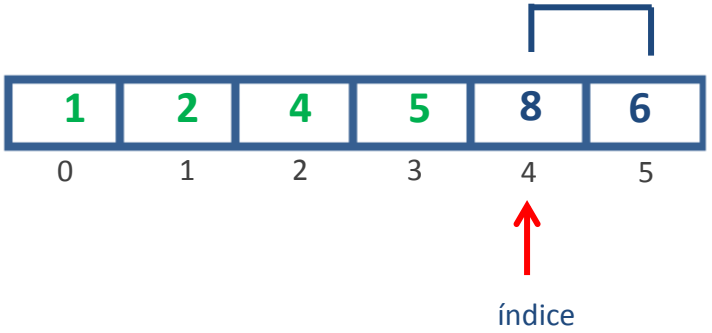
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 5 / inicio]

Posição menor valor: 4

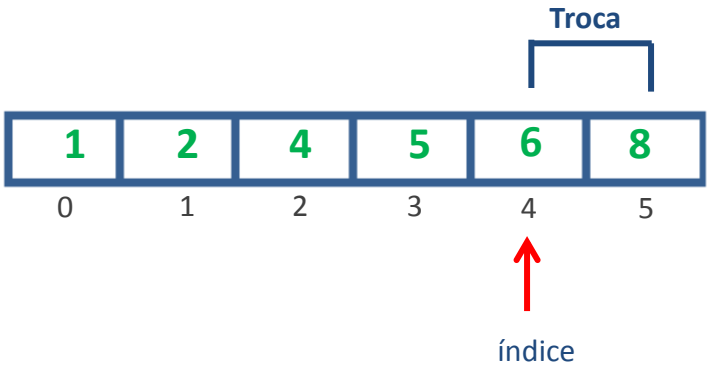
- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 5 / comparação 1]

Posição menor valor: 5

- Como ordenar o vetor abaixo com o Selection Sort?



[Iteração 5 / final]

Posição menor valor: 5

- Como ordenar o vetor abaixo com o Selection Sort?

- **Bubble Sort**
  - O algoritmo de ordenação bubble sort é um método simples de ordenação por troca.
    - O vetor é percorrido do inicio ao fim onde cada elemento é comparado com o elemento posterior.
    - Caso o elemento posterior seja menor, ocorre uma troca entre os dois elementos.
    - Esse procedimento garante que o maior elemento ficará no final do vetor.
  - Baixo desempenho para grandes quantidades de informações.

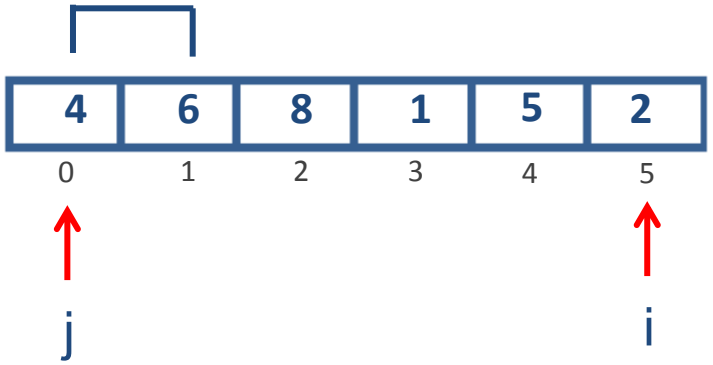
- **Bubble Sort**
  - Na primeira iteração são feitas n-1 comparações e o maior valor vai para o final do vetor. Na segunda iteração são feitas n-2 comparações e o segundo maior valor vai para o final do vetor, e assim por diante. Isso ocorre por que a cada iteração o maior valor que ainda não foi para sua posição, vai para o final do vetor.
  - Algoritmos de ordenação estável.
  - Complexidade no pior caso: $O(n^2)$.

ITnerante

- Implementação do Bubble Sort

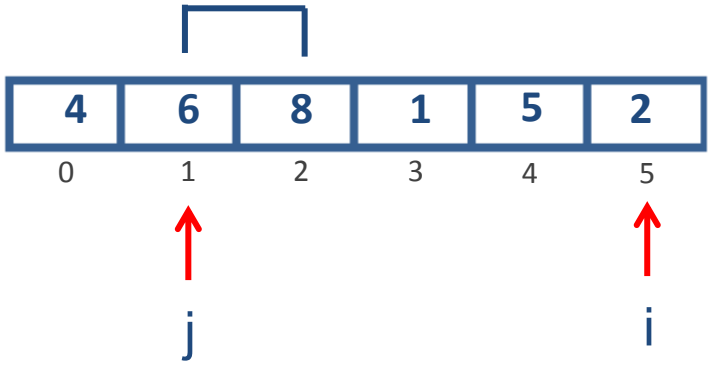
```
1  void bubbleSort(int valores[]) {
2      for (int i = valores.length - 1; i > 0; i--) {
3          for (int j = 0; j < i; j++) {
4
5              if (valores[j] > valores[j + 1]) {
6                  int aux = valores[j];
7                  valores[j] = valores[j + 1];
8                  valores[j + 1] = aux;
9              }
10         }
11     }
12 }
```

- Como ordenar o vetor abaixo com o Bubble Sort?



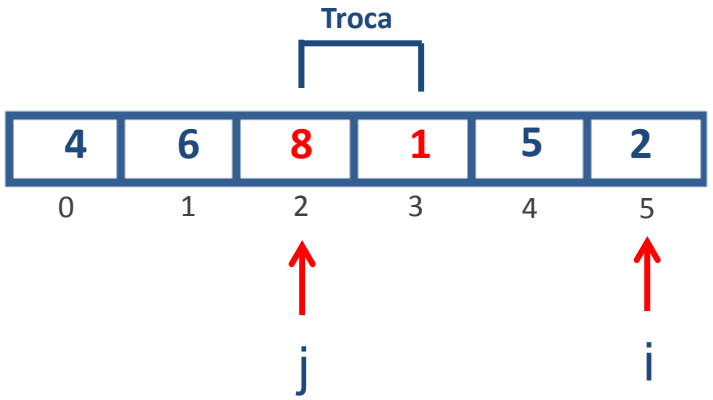
[Iteração 1 / comparação 1]

- Como ordenar o vetor abaixo com o Bubble Sort?



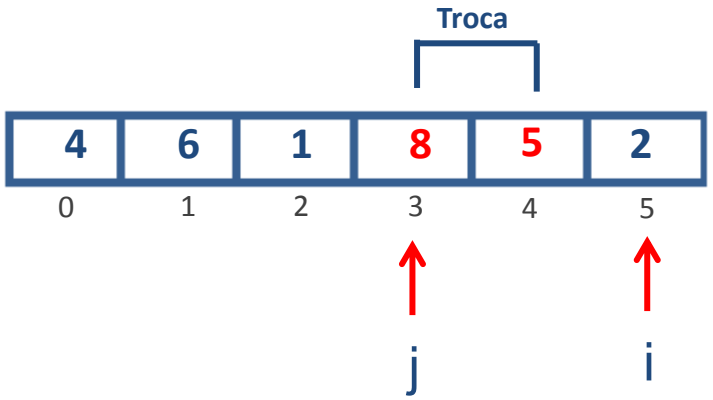
[Iteração 1 / comparação 2]

- Como ordenar o vetor abaixo com o Bubble Sort?



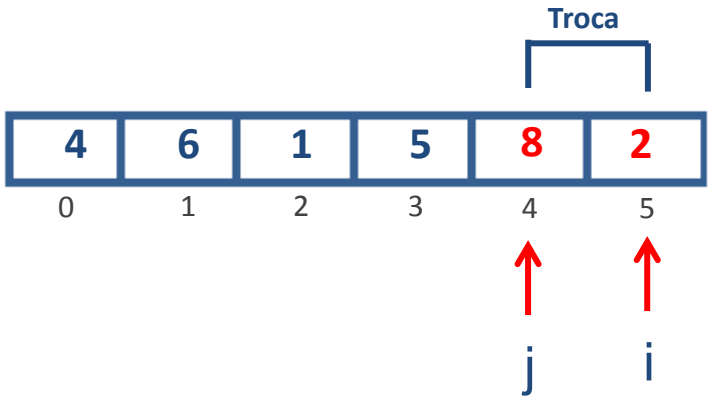
[Iteração 1 / comparação 3]

- Como ordenar o vetor abaixo com o Bubble Sort?



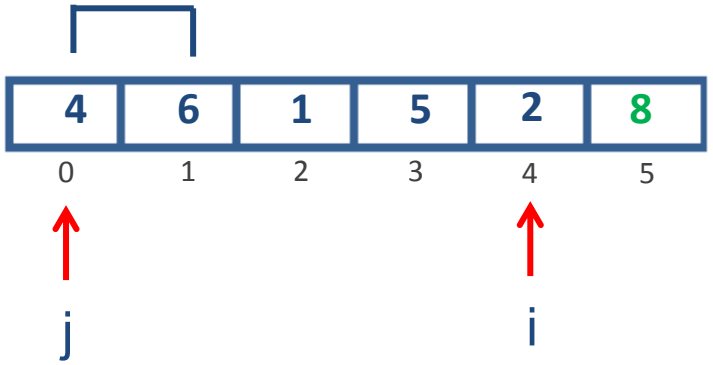
[Iteração 1 / comparação 4]

- Como ordenar o vetor abaixo com o Bubble Sort?



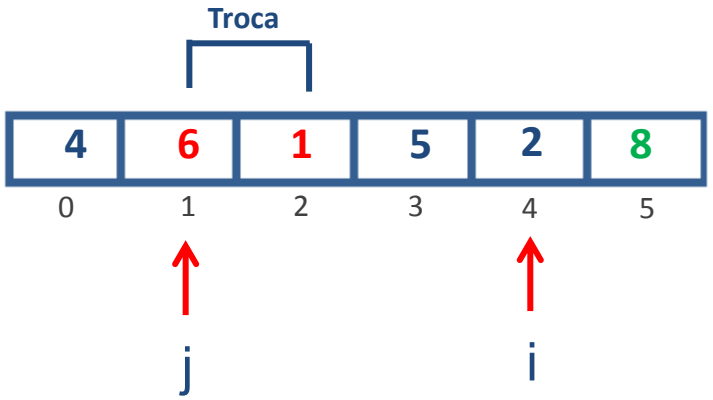
[Iteração 1 / comparação 5]

- Como ordenar o vetor abaixo com o Bubble Sort?



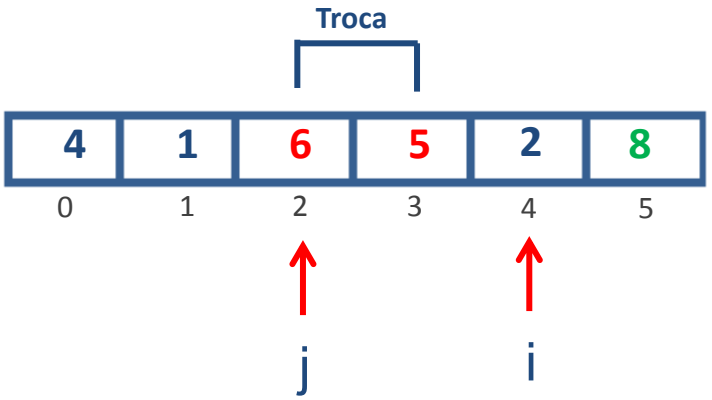
[Iteração 2 / comparação 1]

- Como ordenar o vetor abaixo com o Bubble Sort?



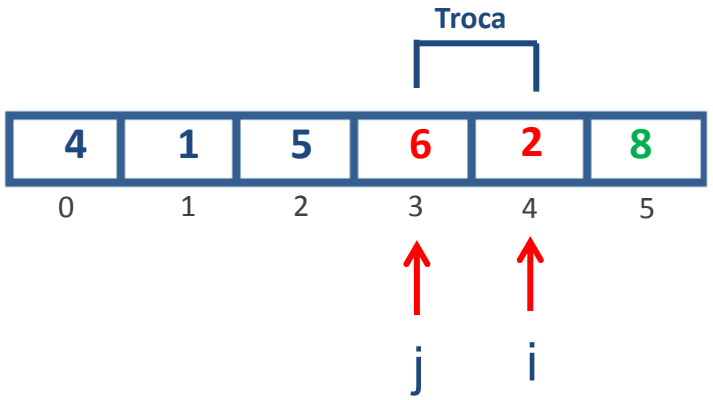
[Iteração 2 / comparação 2]

- Como ordenar o vetor abaixo com o Bubble Sort?



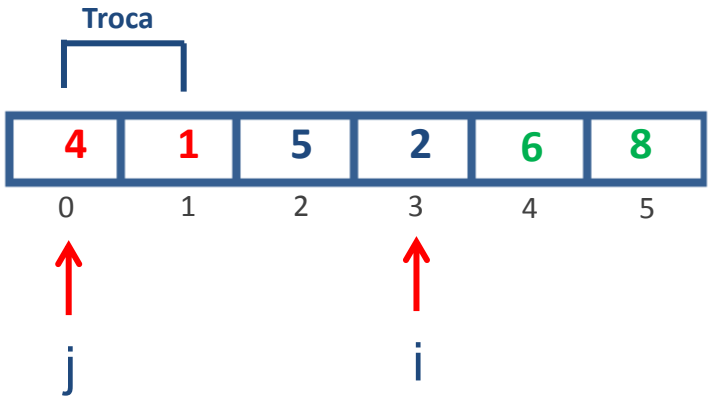
[Iteração 2 / comparação 3]

- Como ordenar o vetor abaixo com o Bubble Sort?



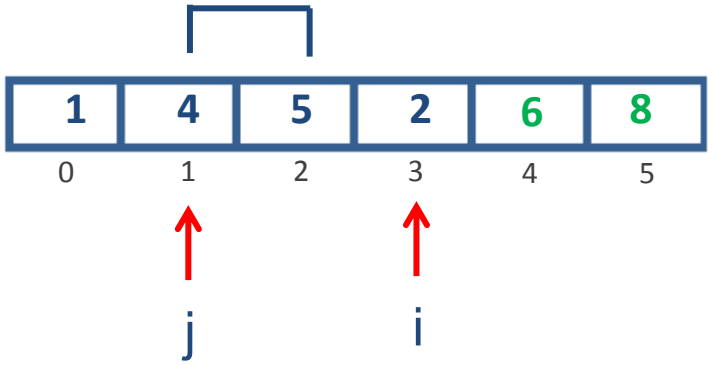
[Iteração 2 / comparação 4]

- Como ordenar o vetor abaixo com o Bubble Sort?



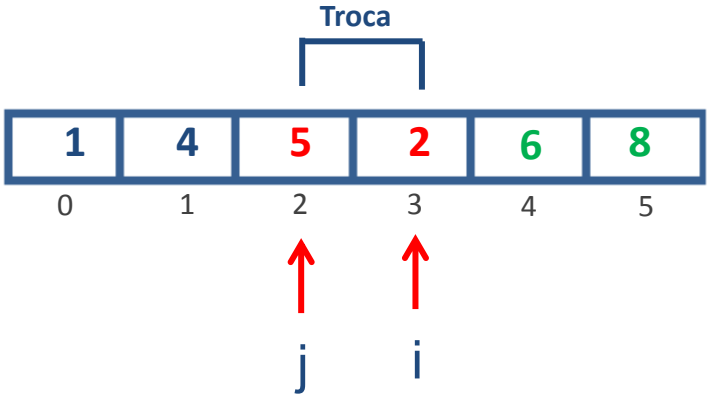
[Iteração 3 / comparação 1]

- Como ordenar o vetor abaixo com o Bubble Sort?



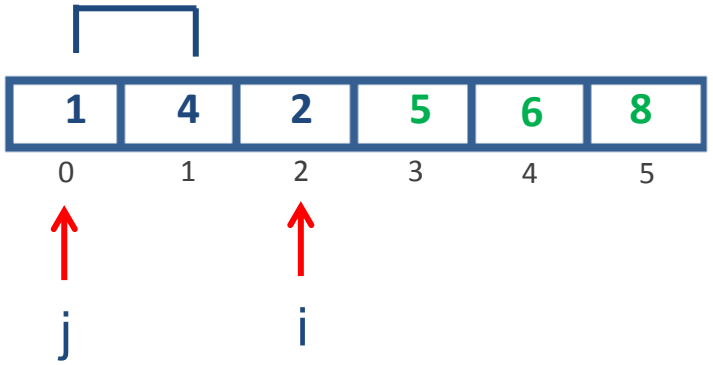
[Iteração 3 / comparação 2]

- Como ordenar o vetor abaixo com o Bubble Sort?



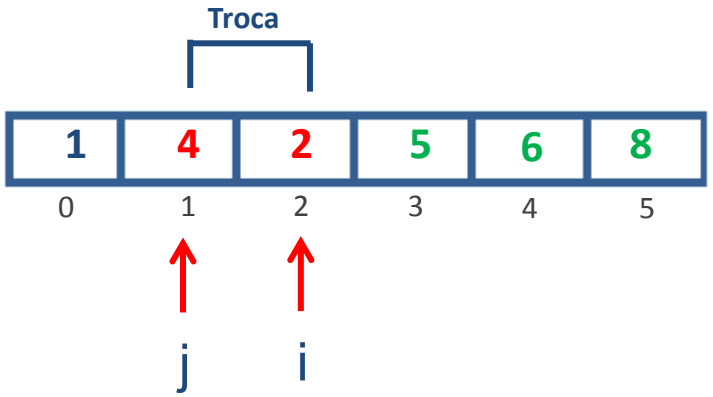
[Iteração 3 / comparação 3]

- Como ordenar o vetor abaixo com o Bubble Sort?



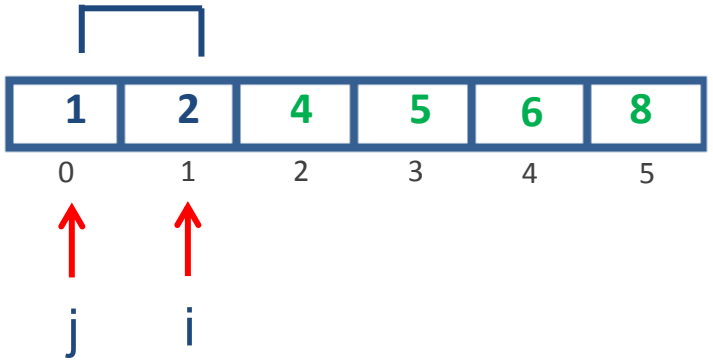
[Iteração 4 / comparação 1]

- Como ordenar o vetor abaixo com o Bubble Sort?



[Iteração 4 / comparação 2]

- Como ordenar o vetor abaixo com o Bubble Sort?



[Iteração 5 / comparação 1]

- Como ordenar o vetor abaixo com o Bubble Sort?

- **Ordenação por Inserção**
  - Os algoritmos inserção direta e ordenação shell são exemplos de algoritmos de ordenação por inserção.
  - Esse método pode ser comparado com as ações que realizamos ao ordenar uma mão de cartas de baralho. As cartas são inseridas, uma de cada vez, em seu local correto. Cada carta é inserida de forma a manter a ordem por naipe e valor.

- **Inserção Direta**
  - Comportamento:
    - O vetor é dividido em dois subgrupos: um com os elementos ordenados e outro com os não ordenados.
    - A princípio, o subgrupo ordenado tem somente o primeiro elemento do vetor e o segundo subgrupo todos os outros.
    - Cada elemento do subgrupo desordenado do vetor é colocado no subgrupo ordenado do vetor, um de cada vez, na posição correta.
    - Ao final desse processo, todos os elementos do vetor estarão ordenados.

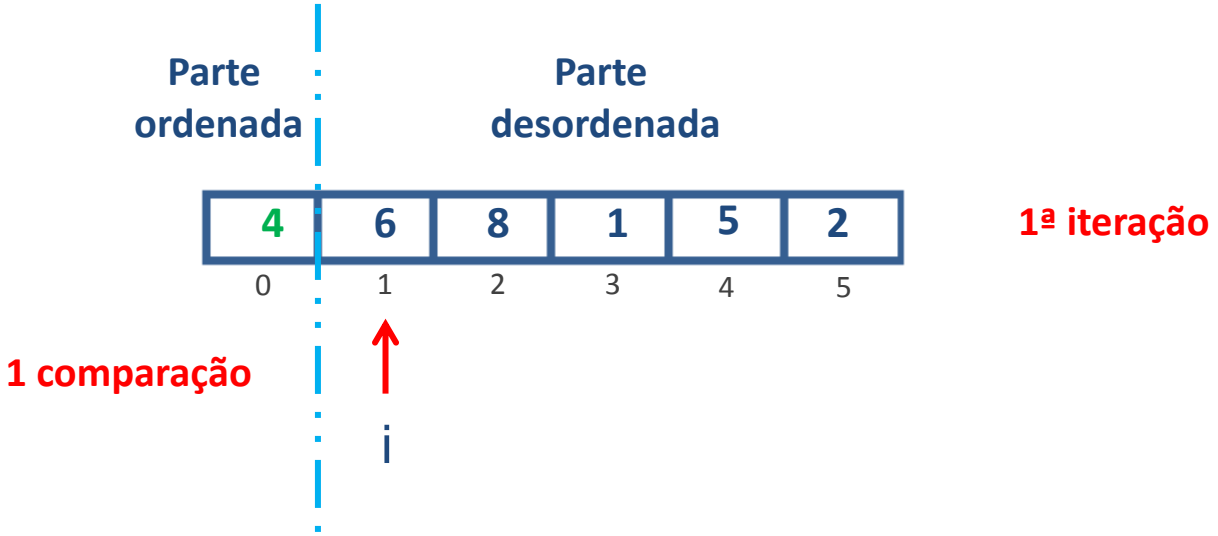
- **Inserção Direta**
  - Algoritmos de ordenação estável.
  - O algoritmo de inserção direta é mais eficiente se os elementos a serem ordenados já estiverem próximos de sua ordem final de classificação.
  - Suas complexidade no pior caso é $O(n^2)$. Ela ocorre quando o vetor esta totalmente desordenado.
  - Sua complexidade no melhor caso é $O(n)$. Ela ocorre quando o vetor esta totalmente ordenado.

Implementação da Inserção Direta

```
1   public void insertionSort( int[] valores ) {
2       for (int i = 1; i < valores.length; i++) {
3           int temp = valores[i];
4           int j;
5
6           for (j = i - 1; j >= 0; j--){
7               if (valores[j] < temp) {
8                   break;
9               }
10              valores[j+1] = valores[j];
11          }
12          valores[j+1] = temp;
13      }
14  }
```

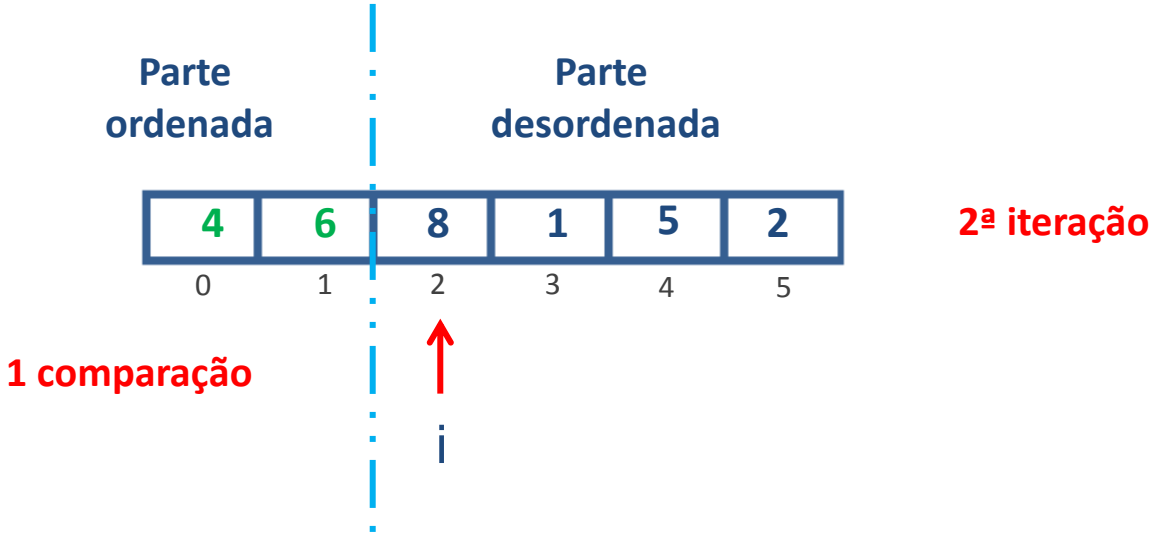
- **Como ordenar o vetor abaixo com a Inserção Direta?**

(Versão simplificada)



- **Como ordenar o vetor abaixo com a Inserção Direta?**

(Versão simplificada)



- **Como ordenar o vetor abaixo com a Inserção Direta?**

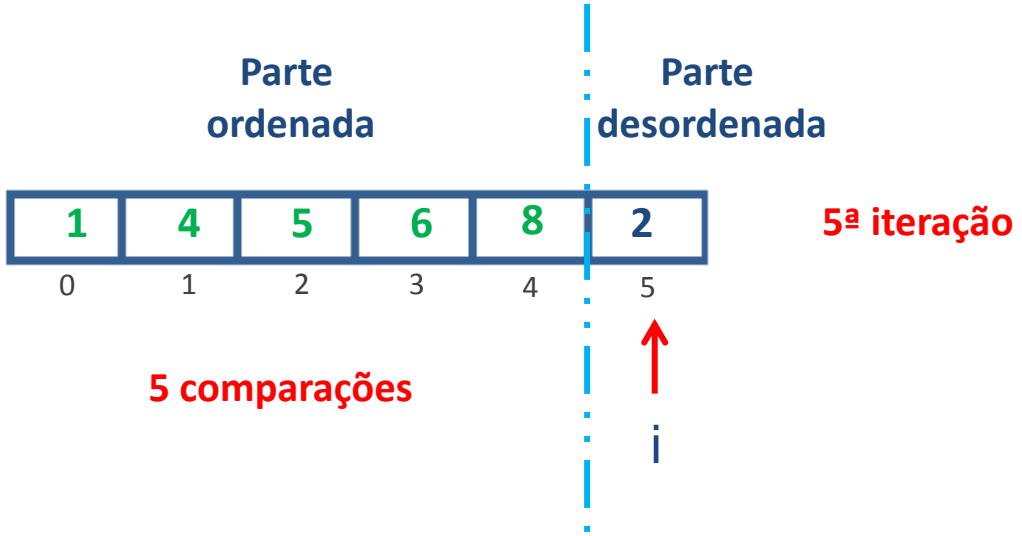
(Versão simplificada)

- **Como ordenar o vetor abaixo com a Inserção Direta?**

(Versão simplificada)

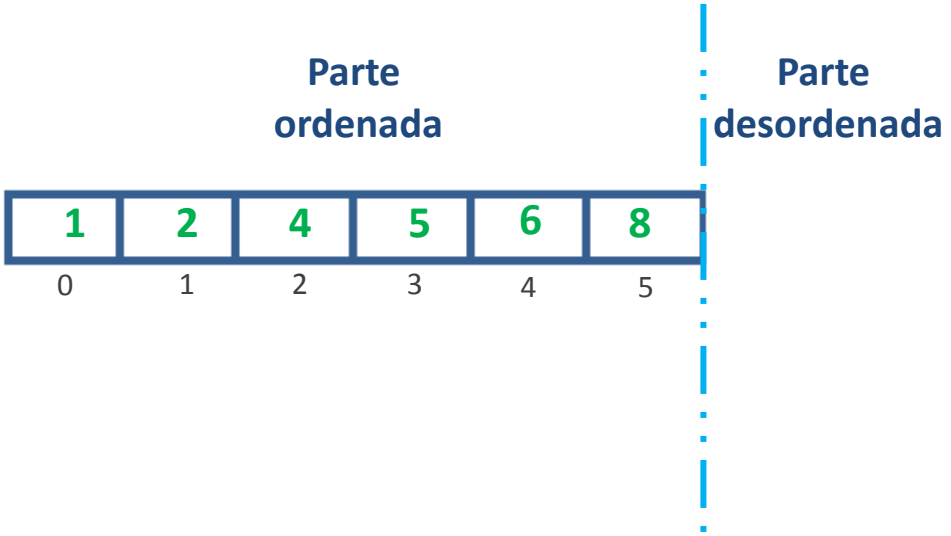
- **Como ordenar o vetor abaixo com a Inserção Direta?**

(Versão simplificada)



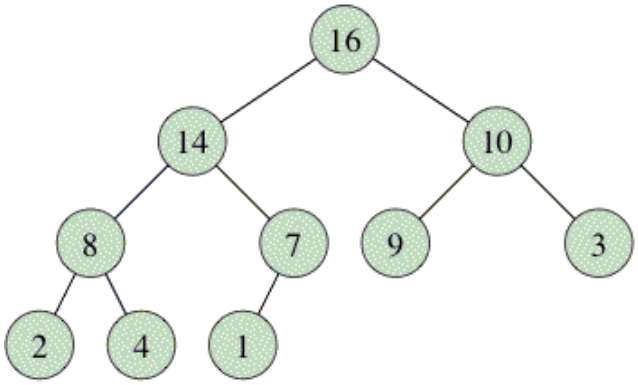
- **Como ordenar o vetor abaixo com a Inserção Direta?**

(Versão simplificada)

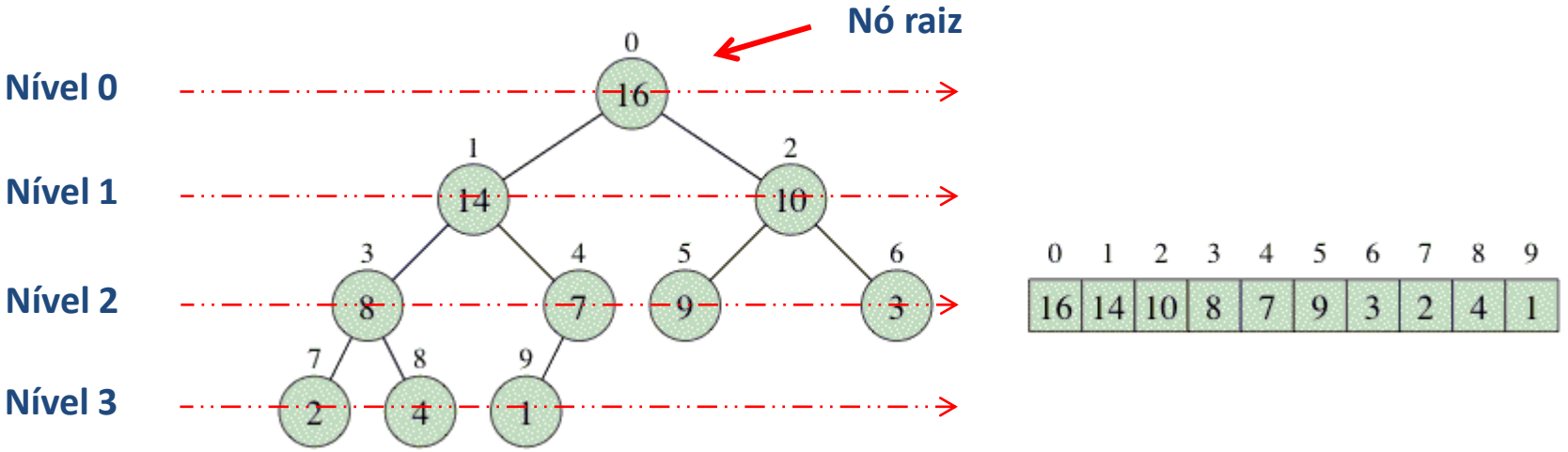


- **Heap Sort**
  - Esse algoritmo de ordenação faz uso de uma estrutura de dados chamada heap para gerenciar as informações.
  - A estrutura de dados heap pode ser vista como uma árvore binária quase completa.
  - Existem dois tipos de heaps, o algoritmo de ordenação por heap utiliza heaps de máximo. Nesse tipo de heap, o valor de um nó deve ser no máximo o valor do seu pai.
  - Heap sort é um algoritmo de ordenação não estável.
  - Possui complexidade O(nlogn).

- **Operações de um heap**
  - Max-Heapify – essa operação é usada para manter a propriedade de um heap de máximo. Ela possui como entrada o arranjo A e o índice i para o arranjo. Essa operação permite que A[i] obedeça à propriedade do heap de máximo.
  - Build-Max-Heap – essa operação converte um arranjo em um heap de máximo.



- Representação de um heap de máximo

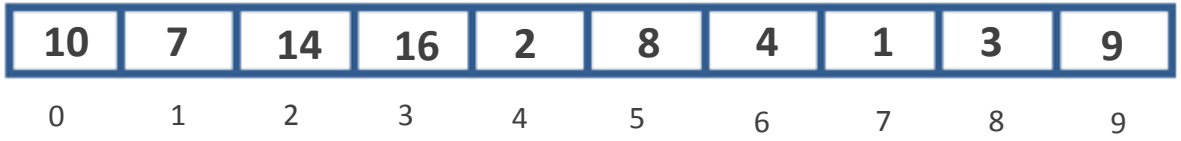


- **Funcionamento do Heap Sort**
  - Construção de um **heap de máximo** no arranjo de entrada (*Build-Max-Heap*).
  - A sequencia abaixo é executada até que não reste mais nenhum elemento no heap:
    - Coloca-se o A[0] (maior elemento) em sua posição correta no vetor, efetuando uma troca entre os elementos.
    - O elemento de maior valor é excluindo logicamente do heap.
    - Faz com que o elemento A[0] mantenha a propriedade de um heap máximo (*Max-Heapify*).

- Implementação do Heap Sort em java

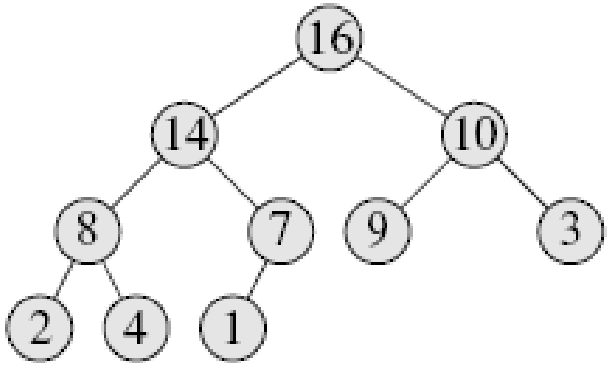
```
1  public void heapSort(int[] valores) {
2      buildMaxHeap(valores);
3      int comprimento = valores.length;
4
5      for (int i = valores.length - 1; i > 0; i--) {
6          swap(valores, i, 0);
7          maxHeapify(valores, 0, --comprimento);
8      }
9  }
```

- Como ordenar o vetor abaixo com o Heap Sort?



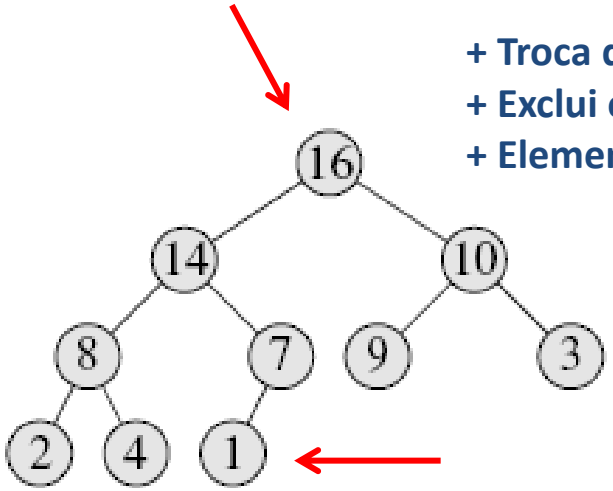
- Como ordenar o vetor abaixo com o Heap Sort?

+ Construção da heap de máximo



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

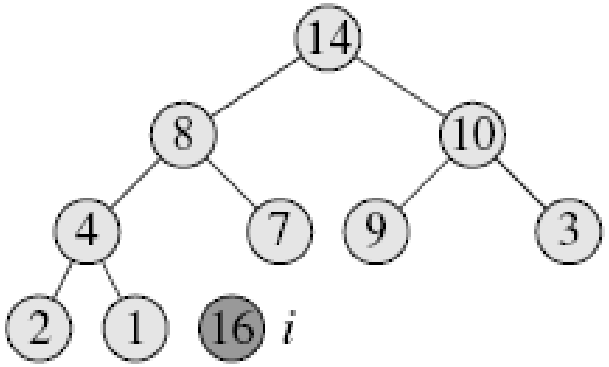
- Como ordenar o vetor abaixo com o Heap Sort?

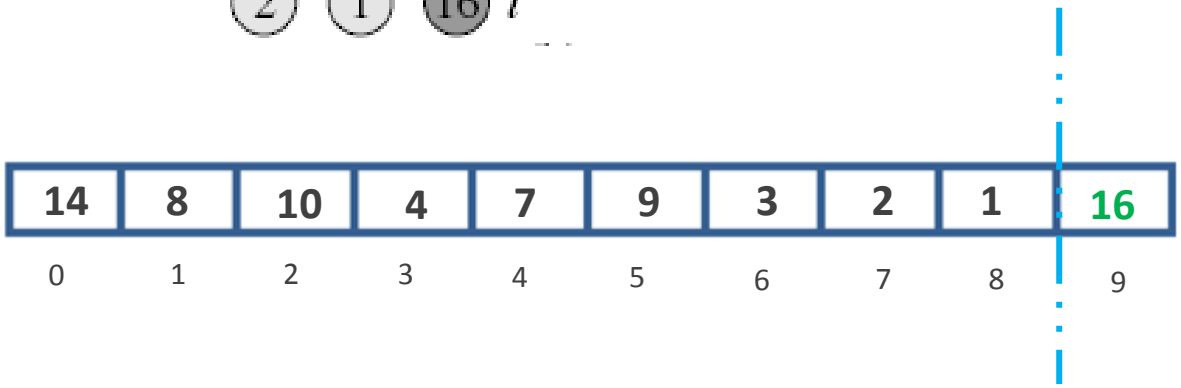


+ Troca dos Elementos.
+ Exclui o elemento de maior valor do heap.
+ Elemento A[0] mantém as características do heap.

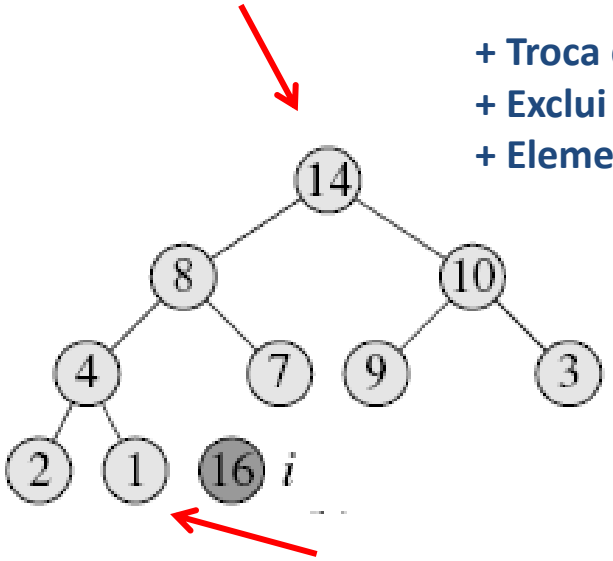
16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

- Como ordenar o vetor abaixo com o Heap Sort?

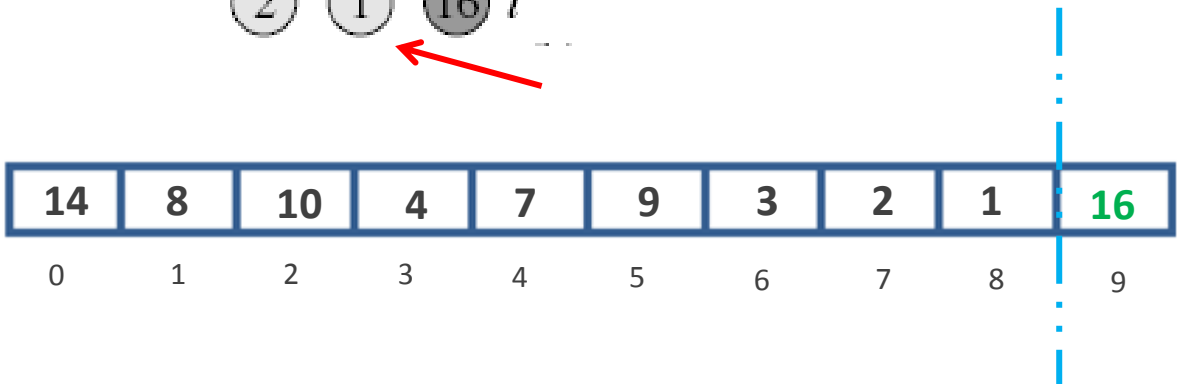




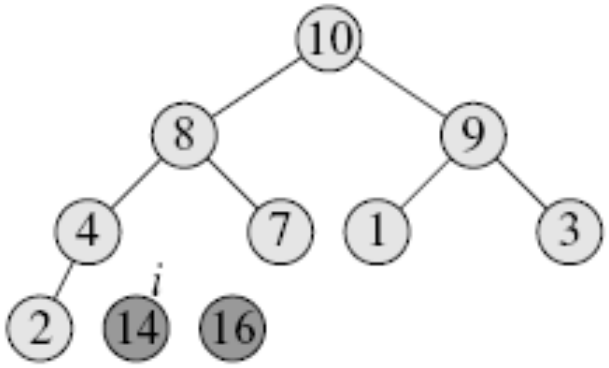
- Como ordenar o vetor abaixo com o Heap Sort?

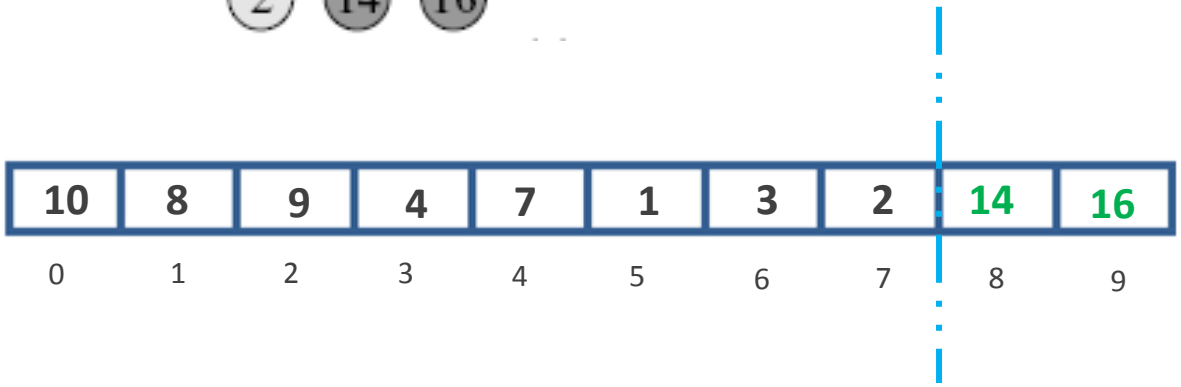


+ Troca dos Elementos
+ Exclui o elemento de maior valor do heap
+ Elemento A[0] mantém as características do heap

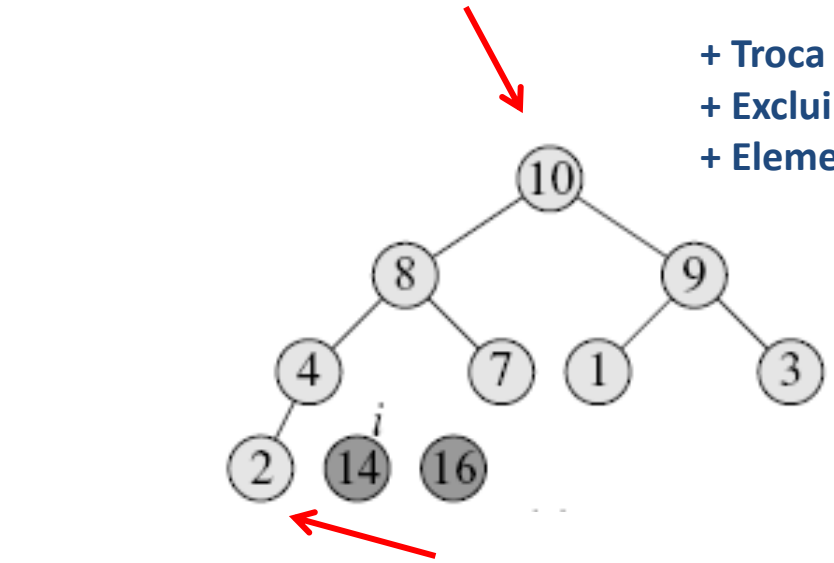


- Como ordenar o vetor abaixo com o Heap Sort?

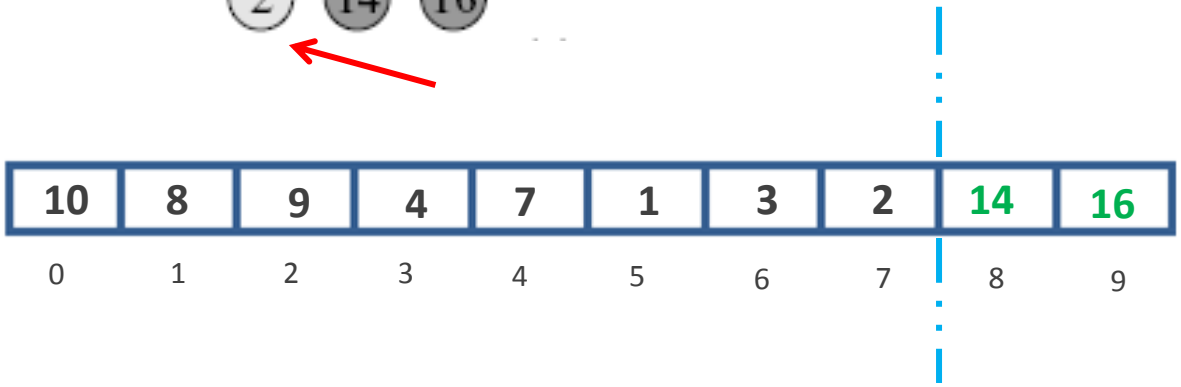




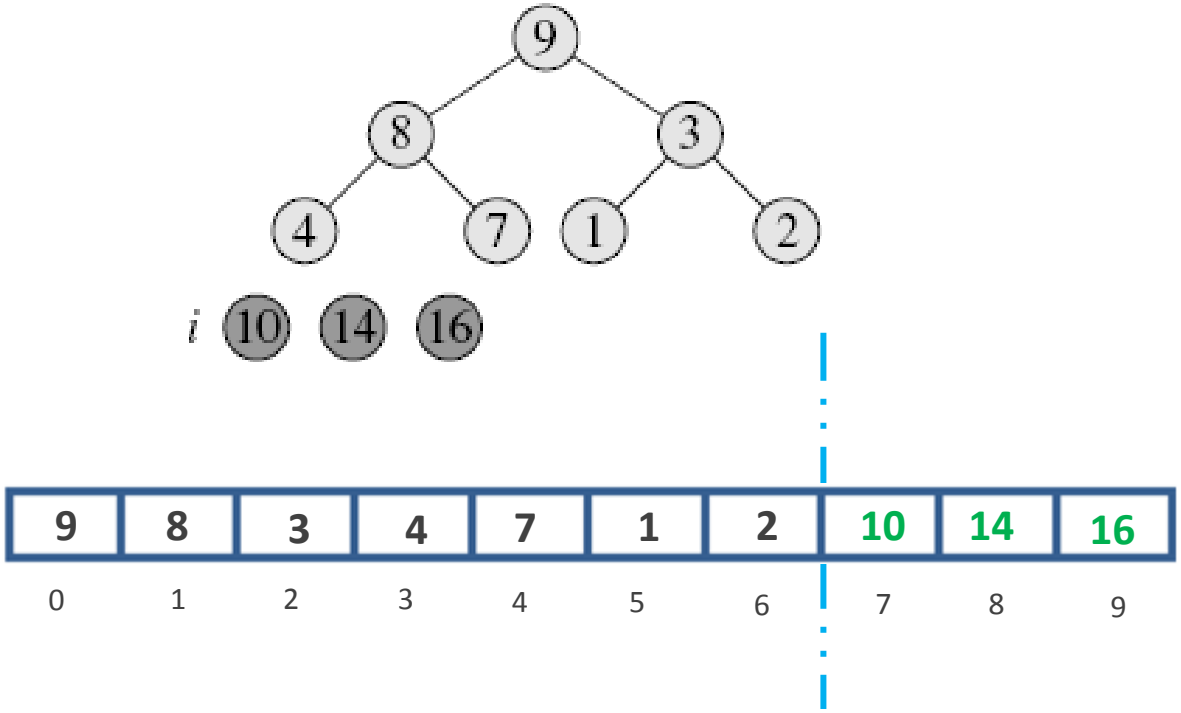
- Como ordenar o vetor abaixo com o Heap Sort?



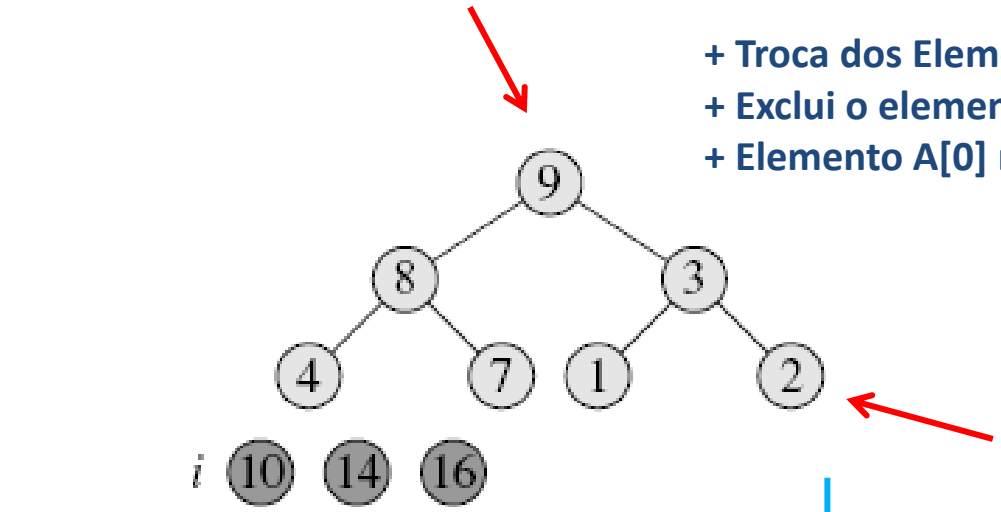
+ Troca dos Elementos
+ Exclui o elemento de maior valor do heap
+ Elemento A[0] mantém as características do heap



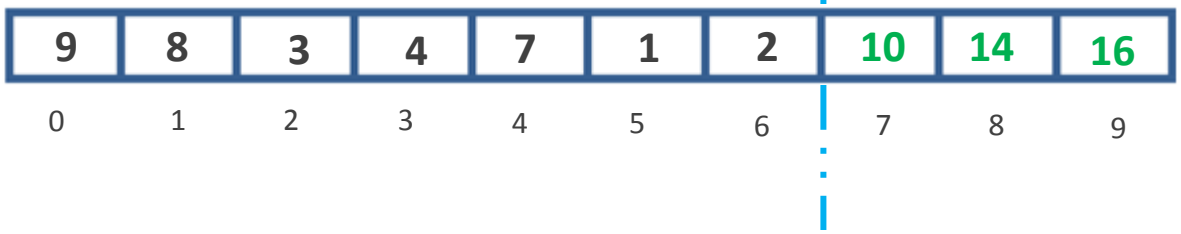
- Como ordenar o vetor abaixo com o Heap Sort?



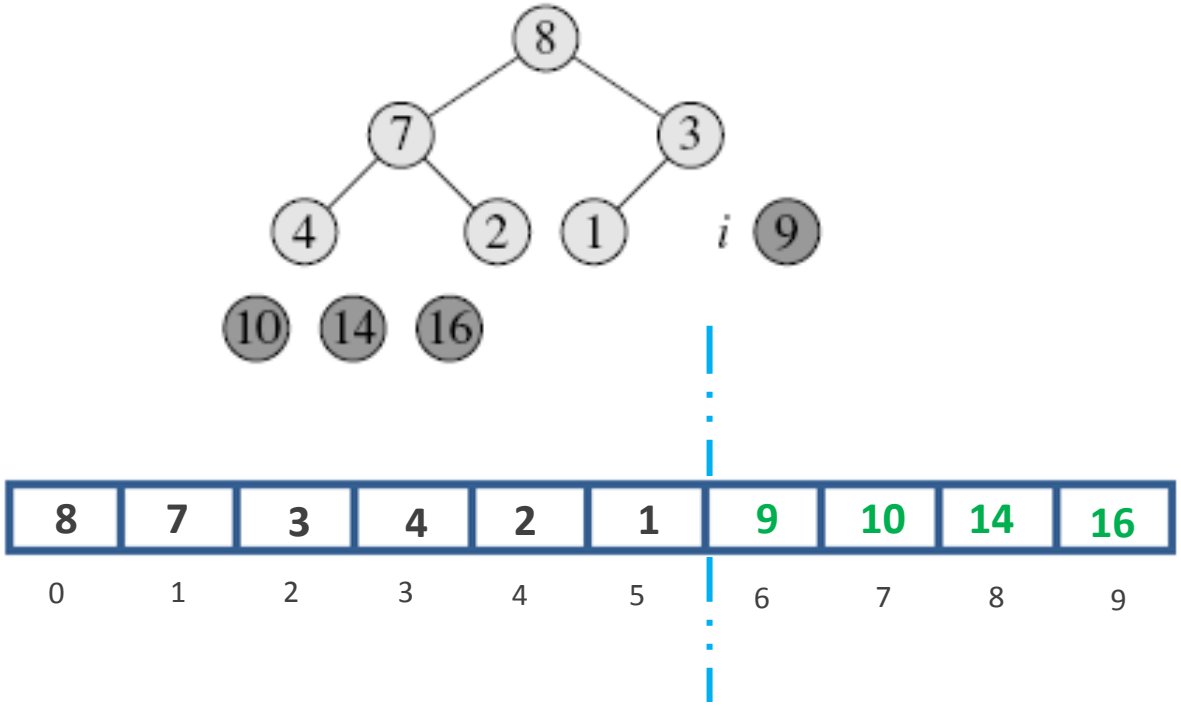
- Como ordenar o vetor abaixo com o Heap Sort?



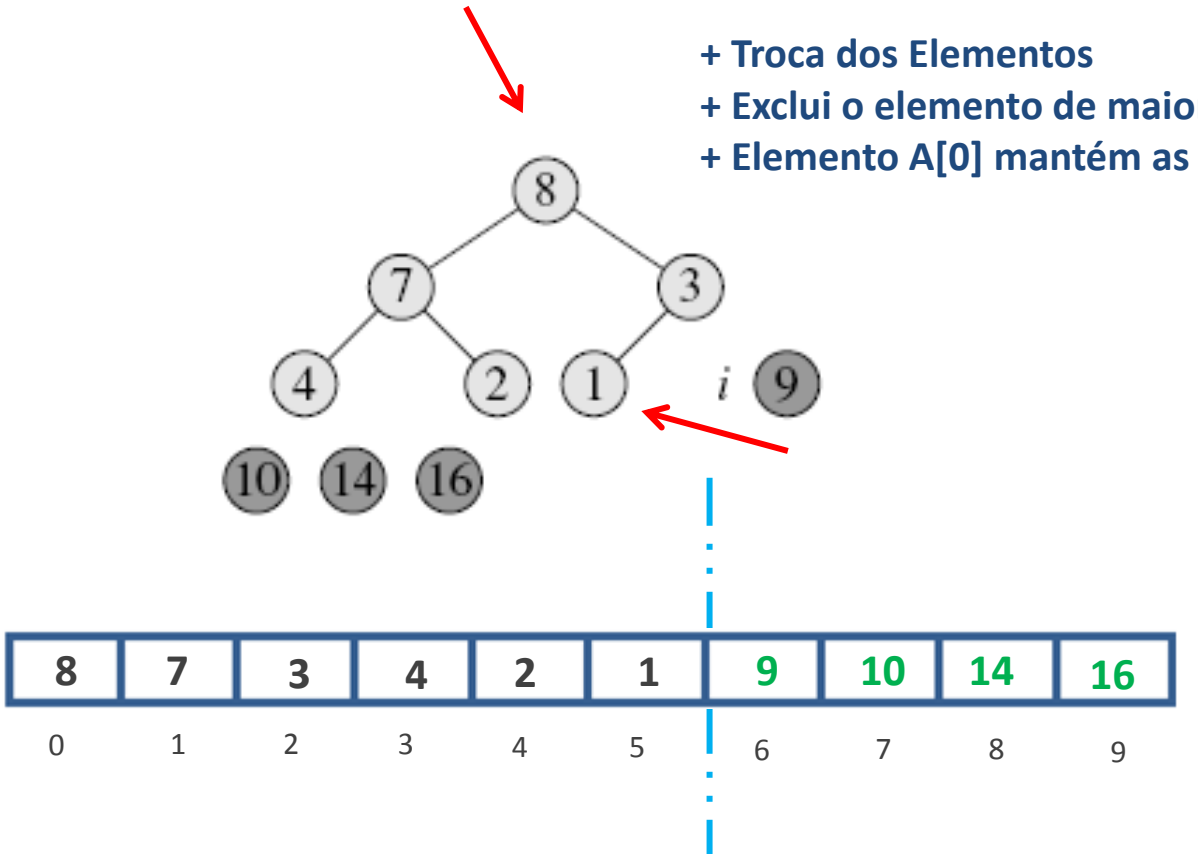
+ Troca dos Elementos
+ Exclui o elemento de maior valor do heap
+ Elemento A[0] mantém as características do heap



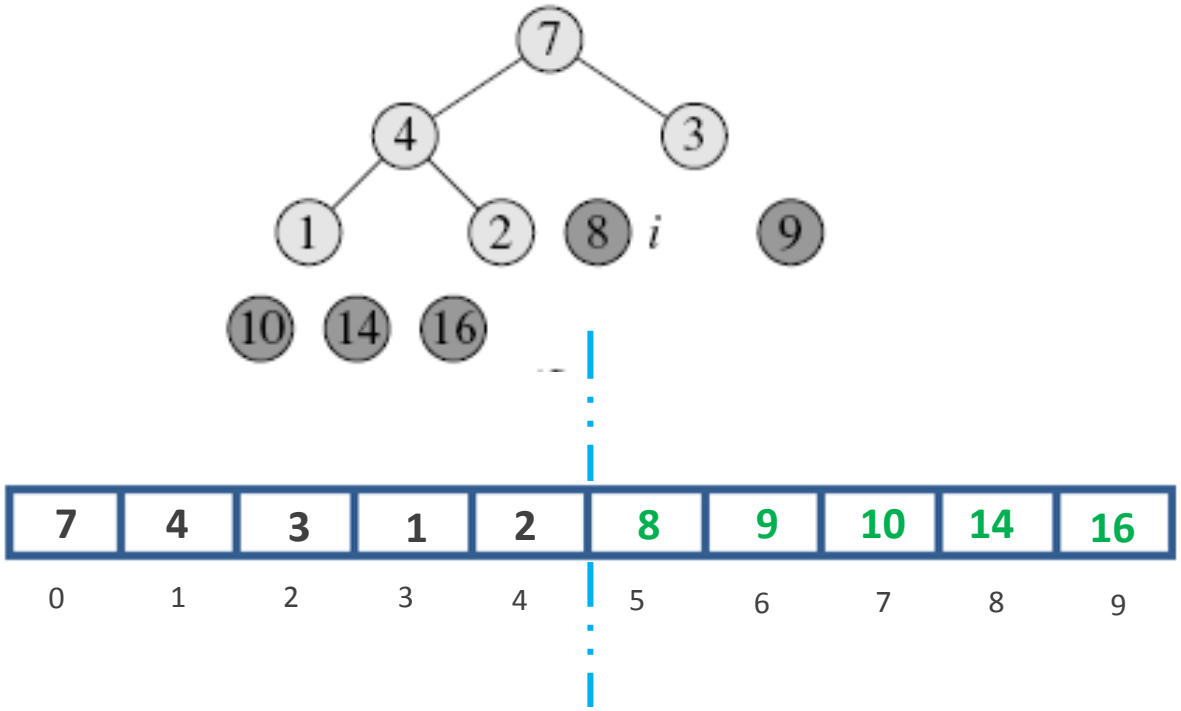
- Como ordenar o vetor abaixo com o Heap Sort?



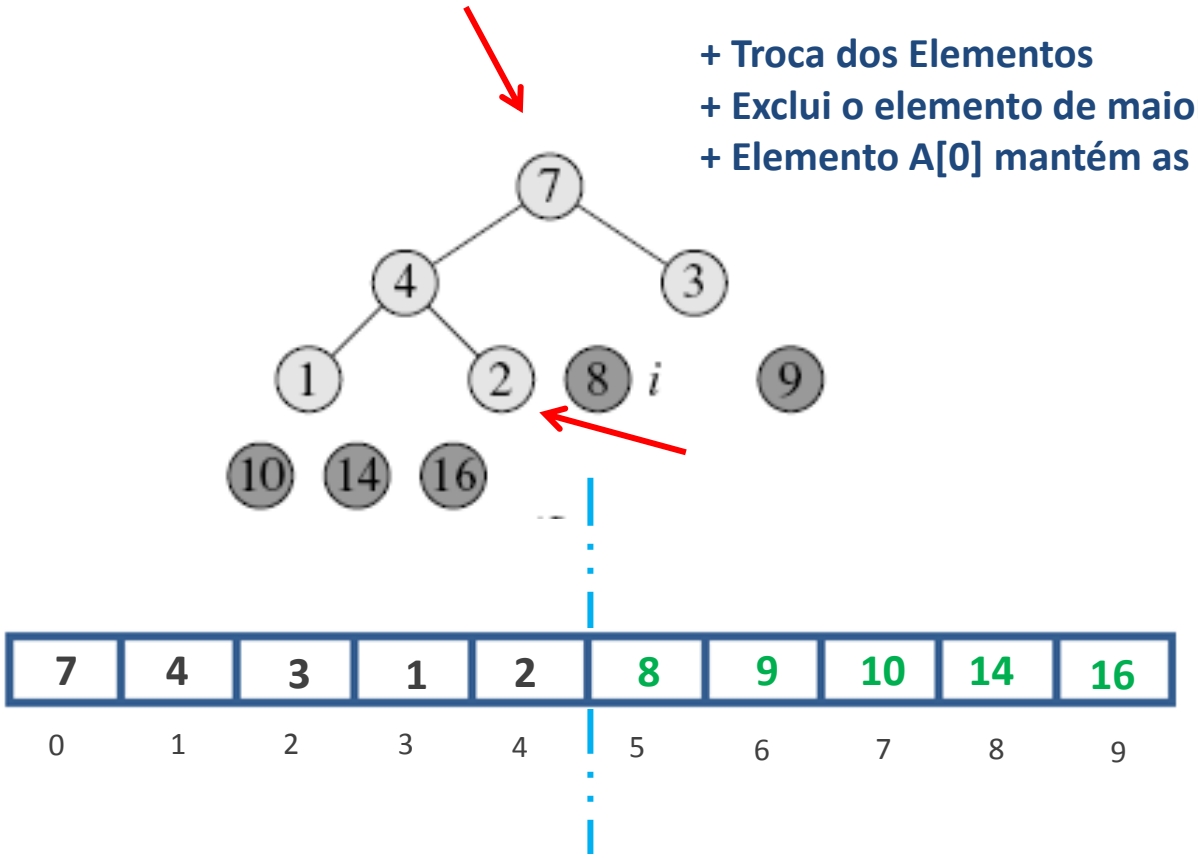
- Como ordenar o vetor abaixo com o Heap Sort?



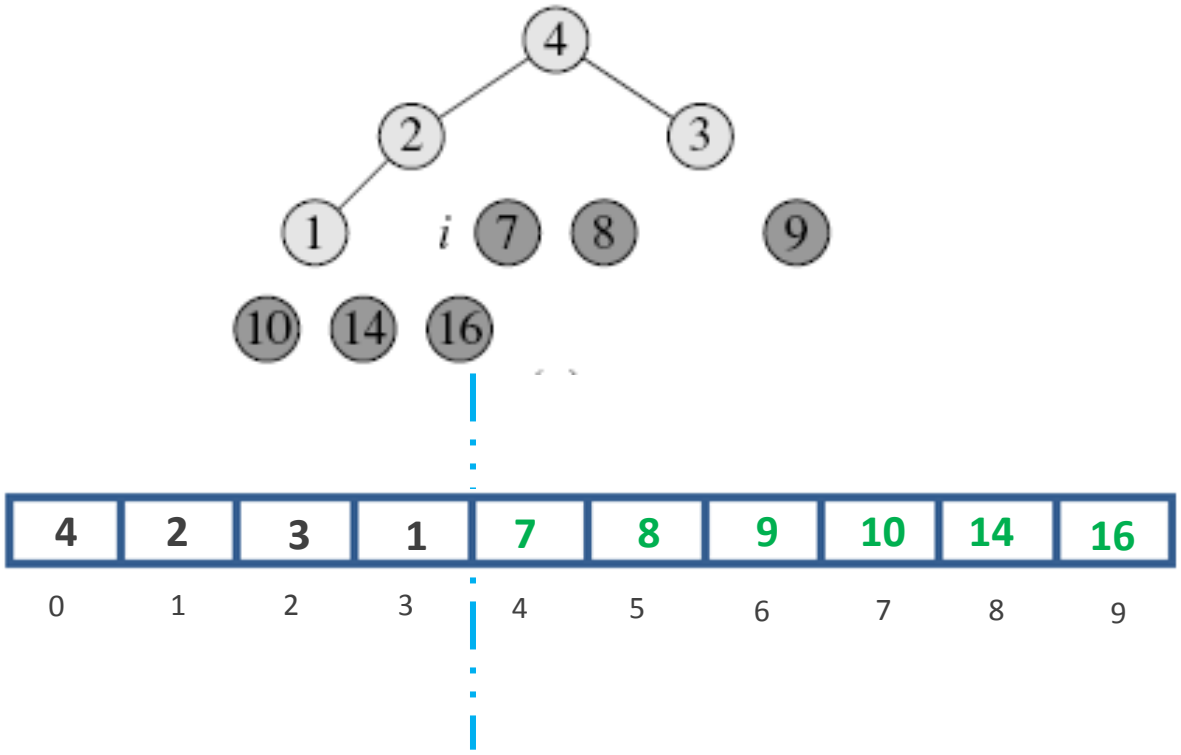
- Como ordenar o vetor abaixo com o Heap Sort?



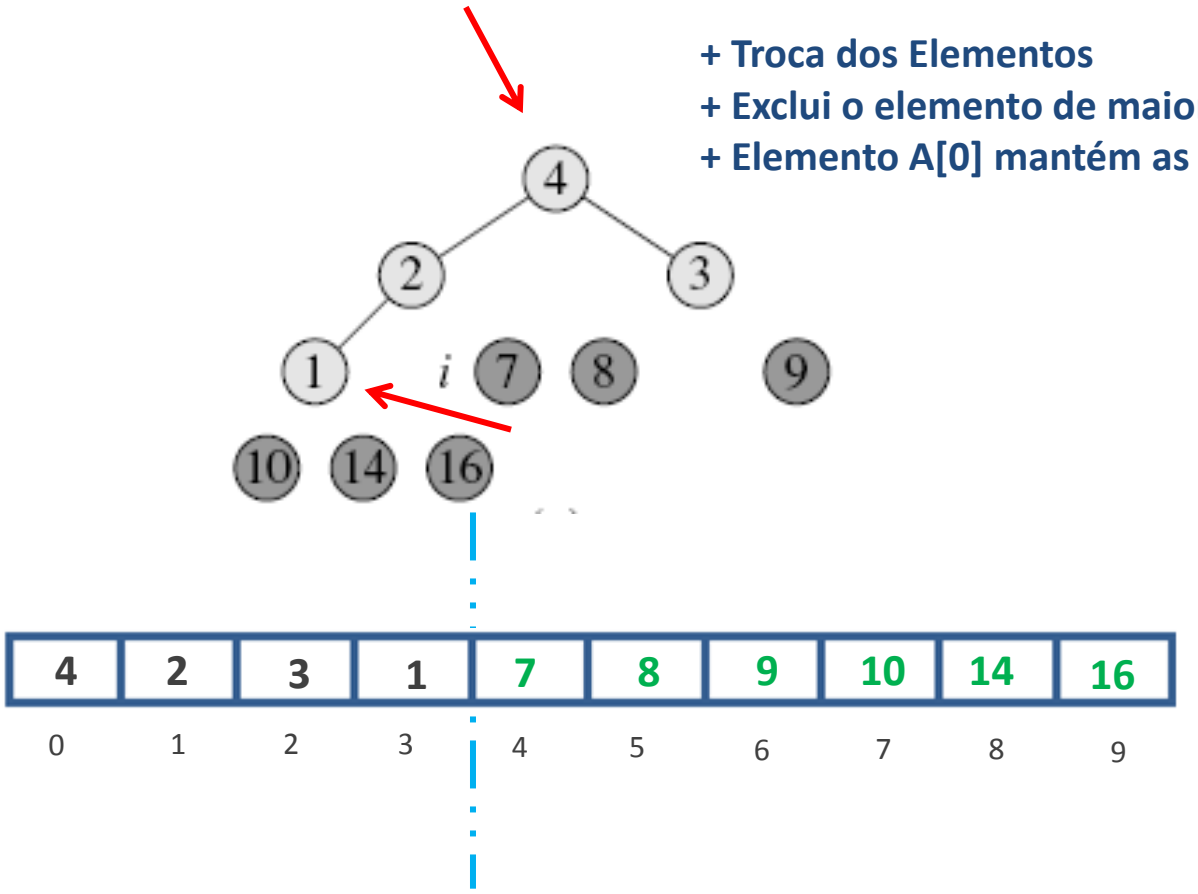
- Como ordenar o vetor abaixo com o Heap Sort?



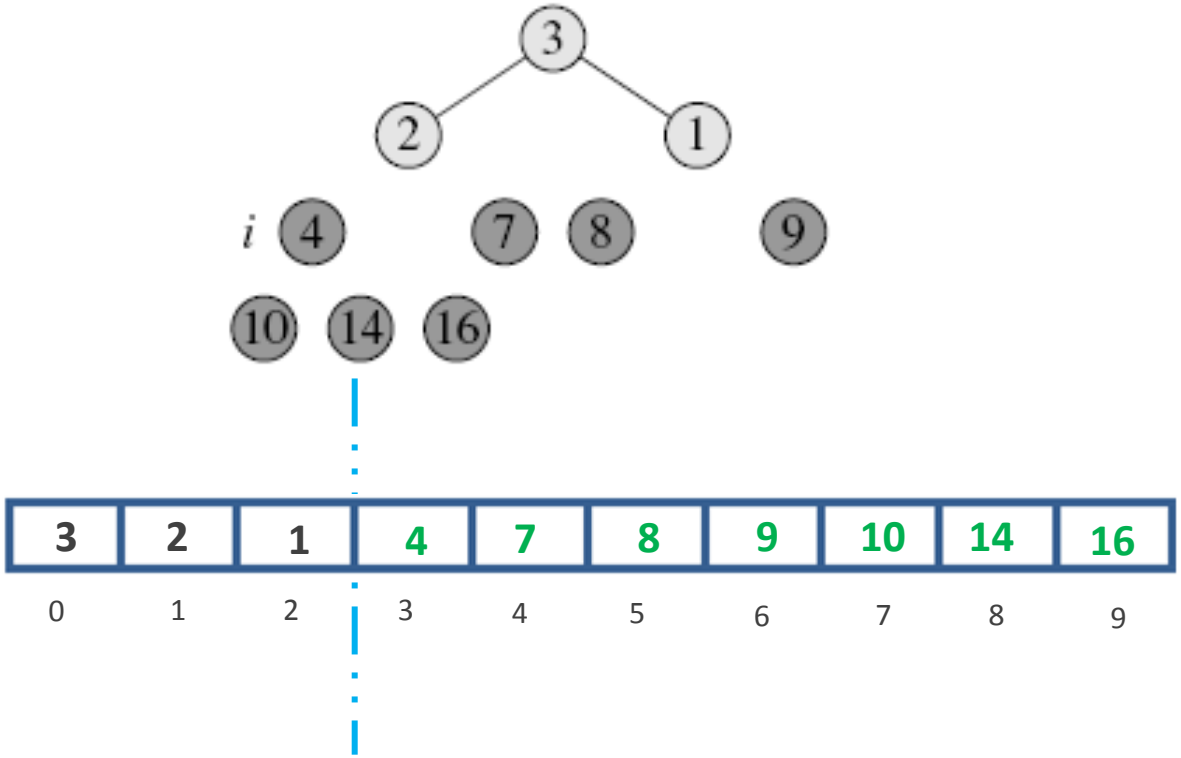
- Como ordenar o vetor abaixo com o Heap Sort?



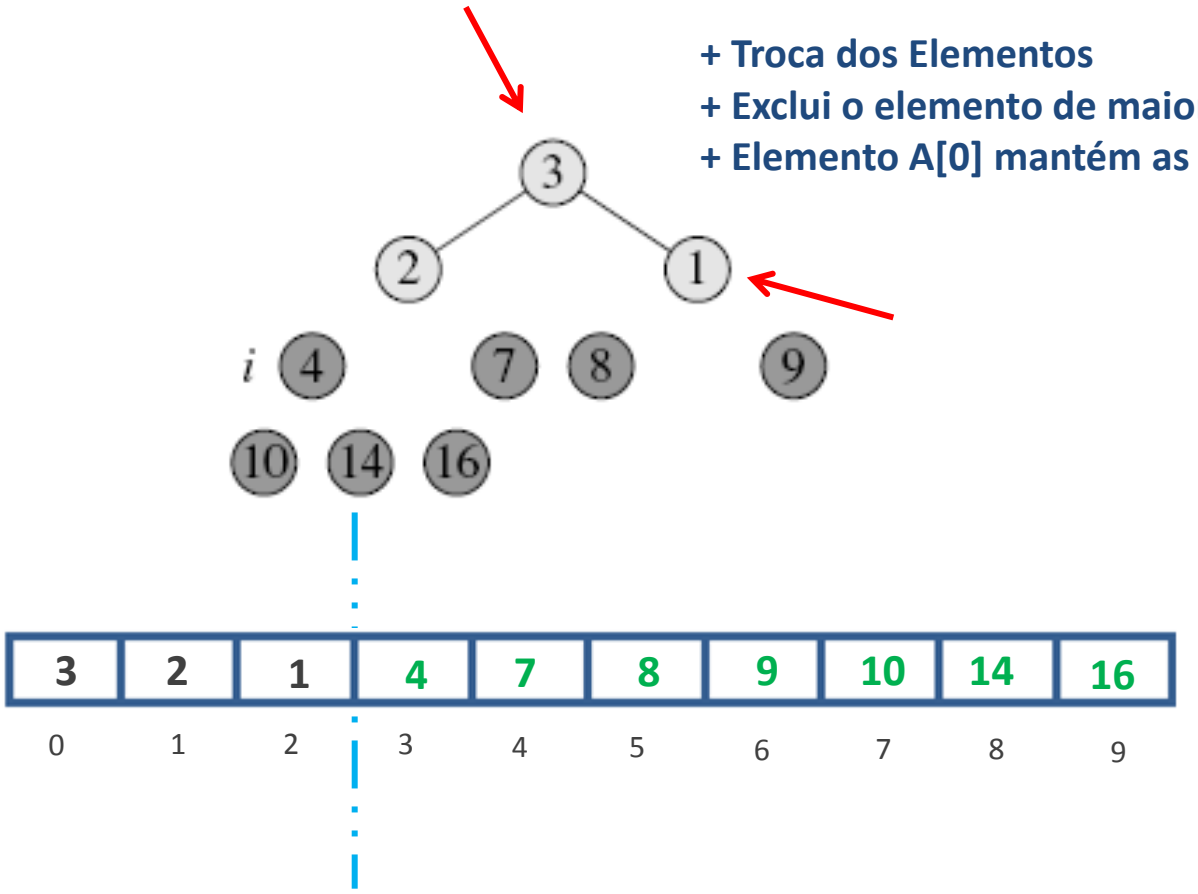
- Como ordenar o vetor abaixo com o Heap Sort?



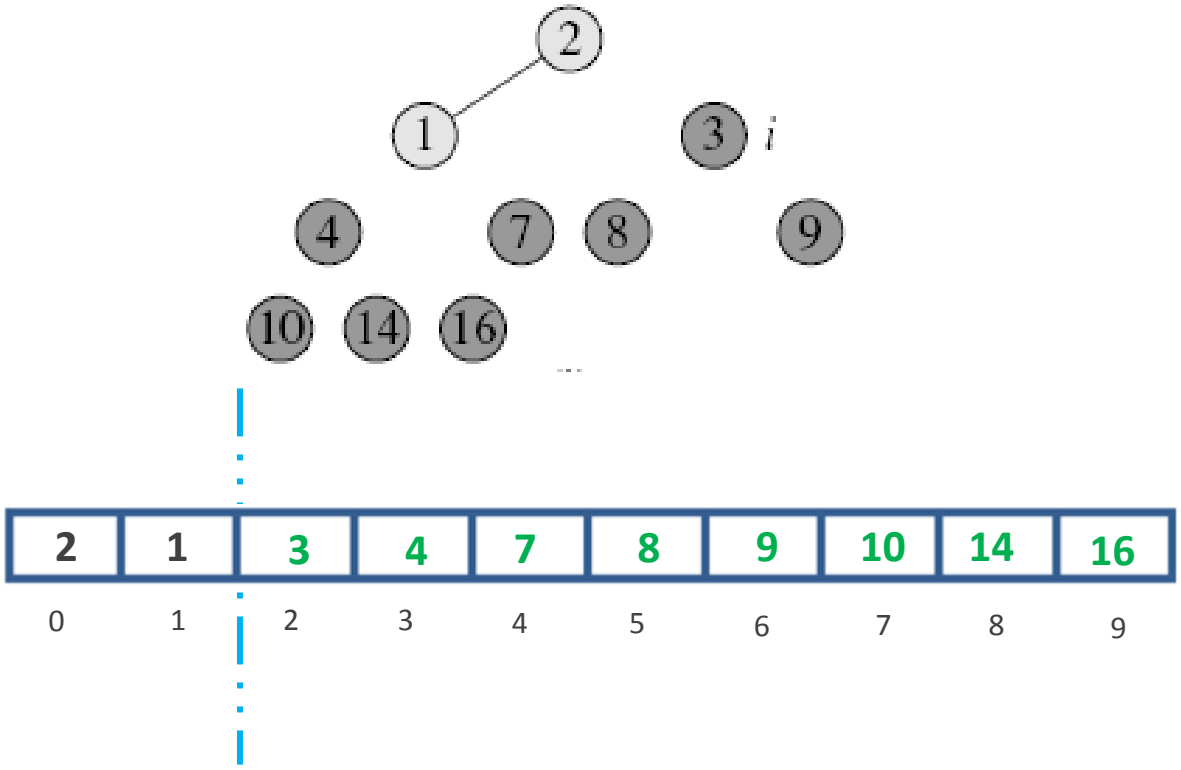
- Como ordenar o vetor abaixo com o Heap Sort?



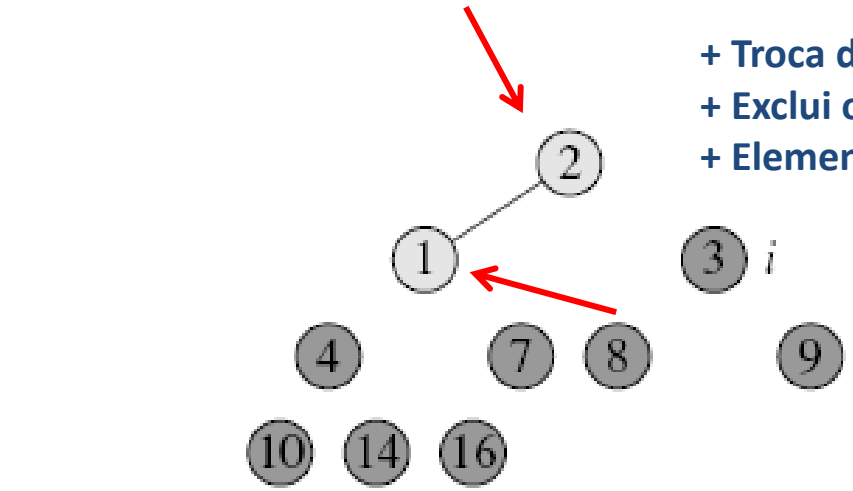
- Como ordenar o vetor abaixo com o Heap Sort?



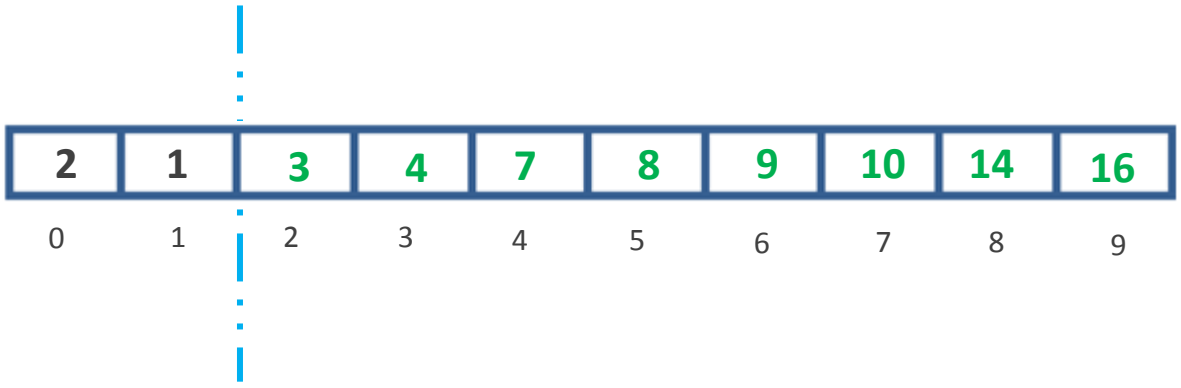
- Como ordenar o vetor abaixo com o Heap Sort?



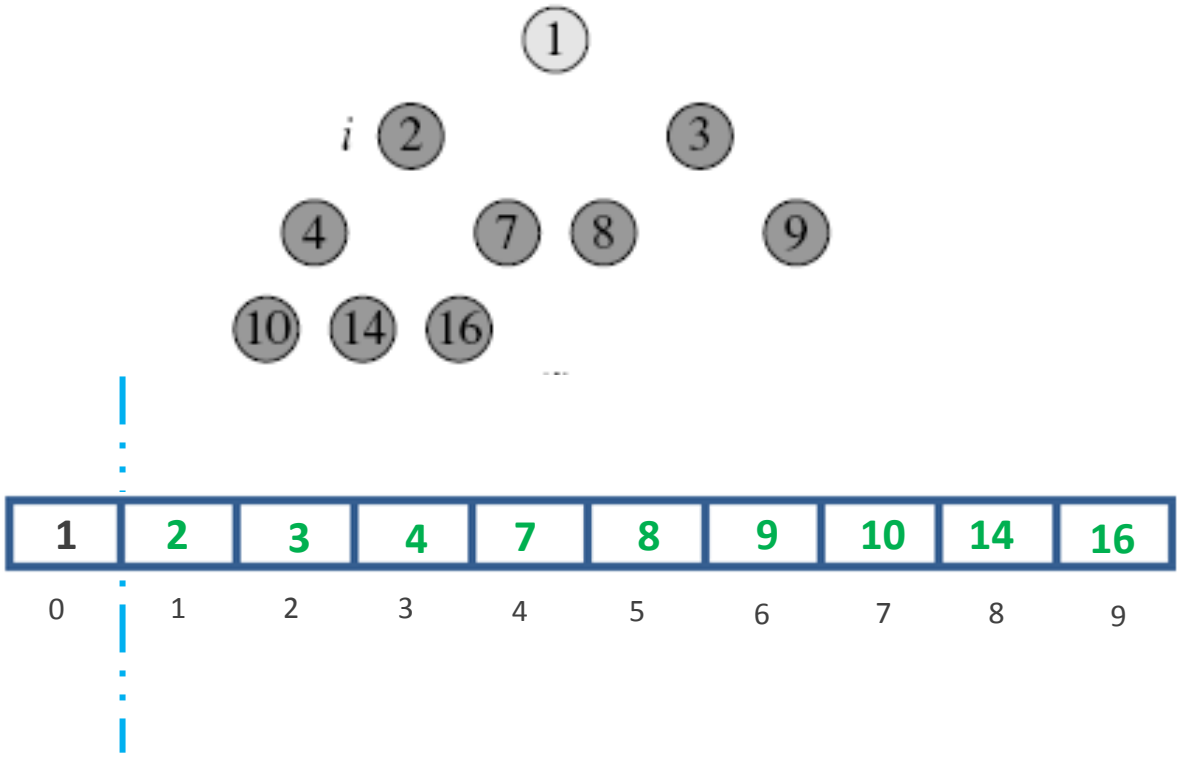
- Como ordenar o vetor abaixo com o Heap Sort?



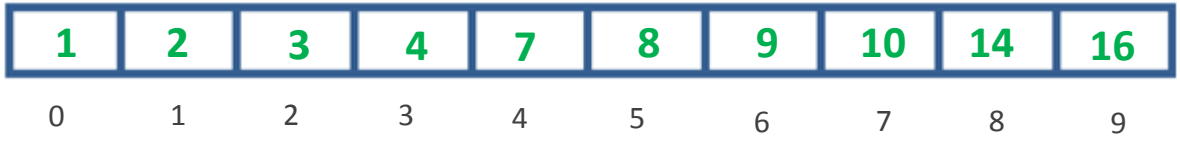
+ Troca dos Elementos
+ Exclui o elemento de maior valor do heap
+ Elemento A[0] mantém as características do heap



- Como ordenar o vetor abaixo com o Heap Sort?



- Como ordenar o vetor abaixo com o Heap Sort?



- **Merge Sort**
  - Esse algoritmo segue o paradigma divisão e conquista.
  - Funcionamento do merge sort:
    - **Divisão:** Divide a sequência de n elementos que deve ser ordenada em subsequências de n/2 elementos cada uma.
    - **Conquista:** Ordena as duas sequencias recursivamente. A recursão termina quando a sequência a ser ordenada tiver apenas um elemento.
    - **Combinação:** Intercala as duas subsequências ordenadas para produzir a resposta ordenada.

- **Merge Sort**
  - A operação mais importante desse algoritmo é conhecida como *merge*. Ela consiste da intercalação de duas sequencias já ordenadas.
  - A operação *merge* recebe duas subsequências já ordenadas e mescla todos os valores mantendo o resultado ordenado em um único subarranjo.
  - Merge sort é um algoritmo de ordenação estável.
  - Possui complexidade O(nlogn).

- Implementação do Merge Sort em java

```
1  private void mergesort(int[] valores, int inicio, int fim) {
2      if (inicio < fim) {
3          int meio = (inicio + fim) / 2;
4          mergesort(valores, inicio, meio);
5          mergesort(valores, meio + 1, fim);
6          merge(valores, inicio, meio, fim);
7      }
8  }
```

Implementação do Merge Sort em java

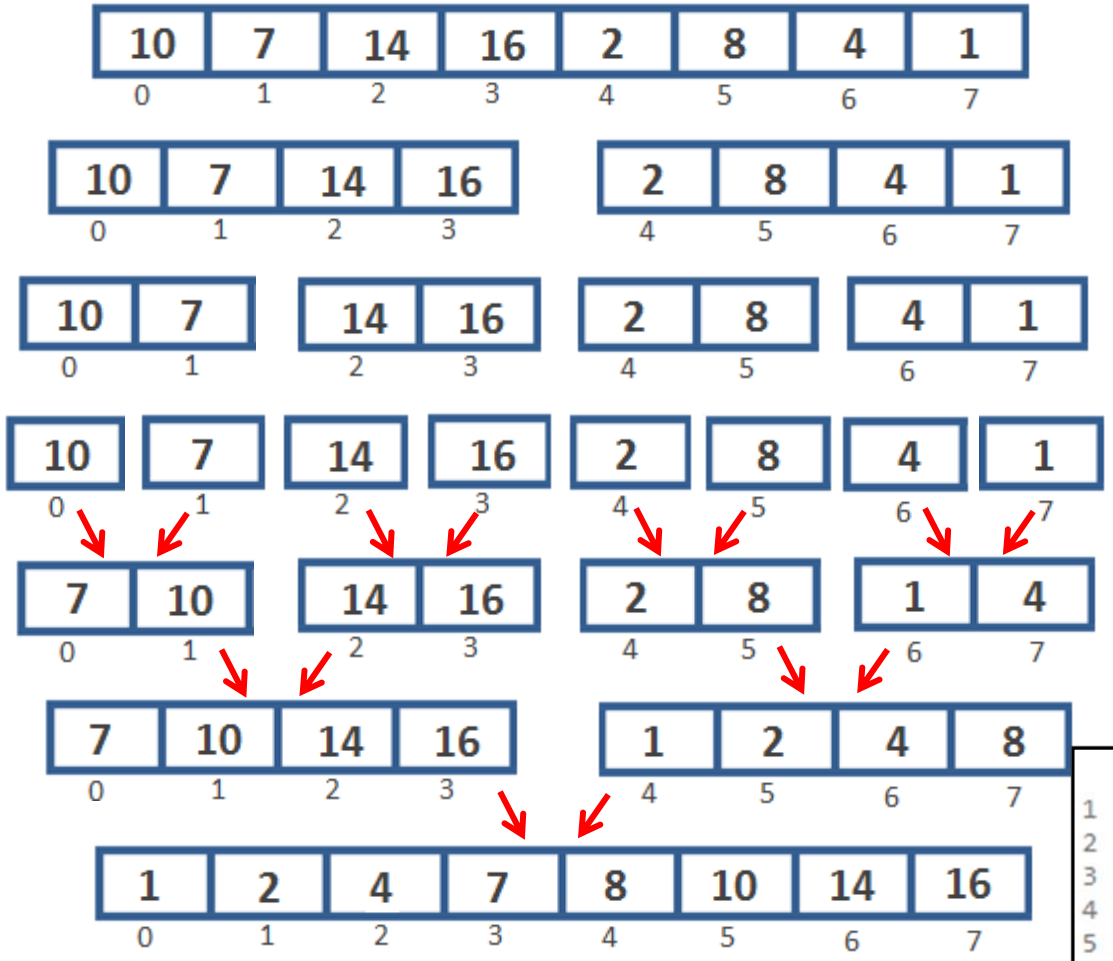
```java
private void merge(int[] valores, int inicio, int meio, int fim) {
    int[] vetorTemp = new int[valores.length];

    // Copia todos os elementos no vetor temporário
    for (int i = inicio; i <= fim; i++) {
        vetorTemp[i] = valores[i];
    }

    // Percorre os dois vetores lógicos e vai copiando o menor dos valores
    int i = inicio;
    int j = meio + 1;
    int k = inicio;
    while (i <= meio && j <= fim) {
        if (vetorTemp[i] <= vetorTemp[j]) {
            valores[k] = vetorTemp[i];
            i++;
        } else {
            valores[k] = vetorTemp[j];
            j++;
        }
        k++;
    }

    // Copia o restante dos elementos do vetor lógico que ainda possui elementos
    while (i <= meio) {
        valores[k] = vetorTemp[i];
        k++;
        i++;
    }
}
```

- Como ordenar o vetor abaixo com o Merge Sort?




```
    private void mergesort(
1               int[] valores, int inicio, int fim) {
2       if (inicio < fim) {
3           int meio = (inicio + fim) / 2;
4           mergesort(valores, inicio, meio);
5           mergesort(valores, meio + 1, fim);
6           merge(valores, inicio, meio, fim);
7       }
8   }
```

- **Quick Sort**
  - O quick sort é um método de ordenação por troca que aplica o paradigma de divisão e conquista.
  - Funcionamento:
    - Um elemento do arranjo será escolhido como **pivô**.
    - Em seguida o arranjo é dividido em 2 subarranjos:
      - Elementos menores ou iguais ao pivô.
      - Elementos maiores que o pivô
    - Os dois arranjos do passo anterior são ordenados recursivamente com o quick sort.
    - **Observação:** Caso base ocorre quando o vetor possui somente um elemento.

- **Quick Sort**
  - Esse algoritmo não requer uma estrutura complementar para sua execução, todo o processamento ocorre no mesmo arranjo de origem.
  - Algoritmo de ordenação não estável.
  - Complexidade no pior caso $O(n^2)$.
  - Complexidade no melhor caso O(nlogn).

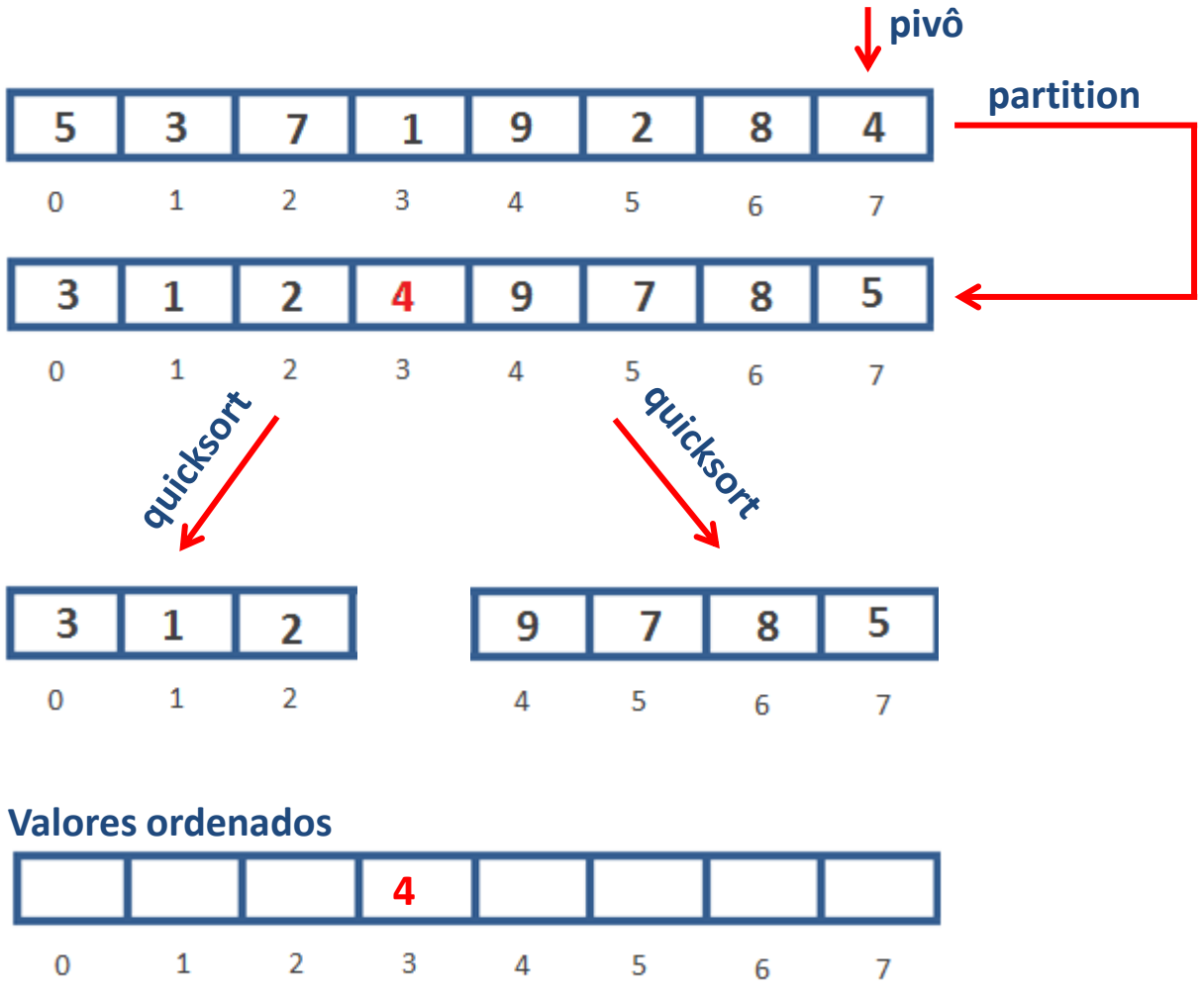
- Implementação do Quick Sort em java

```
1  private void quicksort(int[] valores, int i, int f) {
2      if (i < f) {
3          int q = partition(valores, i, f);
4          quicksort(valores, i, q - 1);
5          quicksort(valores, q + 1, f);
6      }
7  }
```

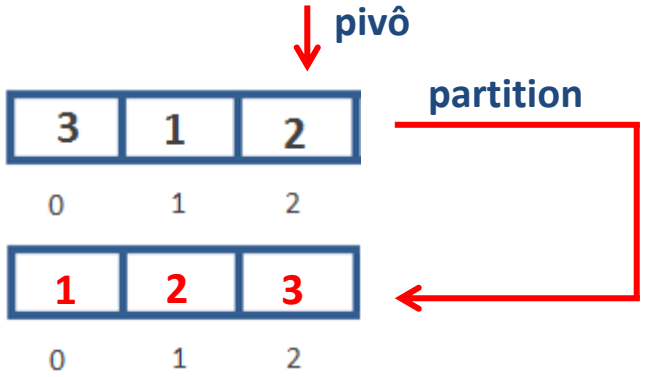
- Implementação do Quick Sort em java

```
1 private int partition(int[] valores, int i, int f) {
2     int pivo = valores[f];
3     int baixo = i - 1;
4     for (int alto = i; alto < f; alto++) {
5         if (valores[alto] <= pivo) {
6             troca(valores, ++baixo, alto);
7         }
8     }
9     troca(valores, ++baixo, f);
10
11    return baixo;
12 }
```

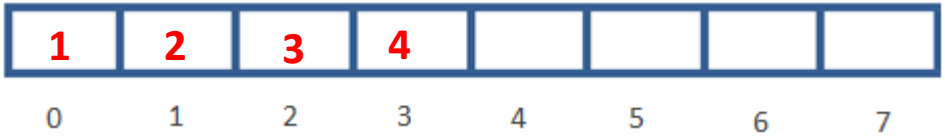
- Como ordenar o vetor abaixo com o Quick Sort?



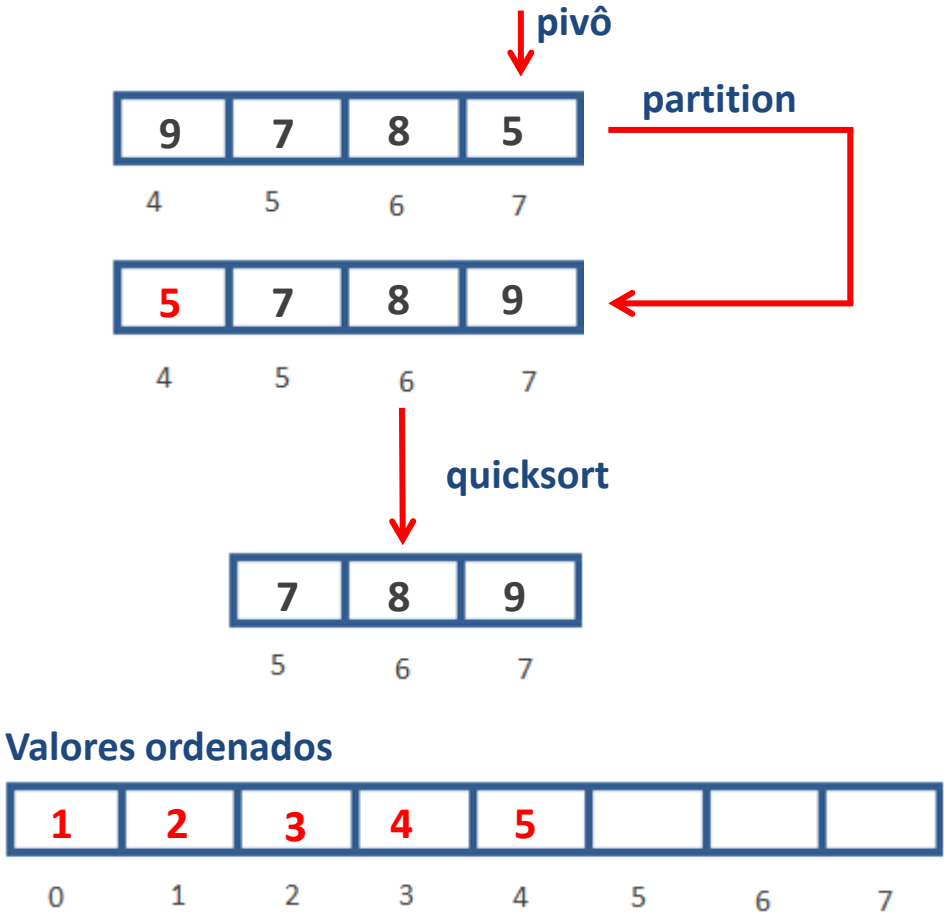
- Como ordenar o vetor abaixo com o Quick Sort?



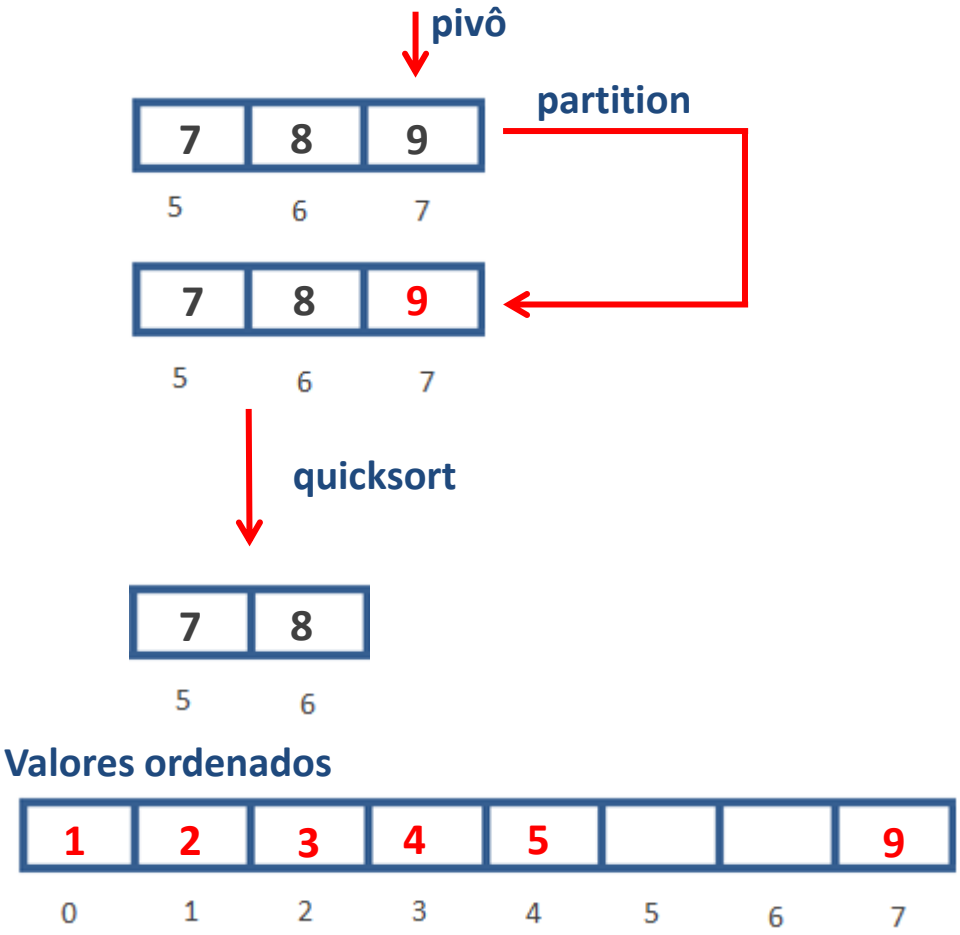
Valores ordenados



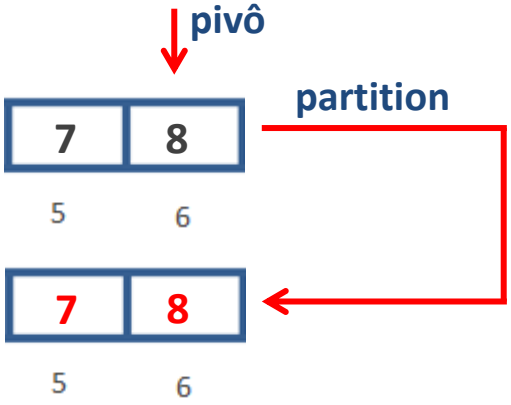
- Como ordenar o vetor abaixo com o Quick Sort?



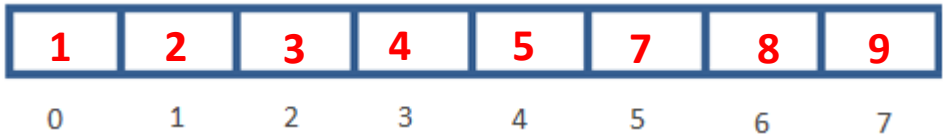
- Como ordenar o vetor abaixo com o Quick Sort?



- Como ordenar o vetor abaixo com o Quick Sort?



Valores ordenados



Algoritmos de Ordenação

ITnerante

- Tabela resumo

(-) eficientes

Algoritmos de Ordenação	Melhor Caso	Caso Médio	Pior Caso	Ordenação estável?
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Não
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Sim
Inserção Direta	$O(n)$	$O(n^2)$	$O(n^2)$	Sim
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Não
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Sim
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	Não

(+) eficientes

Algoritmos de Ordenação

- **Métodos de Ordenação**
  - **Ordenação por troca**
    - Bubble Sort
    - Quick Sort
  - **Ordenação por Inserção**
    - Inserção Direta
  - **Ordenação por Seleção**
    - Selection Sort
    - Heap Sort

Questão 24

ITnerante

[CESPE - 2010 - ABIN - Oficial Técnico de Inteligência]

A respeito dos métodos de ordenação, pesquisa e hashing, julgue os seguintes itens:

A estabilidade de um método de ordenação é importante quando o conjunto de dados já está parcialmente ordenado.

Certo Errado

Questão 25

ITnerante

[2011 - Cesgranrio - FINEP – Analista de Desenvolvimento de Sistemas]

Considerando-se a análise assintótica (Notação Big O), qual é a complexidade do caso médio do algoritmo de ordenação chamado de Ordenação por Inserção?

A) $O(n^2)$
B) $O(1)$
C) $O(n)$
D) $O(n \log n)$
E) $O(\log n)$

Questão 26

ITnerante

[2010 - Cesgranrio – BACEN – Analista]

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

Questão 27

ITnerante

[2009 - FCC - TRT - Analista 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.

Gabarito

24 – Errado

25 – A

26 – A

27 – D

PROVAS DE TI
TUDO PARA VOCÊ PASSAR

FIM

Rodrigo Adur
rodrigoadurti@gmail.com

ITnerante

TIMASTERS