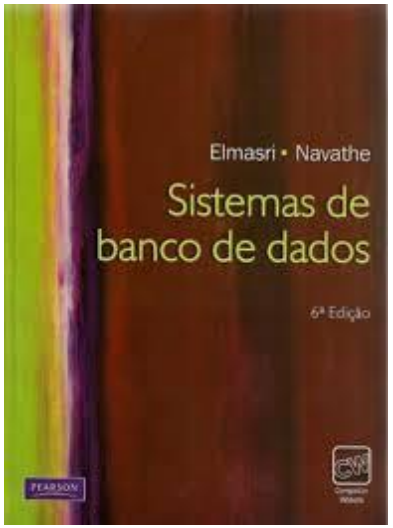
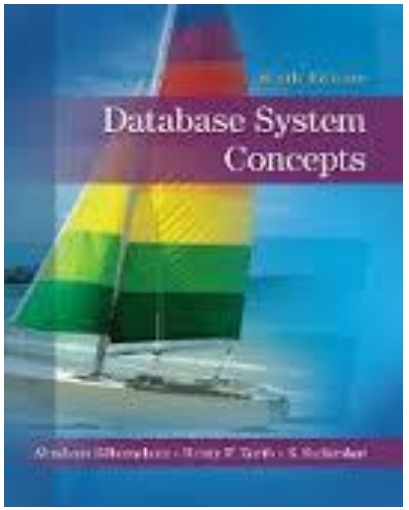
Unidade 13 – Algoritmos para Processamento de Consultas em Banco de Dados

Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP

Bibliografia



Sistemas de Banco de Dados
Elmasri / Navathe 6ª edição



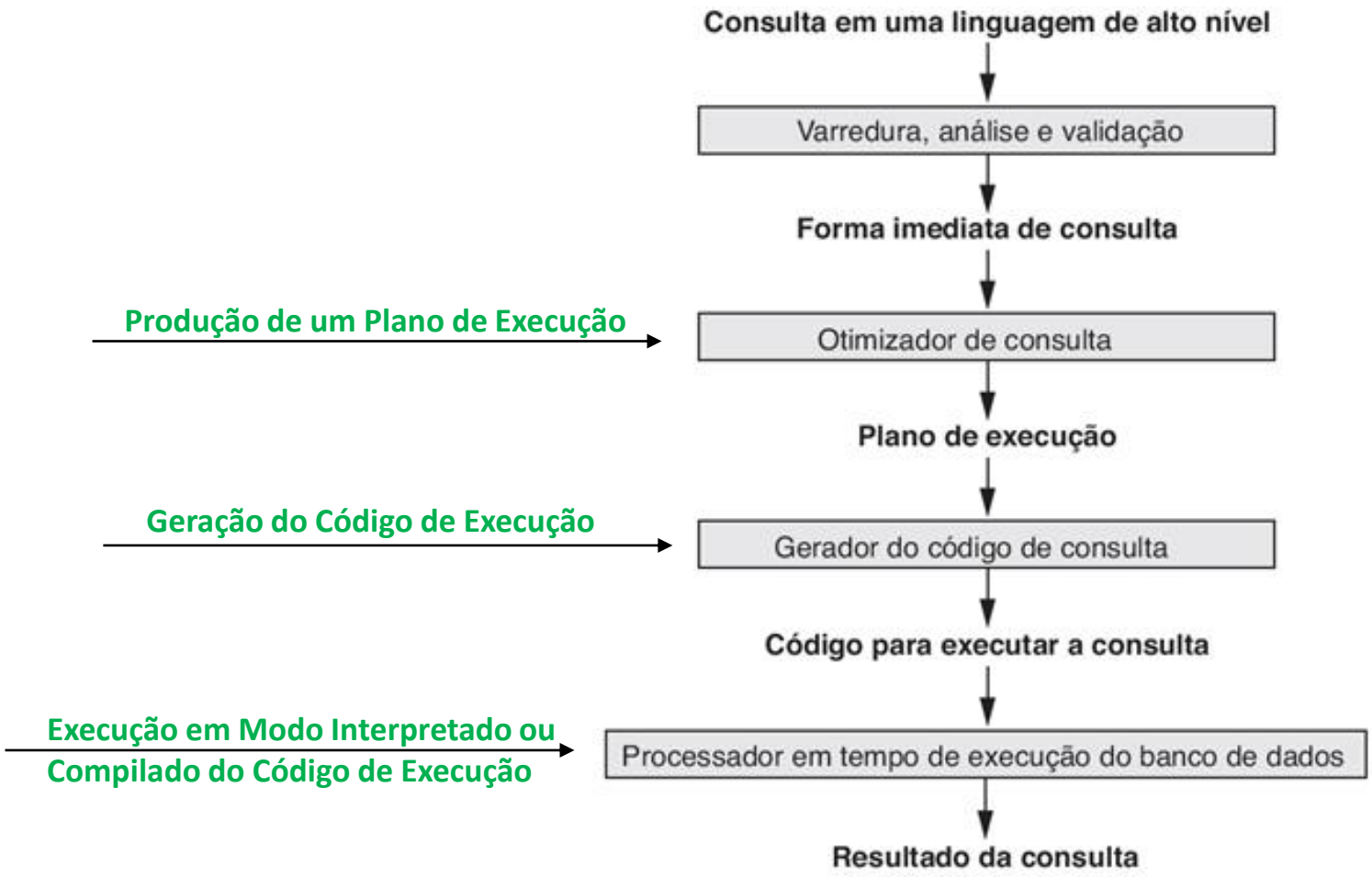
Sistema de Banco de Dados
Korth, Silberschatz – Sixth Editon

Como uma query em SQL é processada pelo Sistema Gerenciador de Banco de Dados ?

Processamento de Consultas

- Uma consulta no SGBD, expressa em **SQL**, precisa ser primeiramente lida, analisada e validada;
- A varredura consiste em identificar os tokens de consulta (**Análise Léxica**);
- A análise verifica a sintaxe da consulta para verificar se ela está formulada de acordo com as regras gramaticais da linguagem de consulta (**Análise Sintática**);
- A validação consiste em se verificar se os nomes de atributos e relações são válidos e semanticamente significativos no esquema do banco de dados.

Etapas do Processador de Consultas



Fonte: Elmasri,Navathe

Otimização de Consultas

- O SGBD precisa idealizar **uma estratégia de execução** (ou **plano de execução**) para recuperar os resultados da query com base nos arquivos de banco de dados;

- Uma query, em geral, costuma ter muitas estratégias de execução possíveis e o processo de escolha de uma dessas estratégias adequadas para a execução é conhecida por **Otimização de Consulta**.

- O termo **otimização**, na verdade, um nome errado, pois em alguns casos o plano de execução escolhido não é a estratégia ótima. As vezes, encontrar a estratégia ideal é muito demorado.

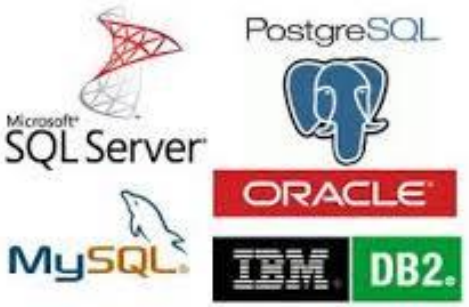
Por que encontrar o plano de execução ideal pode ser demorado ?

Plano de Execução

- A obtenção do plano ideal pode exigir informações detalhadas sobre como os arquivos do banco de dados são implementados e até mesmo sobre o seu conteúdo;
- Essas informações podem **não** estar disponíveis no catálogo do SGBD;
- Assim, o **planejamento de uma boa estratégia de execução** pode **não** ser, na verdade, o plano ótimo de execução.

Tradução de Consultas para a Álgebra Relacional

- Na prática, a linguagem **SQL** é usada na maioria dos SGBDR comerciais;
- Uma query **SQL** é inicialmente traduzida para uma expressão equivalente da Álgebra Relacional (representada por uma árvore de consulta) que é, então, **otimizada**.



Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

- Considere o seguinte esquema de banco de dados:

FUNCIONARIO

Pnome	Minicial	Unome	Cpf	Datanasc	Endereco	Sexo	Salario	Cpf_supervisor	Dnr

DEPARTAMENTO

Dnome	Dnumero	Cpf_gerente	Data_inicio_gerente

LOCALIZACAO_DEP

Dnumero	Dlocal

PROJETO

Projnome	Projnumero	Projlocal	Dnum

TRABALHA_EM

Fcpf	Pnr	Horas

DEPENDENTE

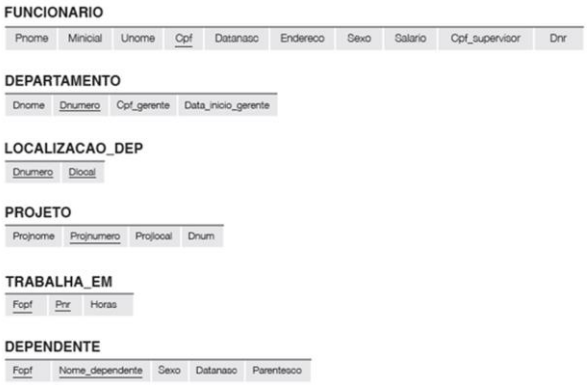
Fcpf	Nome_dependente	Sexo	Datanasc	Parentesco

Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

- Considere a seguinte consulta SQL no banco de dados

```
SELECT  Unome, Pnome
FROM    FUNCIONARIO
WHERE   Salario > ( SELECT  MAX (Salario)
                    FROM    FUNCIONARIO
                    WHERE   Dnr=5 );
```



Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

```
SELECT   Unome, Pnome
FROM     FUNCIONARIO
WHERE    Salario > ( SELECT   MAX (Salario)
                     FROM     FUNCIONARIO
                     WHERE    Dnr=5 );
```

- Essa consulta recupera os nomes dos funcionários (de qualquer departamento da empresa) que ganham um salário maior que o maior salário.

Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

```
SELECT   Unome, Pnome
FROM     FUNCIONARIO
WHERE    Salario > ( SELECT   MAX (Salario)
                     FROM     FUNCIONARIO
                     WHERE    Dnr=5 );
```

Bloco mais interno

```
( SELECT  MAX (Salario)
  FROM    FUNCIONARIO
  WHERE   Dnr=5 )
```

- O bloco mais **interno** recupera o salário mais alto do departamento 5;

Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

```
SELECT   Unome, Pnome
FROM     FUNCIONARIO
WHERE    Salario > ( SELECT   MAX (Salario)
                     FROM     FUNCIONARIO
                     WHERE    Dnr=5 );
```

Bloco mais externo

```
SELECT   Unome, Pnome
FROM     FUNCIONARIO
WHERE    Salario > c
```

- No bloco mais **externo**, c representa o resultado retornado do bloco interno.

Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

Bloco mais interno

```
( SELECT  MAX (Salario)
  FROM    FUNCIONARIO
  WHERE   Dnr=5 )
```

- O bloco mais **interno** poderia ser traduzido para a seguinte expressão da Álgebra Relacional:

$$\Im_{\text{MAX Salario}}(\sigma_{\text{Dnr}=5}(\text{FUNCIONARIO}))$$

Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

Bloco mais externo

```
SELECT   Unome, Pnome
FROM     FUNCIONARIO
WHERE    Salario > c
```

- O bloco mais **externo** poderia ser traduzido para a seguinte expressão da Álgebra Relacional:

$$\pi_{Unome,Pnome}(\sigma_{Salario>c}(FUNCIONARIO))$$

Fonte: Navathe/Elmasri

Exemplo – Consulta em Álgebra Relacional

```
SELECT  Unome, Pnome
FROM    FUNCIONARIO
WHERE   Salario > ( SELECT  MAX (Salario)
                    FROM    FUNCIONARIO
                    WHERE   Dnr=5 );
```

- O **Otimizador de Consulta** irá escolher um Plano de Execução para cada bloco de consulta;

Bloco mais interno

```
( SELECT  MAX (Salario)
  FROM    FUNCIONARIO
  WHERE   Dnr=5 )
```

Bloco mais externo

```
SELECT  Unome, Pnome
FROM    FUNCIONARIO
WHERE   Salario > c
```

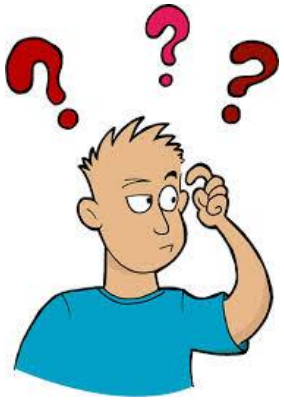
Fonte: Navathe/Elmasri

Algoritmos para Ordenação

- A **intercalação** (**Sorting**) é um dos principais algoritmos utilizados no processamento de queries;
- Por exemplo, sempre que uma consulta SQL especifica uma cláusula **ORDER BY**, o resultado da consulta precisa ser **ORDENADO**;
- A **intercalação** também é um componente chave nos algoritmos ordenação-intercalação (SORT-MERGE) usados em operações de JUNÇÃO e outras operações tais como UNIÃO e INTERSECÇÃO;
- **Intercalação** também é empregada em algoritmos de eliminação de duplicatas para a operação PROJEÇÃO (quando uma consulta SQL especifica a cláusula **DISTINCT**).

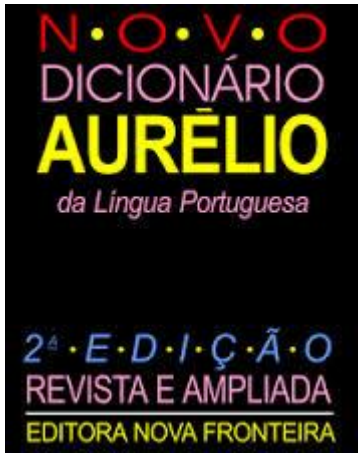
Algoritmos para Ordenação

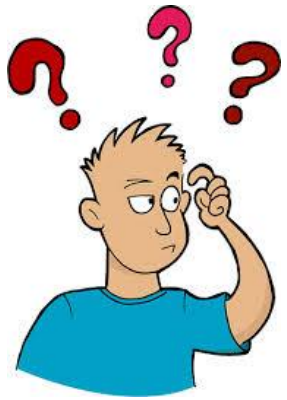
Como seria encontrar um nome em lista telefônica desordenada ?



Algoritmos para Ordenação

Ou encontrar um nome em um dicionário que não estivesse ordenado ?

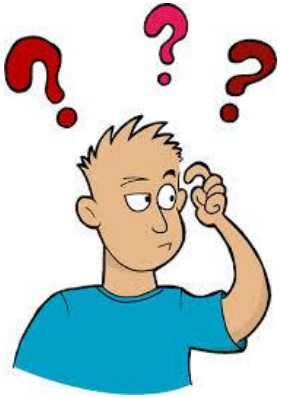




Ordenação

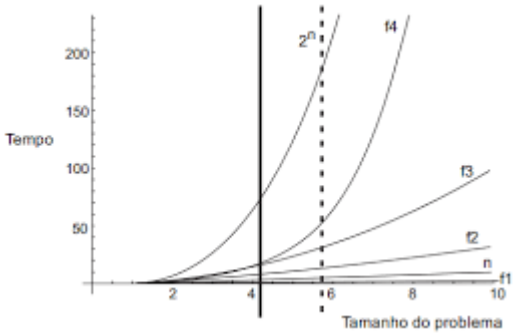
- A conveniência de se usar dados ordenados é inquestionável em um consulta ao **Banco de Dados**.
- Felizmente, existem bons algoritmos de ordenação.

Como se mede a eficiência de um algoritmo de ordenação



Algoritmos de Ordenação

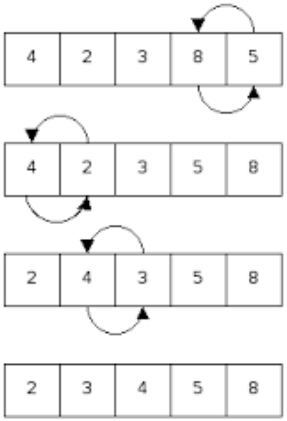
- A **eficiência** em geral é medida pelo número de comparações e pelo número de movimentações de dados;
- Funções de Complexidade (BIG O) são usualmente empregadas para se avaliar o desempenho dos algoritmos.



Problema de Ordenação

- Permutar (rearranjar) os elementos de um array a[0..n-1] de tal modo que eles fiquem em **ordem crescente**, ou seja, de tal forma que se tenha:

$$a[0] \leq a[1] \leq \ldots \leq a[n-1]$$



Algoritmo Bubble-Sort

- Uma **bolha** nada mais é que um par de números ordenados;
- Exemplo (**2**,**7**) e (**7**,**4**);
- A primeira destas bolhas está ordenada crescentemente, mas a segunda não;
- Toda vez que se encontra uma bolha desordenada, deve-se invertê-la, transformando (**7**,**4**) em (**4**,**7**);
- As bolhas são assim chamadas pois sobem pela sequência da mesma forma como bolhas de gás na água;
- Complexidade: **$O(n^2)$**, com algumas variações dependendo da forma dos dados de entrada.

Sequência a ser ordenada: 4, 2, 12, 7, 4, 1

(4	2)	12	7	4	1
2	(4	12)	7	4	1
2	4	(12	7)	4	1
2	4	7	(12	4)	1
2	4	7	4	(12	1)
2	4	7	4	1	**12**

(2	4)	7	4	1	12
2	(4	7)	4	1	12
2	4	(7	4)	1	12
2	4	4	(7	1)	12
2	4	4	1	(7	12)
2	4	4	1	**7**	**12**

Algoritmo Bubble-Sort

```
void Bolha (int tam, int vet[]) {
     int  passo, i, aux;

 // Passos de Ordenação
for (passo=1 ; passo < tam ; passo++){

          // Passos de Troca
          for(i=1 ; i < tam; i++) {
               if (vet[i-1] > vet[i]){
                    aux      = vet[i-1];
                    vet[i-1] = vet[i];
                    vet[i]   = aux;
               }
          }
     }
}
```

Algoritmo Bubble-Sort Implementação

```
package maua;

import java.util.Arrays;

public class Bubble_01 {

        public static void main (String [] args ) {

                int[] lista = { 9, 7 , 4, 3, 1 } ;

                lista = bubble(lista);

                System.out.println (Arrays.toString(lista) ) ;
}
```

Algoritmo Bubble-Sort Implementação

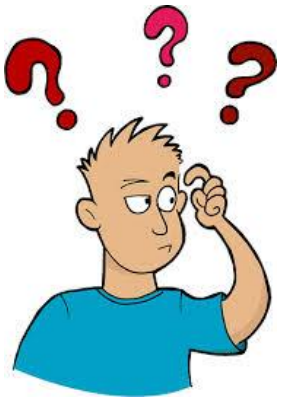
```
public static int[] bubble(int[] lista) {

        int aux;
        int n = lista.length;

        for (int i = 1; i < n ; i++ ) {

                for (int j = 1; j < n ; j ++)

                        if (lista[j-1]  > lista[j] )  {
                                aux = lista[j-1];
                                lista[j-1] = lista[j];
                                lista[j] = aux;
                        }

        }

        return lista;

    }

}
```

Bubble Sort com Trace

Algoritmo Bubble-Sort Implementação

```
public static int[] bubble (int[] lista) {

int aux;

     for(int i = 1 ; i < lista.length ; i++) {

          System.out.println ("\n*******PASSSO: " + i  + "   ******");
          System.out.println ("\n Lista antes de aplicar o PASSO: " +
                    Arrays.toString(lista) );
          for (int j=1; j < lista.length;j++)
                    if (lista[j-1] > lista[j] ) {
                              System.out.println("Troca de " + lista[j-1] +
                                        " com " + lista[j]);
                              aux = lista[j-1];
                              lista[j-1]=lista[j];
                              lista[j] = aux;
                    }
                    else System.out.println("Não trocou  " + lista[j-1] +
                              " com " + lista[j]);
          }
          return lista;

    }
  }
```

Quantas comparações e trocas são feitas no algoritmo Bubble Sort ?



Análise de Complexidade – Bubble Sort

- O número de comparações é o mesmo em cada caso (melhor, pior e médio) e igual ao número total de iterações do **for** interno;
- O número de trocas, no pior caso, ocorre quando o array estiver em ordem inversa;
- Ordem de complexidade: **$O(n^2)$** .

Exercício

- Ordenar a sequência **4, 2, 12, 1 pelo método Bolha.**
- **Mostrar todas as** TROCAS EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

******* PASSO: 1 ******

- Lista antes de aplicar o PASSO: [4, 2, 12, 1]
- Troca de 4 com 2
- Não trocou 4 com 12
- Troca de 12 com 1
- Lista após o PASSO 1: [2,4,1,12]

Exercício

- Ordenar a sequência **4, 2, 12, 1 pelo método Bolha.**
- **Mostrar todas as** TROCAS EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

******* PASSO: 2 ******

- Lista antes de aplicar o PASSO: [2, 4, 1, 12]
- Não trocou 2 com 4
- Troca de 4 com 1
- Não trocou 4 com 12
- Lista após o PASSO 2: [2,1,4,12]

Exercício

- Ordenar a sequência **4, 2, 12, 1 pelo método Bolha.**
- **Mostrar todas as** TROCAS EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

```
******* PASSO: 3  ******

Lista antes de aplicar o PASSO: [2, 1, 4, 12]
Troca de 2 com 1
Não trocou  2 com 4
Não trocou  4 com 12
Lista após o PASSO 3:  [1,2,4,12]
```

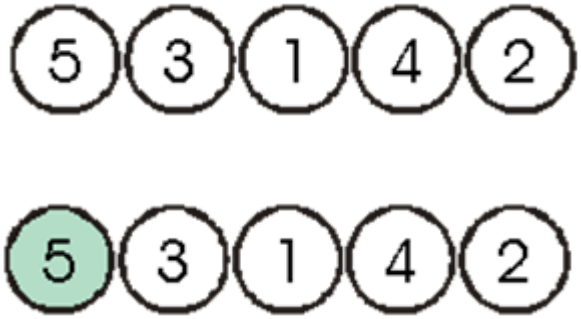
Exercício

- Alterar o algoritmo Bubble_Sort para classificar o array em ordem **DECRESCENTE**.

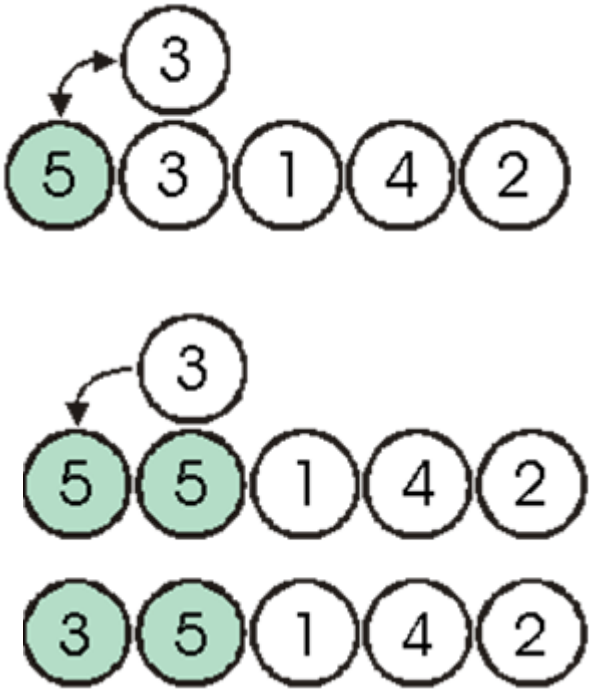
Algoritmo Inserção Direta

- Usado por exemplo, para colocar em ordem um baralho de cartas;
- Tem a mesma ordem de complexidade do método Bubble-Sort;
- Ordena um array utilizando um sub-array ordenado localizado em seu início, e a cada novo passo, acrescenta a este sub-array mais um elemento, até atingir o último elemento do array fazendo com que ele se torne ordenado.

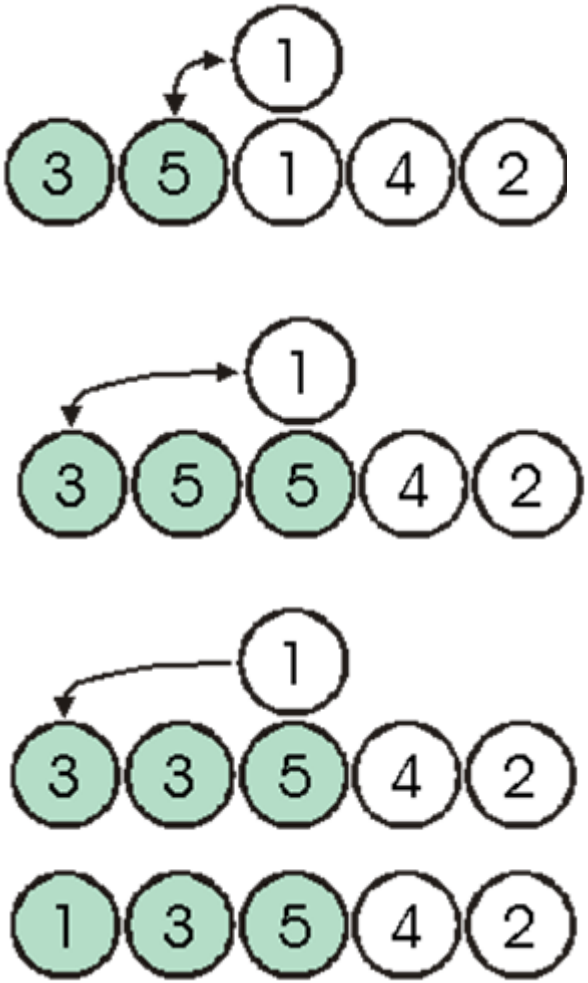
Método de Ordenação Inserção Direta



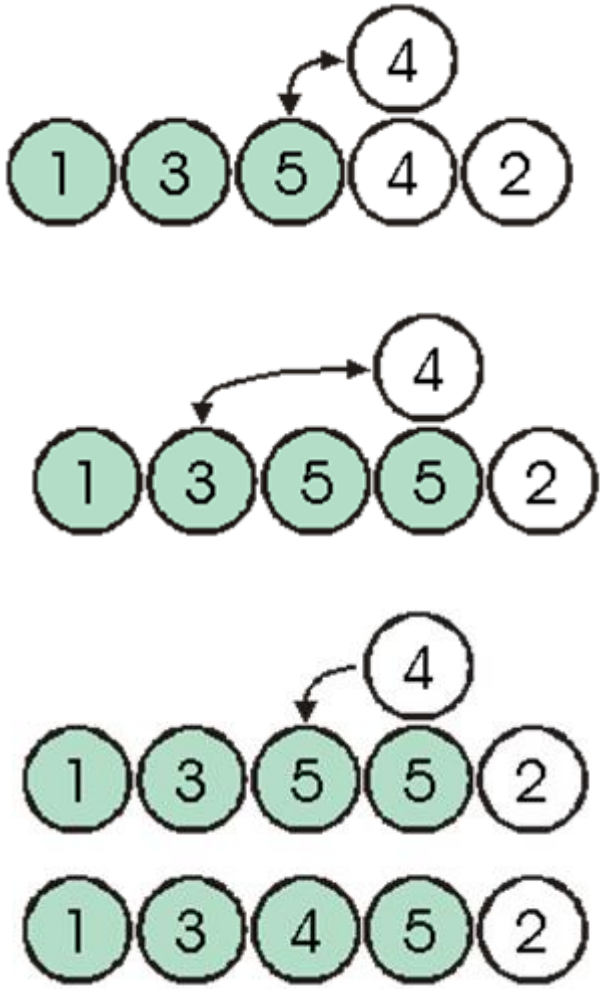
Método de Ordenação Inserção Direta



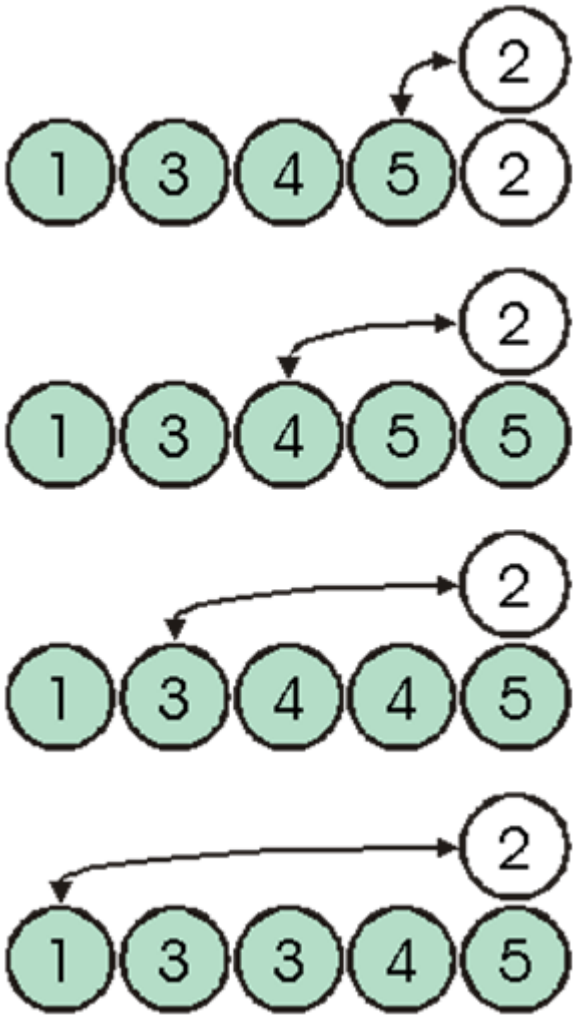
Método de Ordenação Inserção Direta



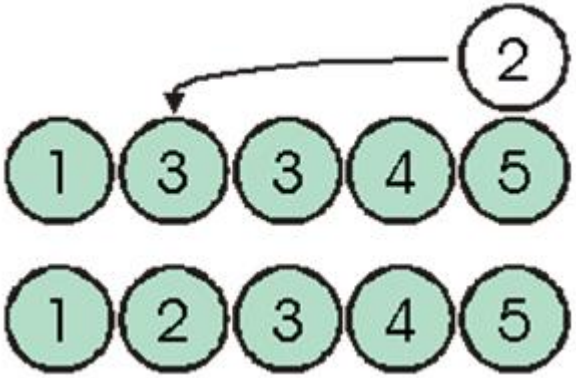
Método de Ordenação Inserção Direta



Método de Ordenação Inserção Direta



Método de Ordenação Inserção Direta



Array Ordenado

Algoritmo – Inserção Direta

```
package maua;

import java.util.Arrays;

public class SortInsertion {

        public static void main (String[] args ) {

                int[] lista = { 2,1,8,5,0} ;

                lista = sortInsertion(lista);

                System.out.println(Arrays.toString(lista));


}
```

Algoritmo – Inserção Direta

```
public static int[] sortInsertion(int[] lista)   {
        int i, j, aux;
        for (i = 1; i < lista.length; i++)  {
                aux = lista[i];
                j = i;
                while (j > 0 && lista[j-1] > aux) {
                        lista[j] = lista[j-1];
                        j--;
                }
                lista[j] = aux;
        }
        return lista;
}
```

Inserção Direta com Trace

Inserção Direta com Trace

```
package maua;

import java.util.Arrays;

public class Insertion {

        public static void main (String [] args ) {

                int[] lista = { 4, 2, 12, 1  } ;

                insertionSort(lista);

                System.out.println ("Lista Ordenada:   " +

                        Arrays.toString(lista) ) ;
        }
```

Inserção Direta com Trace

```
public static void insertionSort(int[] lista)   {

            for (int i = 1; i < lista.length; i++) {

            System.out.println("\n" +
                      "          ******* PASSO: " + i + " *******");

            System.out.println("Lista antes de aplicar o PASSO: " +
                     Arrays.toString(lista));

            int a = lista[i]; // guardo o elemento a ser inserido
```

```
        // passos para encontrar o elemento deve ser inserido
        for (int j = i - 1; j >= 0 && lista[j] > a; j--)  {
           lista[j + 1] = lista[j];  // move ate encontrar posicao correta
            System.out.println("Movimentações dentro do passo: " +
                    Arrays.toString(lista));
             lista[j] = a;  // insere na posicao correta
        }
        System.out.println("Lista dentro do PASSO: "   +
                    Arrays.toString(lista) + "\n");
    }
  }
}
```

Exercício

- Ordenar a sequência **4, 2, 12, 1 pelo método** Inserção Direta.
- **Mostrar todas as** MOVIMENTAÇÕES EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

```
******* PASSO: 1 ******
Lista antes de aplicar o PASSO: [4, 2, 12, 1]
Movimentações dentro do PASSO: [4, 4, 12, 1]
Lista dentro do PASSO: [2, 4, 12, 1]
```

Exercício

- Ordenar a sequência **4, 2, 12, 1 pelo método** Inserção Direta.
- **Mostrar todas as** MOVIMENTAÇÕES EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

******* PASSO: 2 ******
Lista antes de aplicar o PASSO: [2, 4, 12, 1]
Lista dentro do PASSO: [2, 4, 12, 1]

Exercício

- Ordenar a sequência **4, 2, 12, 1 pelo método** Inserção Direta.
- **Mostrar todas as** MOVIMENTAÇÕES EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

```
******* PASSO: 3 ******
Lista antes de aplicar o PASSO: [2, 4, 12, 1]
Movimentações dentro do PASSO: [2, 4, 12, 12]
Movimentações dentro do PASSO: [2, 4, 4, 12]
Movimentações dentro do PASSO: [2, 2, 4, 12]
Lista dentro do PASSO: [1, 2, 4, 12]
```

Lista ordenada: [1, 2, 4, 12]

Análise de Complexidade – Insertion Sort

- Uma vantagem do algoritmo por inserção é que ele ordena o vetor somente quando realmente é necessário;
- O melhor caso é quando os dados já estão em ordem. O pior caso é quando os dados estão em ordem inversa;
- Ordem de complexidade: $O(n^2)$

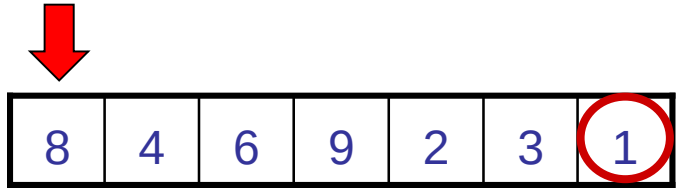
Algoritmo por Seleção

- Encontre o **menor** elemento no array;
- Troque-o com o elemento da primeira posição;
- Encontre o segundo **menor** elemento no array;
- Troque-o com o elemento da segunda posição;
- Continue até que o array esteja ordenado.

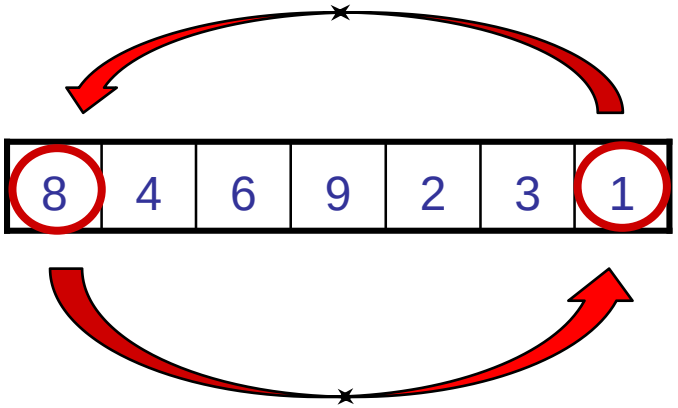
Algoritmo por Seleção - Pseudocódigo

```
Selectionsort(array[], n)
        for (i=0; i < n-1 ; i++)

                selecione o menor elemento
                entre array[i],...,array[n-1];

                troque-o com array[i];
```

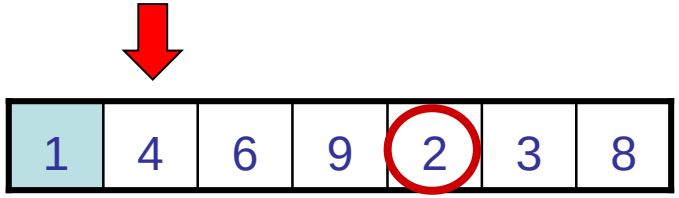
Algoritmo por Seleção



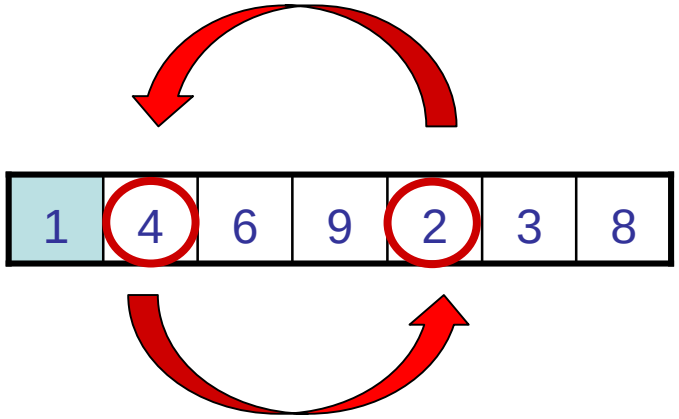
1

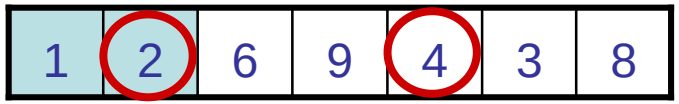


Algoritmo por Seleção

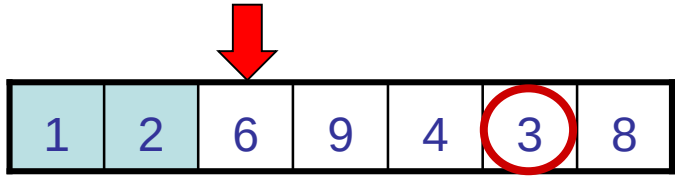


2

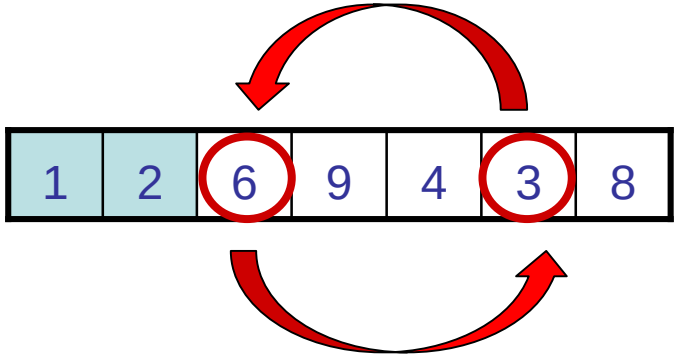


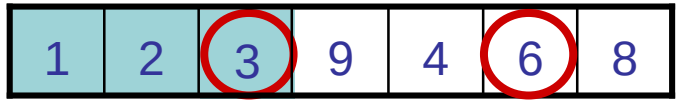


Algoritmo por Seleção

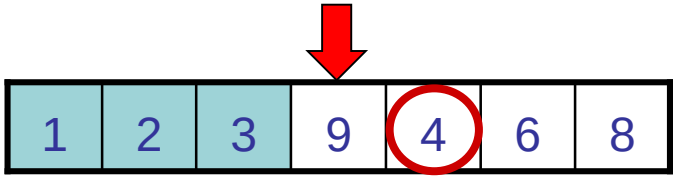


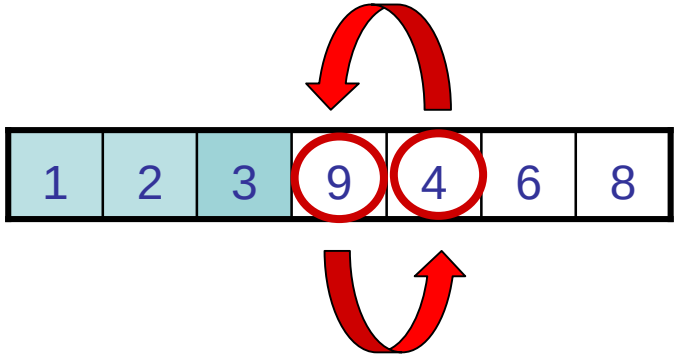
3

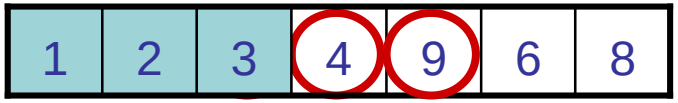




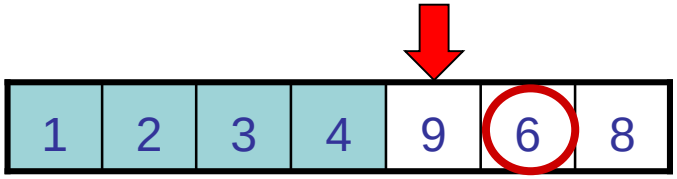
Algoritmo por Seleção



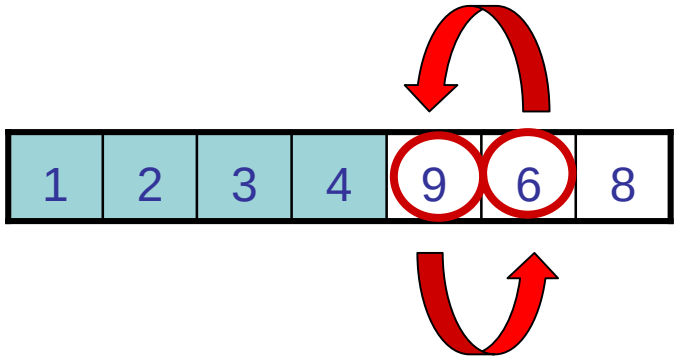




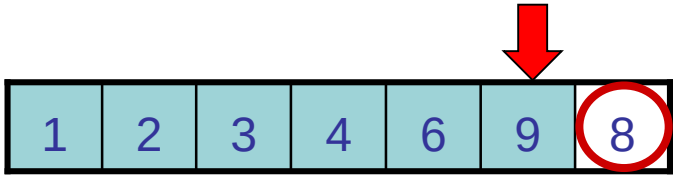
Algoritmo por Seleção

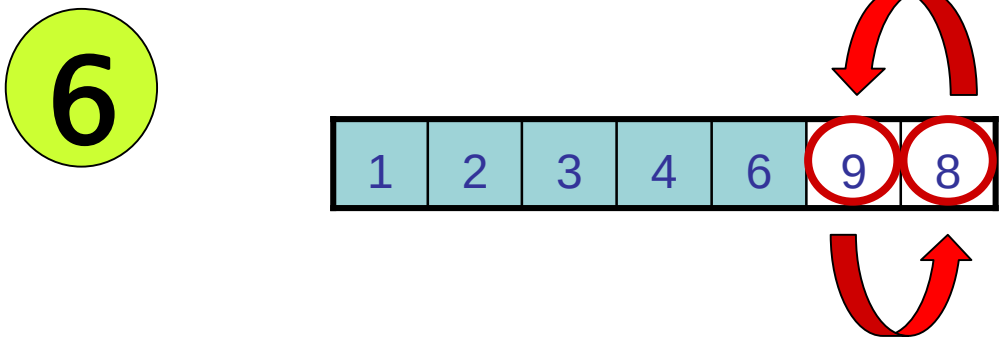


5



Algoritmo por Seleção





Algoritmo por Seleção - Implementação

```
Selectionsort(array[], n) {
    for (i=0; i < n-1 ; i++) {

        selecione o menor elemento
        entre array[i],...,array[n-1];

        troque-o com array[i];
    }
}
```

Algoritmo por Seleção - Implementação

```
package maua;

import java.util.Arrays;

public class Selection {
        public static void main(String[] args) {


                int[] lista = new int[] { 5, 2, 1, 9, 7 } ;

                lista = selection_sort(lista);

                System.out.println("\n\n\nLista ordenada:   " +
                        Arrays.toString(lista) );
}
```

Algoritmo por Seleção - Implementação

```
public static int[] selection_sort(int[] lista) {

        int menor, indiceMenor;

        for (int i = 0 ;  i < lista.length - 1 ; i++) {

                menor = lista[i];

                indiceMenor = i ;

                for (int j = i+1 ;  j < lista.length ; j++) {

                        if (lista[j] < menor) {

                                menor = lista[j];
                                indiceMenor = j;

                        }
                }
                lista[indiceMenor] = lista[i];

                lista[i]=menor;
        }
        return lista;
    }


}
```

Selection com Trace

Selection com Trace

```
public static int[] selection_sort(int[] lista) {

        //armazena o menor valor e o índice do menor valor
        int menor, indiceMenor;

        for (int i = 0 ;  i < lista.length - 1 ; i++) {

                System.out.println ("\n\n\n*******PASSO: " +
                        (i+1)  + "   ******");
                System.out.println ("Lista antes de aplicar o PASSO: " +
                         Arrays.toString(lista) );

                menor = lista[i];
                indiceMenor = i ;

// compara com os outros valores do array
```

```
        for (int j = i+1 ;  j < lista.length ; j++) {

                if (lista[j] < menor) {

                        menor = lista[j];
                        indiceMenor = j;

                }
        }

        //swap

        lista[indiceMenor] = lista[i];
        lista[i]=menor;
        System.out.println("Movimentações dentro do PASSO: " +
                 Arrays.toString(lista) );

        }
        return lista;
    }


}
```

Análise de Complexidade – Selection Sort

- O laço mais externo executa n-1 vezes.
- O laço mais interno executa de i + 1 até n-1.
- Ordem de complexidade: **$O(n^2)$**

Exercício – Selection Sort

- Ordenar a sequência **5, 2, 1, 9, 7 pelo método SelectionSort.**
- **Mostrar todas as** MOVIMENTAÇÕES EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

*******PASSO: 1 ******
Lista antes de aplicar o PASSO: [5, 2, 1, 9, 7]
Movimentações dentro do PASSO: [1, 2, 5, 9, 7]

Exercício – Selection Sort

- Ordenar a sequência **5, 2, 1, 9, 7 pelo método SelectionSort.**
- **Mostrar todas as** MOVIMENTAÇÕES EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

```
*******PASSO: 2   ******
Lista antes de aplicar o PASSO: [1, 2, 5, 9, 7]
Movimentações dentro do PASSO: [1, 2, 5, 9, 7]
```

Exercício – Selection Sort

- Ordenar a sequência **5, 2, 1, 9, 7 pelo método SelectionSort.**
- **Mostrar todas as** MOVIMENTAÇÕES EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

```
*******PASSO: 3    ******
Lista antes de aplicar o PASSO: [1, 2, 5, 9, 7]
Movimentações dentro do PASSO: [1, 2, 5, 9, 7]
```

Exercício – Selection Sort

- Ordenar a sequência **5, 2, 1, 9, 7 pelo método SelectionSort.**
- **Mostrar todas as** MOVIMENTAÇÕES EFETUADAS NA SEQUÊNCIA EM QUE ACONTECEM.
- Apresente também, passo a passo, a evolução do conteúdo do array.

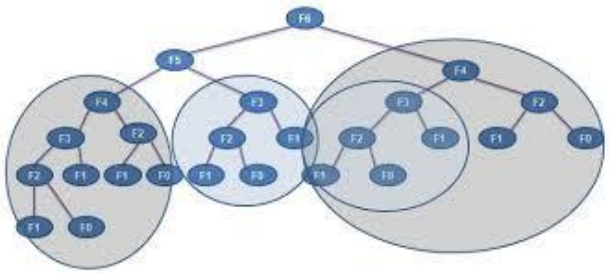
```
*******PASSO: 4    ******
Lista antes de aplicar o PASSO: [1, 2, 5, 9, 7]
Movimentações dentro do PASSO: [1, 2, 5, 7, 9]
```

Algoritmo Sort-Merge

- São largamente empregados em sistemas gerenciadores de banco de dados, pois são adequados para grandes arquivos de registros armazenados em disco que não cabem inteiramente na memória principal;
- Empregam a técnica **divide-and-conquer**;
- A técnica **divide-and-conquer** é uma das mais conhecidas técnicas de projeto de algoritmos;

Técnica Divisão e Conquista

- Dado um problema com uma entrada grande, quebra-se a entrada em porções menores (**DIVISÃO**).
- Resolve-se cada porção separadamente (**CONQUISTA**).
- Combina-se os resultados.

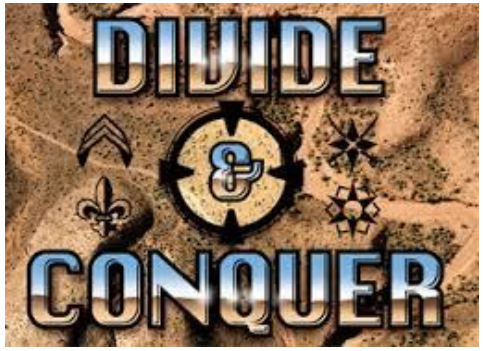


Algoritmos Divisão e Conquista

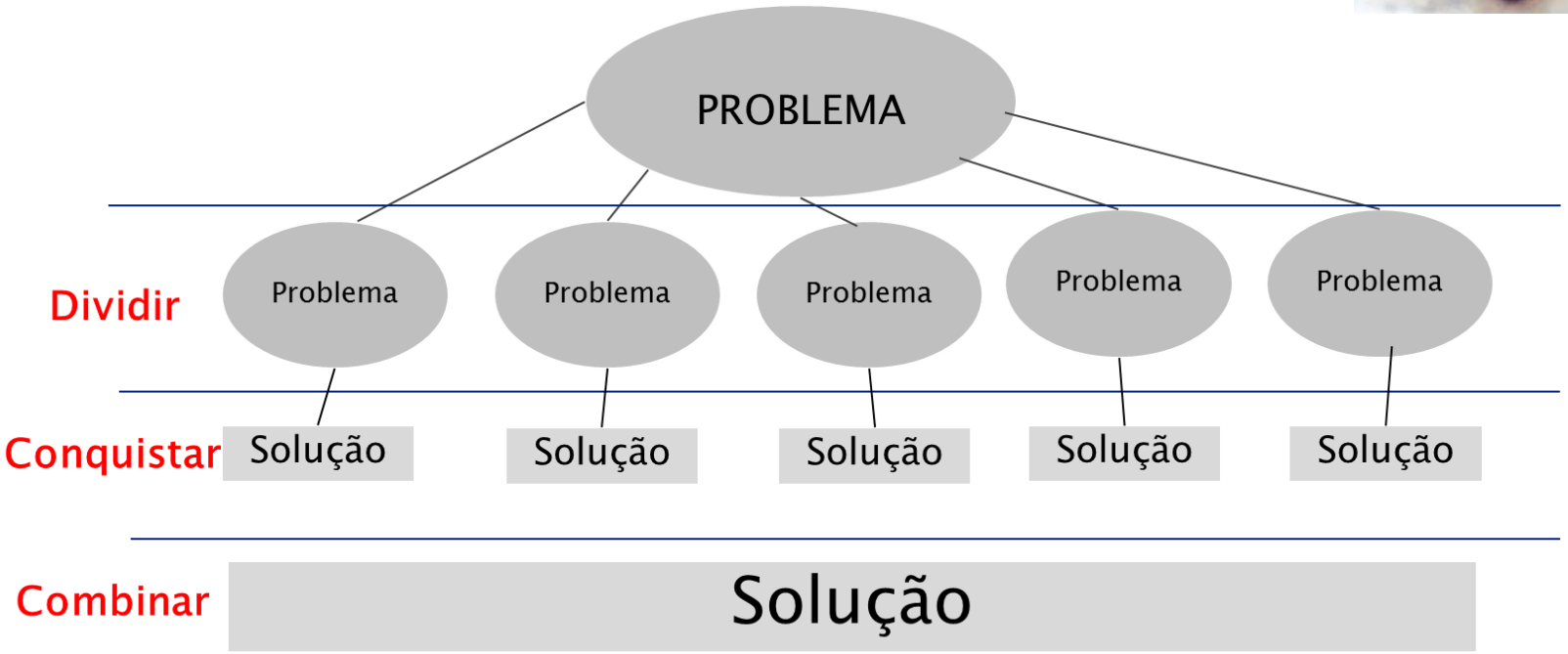
- São, em geral, recursivos;
- Requer um número menor de acessos à memória;
- São altamente paralelizáveis. Se existirem vários processadores disponíveis, a estratégia propicia eficiência.

Quando empregar Divisão e Conquista ?

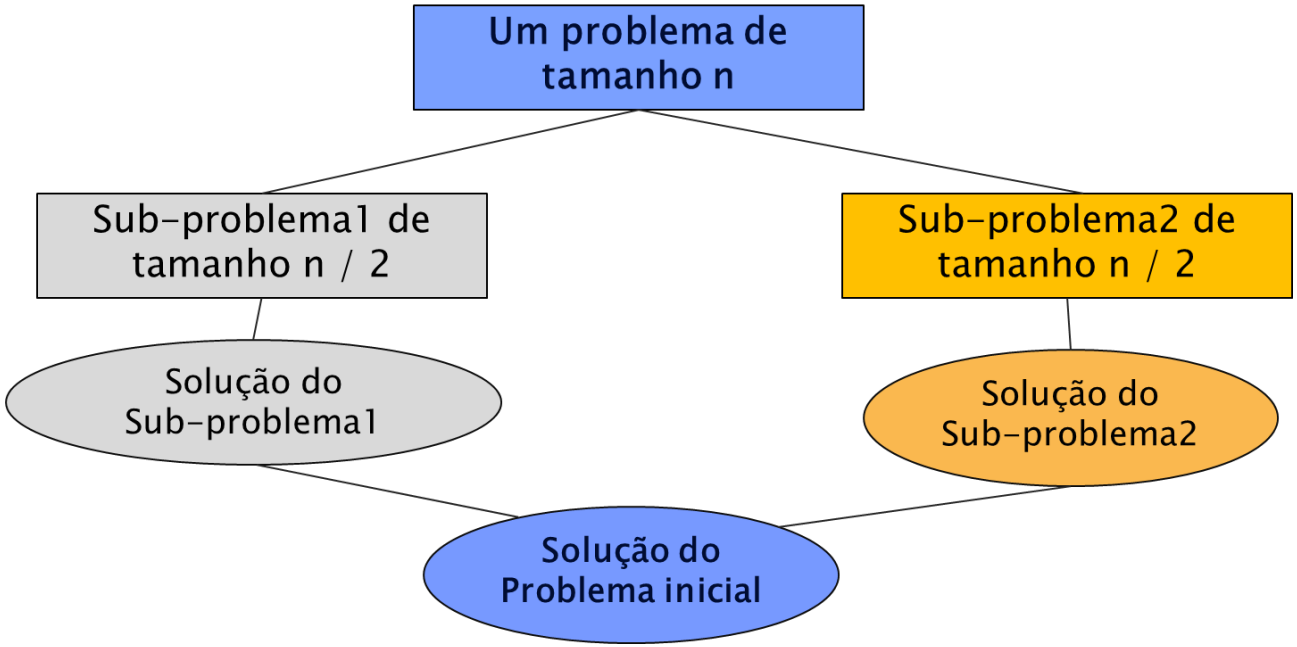
- Quando for possível decompor-se a instância em sub-instâncias.
- Problemas onde um grupo de operações são correlacionadas ou repetidas.



Passos do Algoritmo Divisão e Conquista



Técnica Divisão e Conquista



Técnica Divisão e Conquista
Algoritmo Genérico

```
DivisaoeConquista(x)

        if x é pequeno ou simples do
                return resolver(x)
        else
                decompor x em conjuntos menores x0, x1, … xn
        for i ← 0 to  n  do
                yi ← DivisãoeConquista(xi)
                i ← i +1
        combinar yi's

        return y
```

Técnica Divisão e Conquista

Exemplo 1

O problema consiste em encontrar o maior elemento de um array a [1 .. n]

3	8	1	7	6	2	9	4

Exemplo 1

Solução Força Bruta

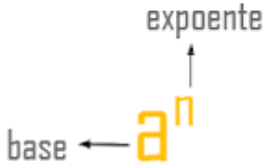
```
Maximo (V[1..n])
    max ← A[1]
    for i ← 2 to n do
        if A[j] > max then
            max ← V[i]
    return max
```

Exemplo 1 – Solução Divisão e Conquista

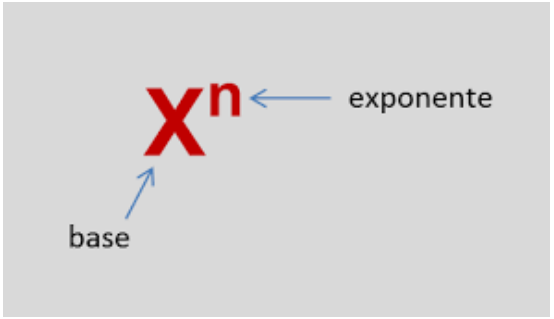
```
Maximo(V[x..y])
if (x - y)  ≤ 1 then
      return max(A[x], A[y])
else
      m ← (x+y)/2
      v1 ← Maximo(V[x..m])
      v2 ← Maximo(V[m+1..y])
return max(v1, v2)
```

Técnica Divisão e Conquista

Exemplo 2



O problema consiste em computar x^n.



Exemplo 2

Solução Força Bruta

```
Pow(a, n)
    p ← a

    for i ← 2 to n do
        p ← p × a
    return p
```

Exemplo 2

Solução Divisão e Conquista

```
Pow(a, n)

      if n=0 then
            return 1
      if n é par then
            return Pow(a, n/2) × Pow(a, n/2)
      else
            return Pow(a,(n-1)/2) × Pow(a, (n-1)/2) × a
```

Técnica Divisão e Conquista

Exemplo 3

O problema consiste em computar a soma dos valores numéricos de uma dada lista

Exemplo 3 – Força Bruta

```
package maua;
// Soma de uma lista de números com forca bruta
public class Soma_Lista_ForcaBruta {

        static int[] Lista = {10,20,30,40};
        static int n = Lista.length;

        public static void main(String[] args ) {

            int Soma=0;
            for (int i=0; i <n ; i++ )

               Soma = Soma + Lista[i];
            System.out.println ("Soma dos Elementos da Lista = " + Soma)
        }
   }
```

Exemplo 3 – Divisão e Conquista

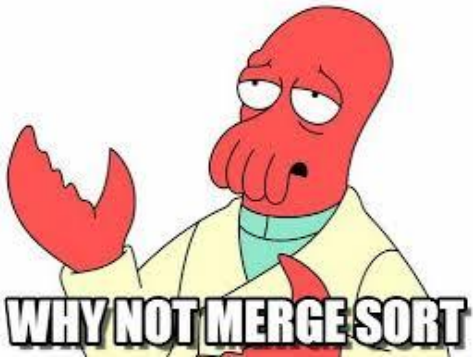
```
package exercicios;
// Soma de uma lista de numeros com Divisao e Conquista

        public class Soma_Lista_DivConq {

          static int[] Lista = {10,20,30,40};
          static int n = Lista.length;

          public static void main(String[] args ) {
            int i=0, j = n - 1;

            System.out.println ("Soma dos Elementos da Lista =  " + Soma(i,j)) ;
}
```

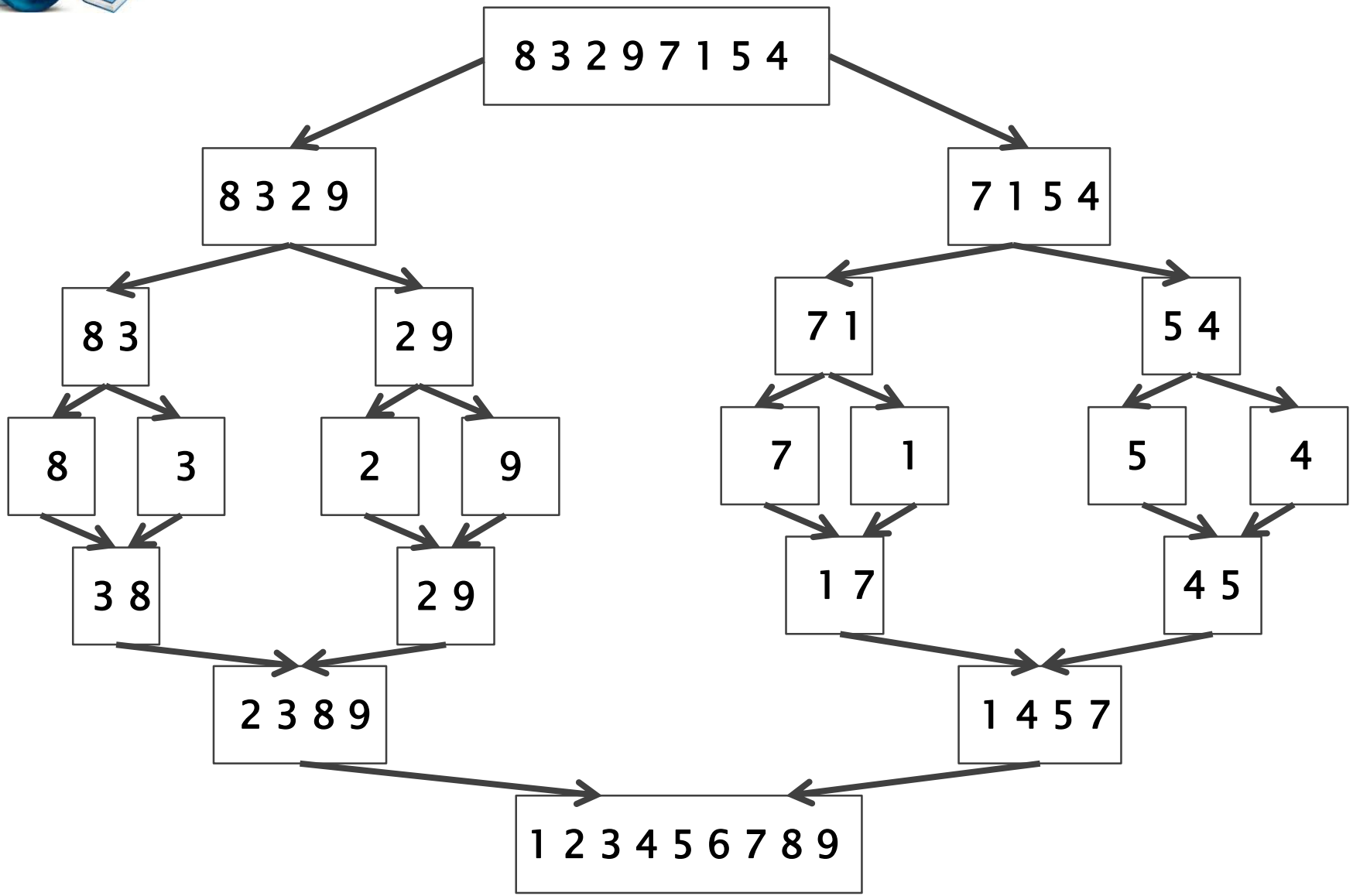
Exemplo 3 – Divisão e Conquista

```
public static int Soma(int x, int y) {

        int m, S1, S2;
        if (x == y) return Lista[x];
        if (y-x <= 1) return (Lista[x] + Lista[y]);
        else {
                m = (x+y)/2;
                S1 = Soma(x,m);
                S2 = Soma(m+1,y);
                return (S1 + S2);
        }
    }
}
```

SortMerge

- ✓ É um perfeito exemplo de uma aplicação da técnica Divide-and-Conquer.
- ✓ O algoritmo classifica um dado array A[0..n-1] dividindo-o em duas metades A[0 . . ⌊n/2⌋-1] e A[⌊n/2⌋ . . n-1], sorteando cada uma delas de forma recursiva, e em seguida efetua um merge das partes sorteadas.




8 3 2 9 7 1 5 4
8 3 2 9
7 1 5 4
8 3
2 9
7 1
5 4
8
3
2
9
7
1
5
4
3 8
2 9
1 7
4 5
2 3 8 9
1 4 5 7
1 2 3 4 5 6 7 8 9

SortMerge - Implementação

```
package maua;

public class MergeSort {

  public static int[] vetor = {8,3,2,9,7,1,5,4};

  static int n = vetor.length;

  public static void main(String[] args) {

        Sort(0,n-1);
        for (int i=0;i<n;i++ )
              System.out.println("vetor[" + i + "] = " + vetor[i]);
}
```

SortMerge - Implementação

```
public static void Sort(int inicio, int fim) {

        if (inicio < fim) {

                int meio = (inicio + fim) / 2;

                Sort(inicio, meio);
                Sort(meio + 1, fim);
                Merge(inicio, meio, fim);
        }
}
```

SortMerge - Implementação

```
public static void Merge(int inicio, int meio, int fim) {

        int n = fim - inicio + 1;

        int[] temp = new int[n];
        int tamanho = temp.length;

        for (int posicao = 0; posicao < tamanho; posicao++) {
                temp[posicao] = vetor[inicio + posicao];
        }
```

SortMerge - Implementação

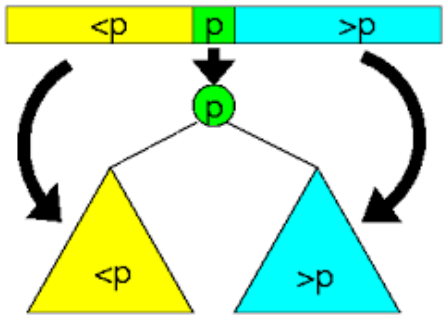
```
int i = 0;
int j = meio - inicio + 1;

 for (int posicao = 0; posicao < tamanho; posicao++) {

   if (j <= n- 1 )
    if ( i <= meio - inicio)
       if (temp[i] < temp[j] )
              vetor[inicio + posicao] = temp[i++];
       else vetor[inicio + posicao] = temp[j++];
    else vetor[inicio + posicao] = temp[j++];
   else vetor[inicio + posicao] = temp[i++];
 }

}

}
```

SortMerge - Eficiência

- Prova-se que a ordem de complexidade do algoritmo SortMerge é:

$$O(n) = n * \ln(n)$$

Quicksort

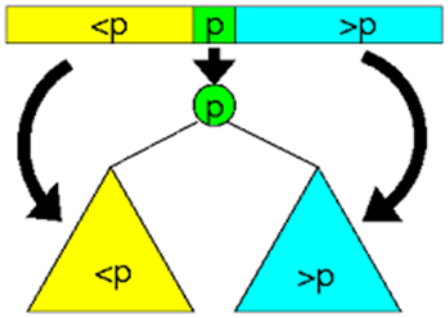


Quicksort

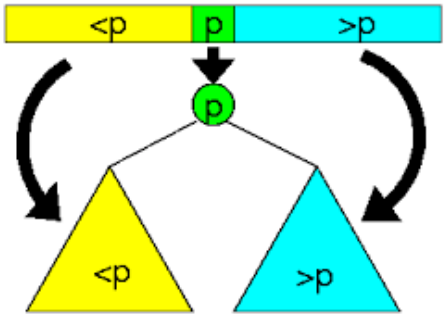
- O Quicksort adota a estratégia de **divisão e conquista**.
- A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores".
- Em seguida o Quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada.

Passos do Quicksort

- Escolha um elemento da lista, denominado **pivô**;
- Rearranje a lista de forma que todos os elementos anteriores ao **pivô** sejam menores que ele, e todos os elementos posteriores ao **pivô** sejam maiores que ele.
- Ao fim do processo o **pivô** estará em sua posição final e haverá duas sublistas não ordenadas.



Passos do Quicksort



- Essa operação é denominada partição;
- Recursivamente ordene a sublista dos elementos menores e a sublista dos elementos maiores;
- A base da recursão são as listas de tamanho zero ou um, que estão sempre ordenadas.
- O processo é finito, pois a cada iteração pelo menos um elemento é posto em sua posição final e não será mais manipulado na iteração seguinte.

Quicksort- Eficiência

- Prova-se que a ordem de complexidade do algoritmo QuickSort é:
  - O(n) = n * ln (n) (melhor caso)
  - O(n) = n^2 no pior caso