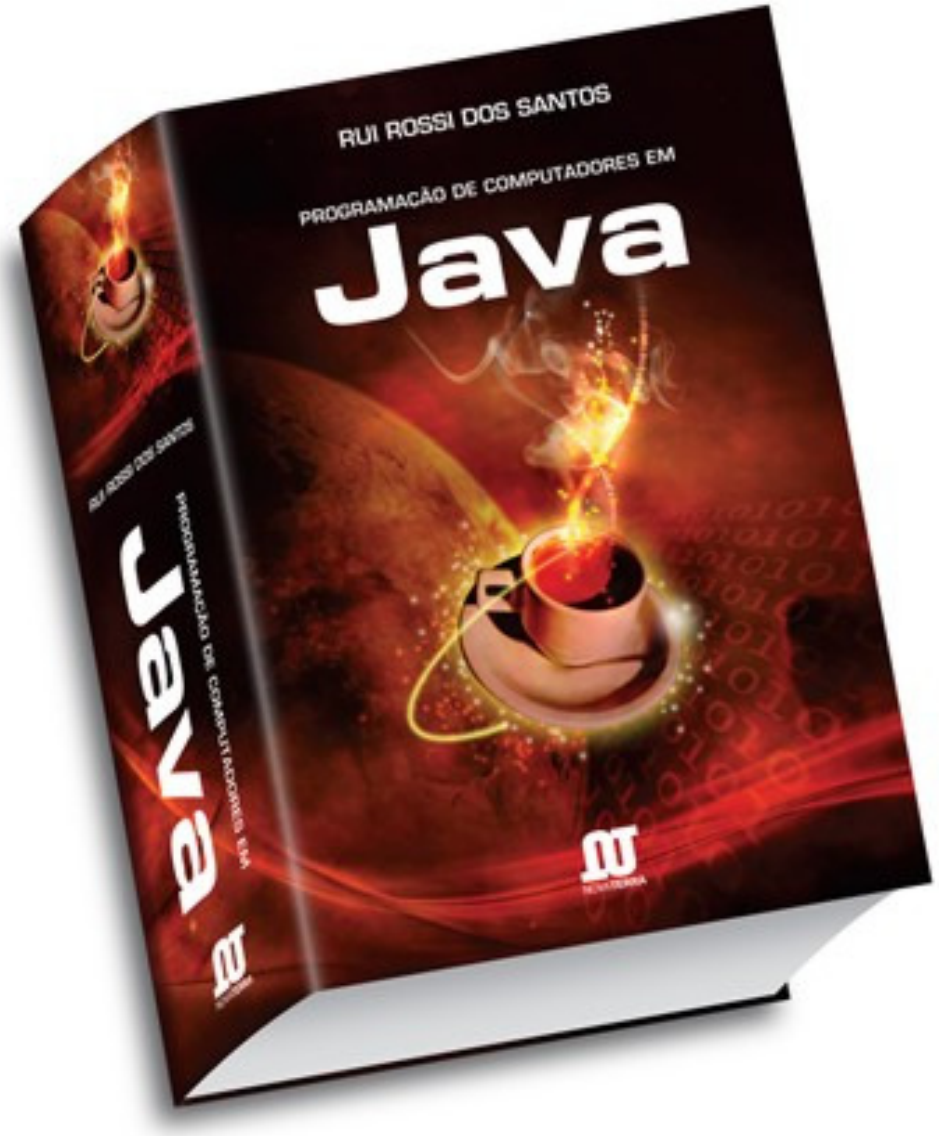
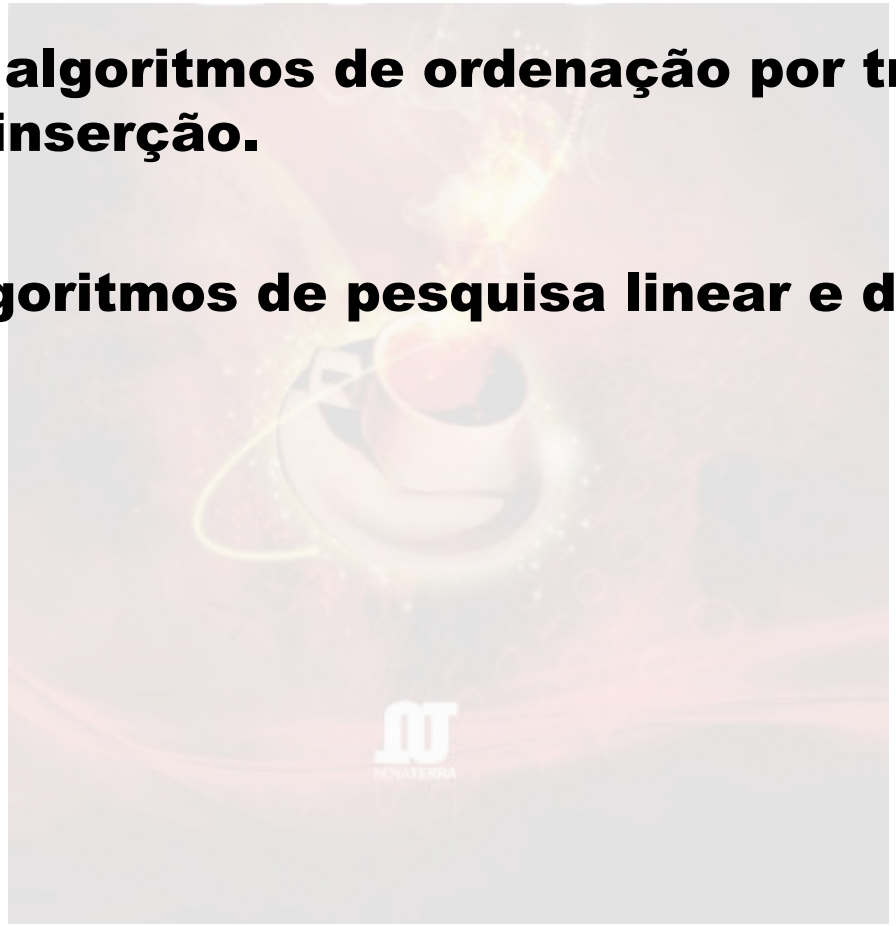


Capítulo 19

Algoritmos de Ordenação e de Pesquisa

Objetivos do Capítulo

- Conceituar ordenação e pesquisa.
- Apresentar os algoritmos de ordenação por troca, por seleção e por inserção.
- Explorar os algoritmos de pesquisa linear e de pesquisa binária.



Introdução

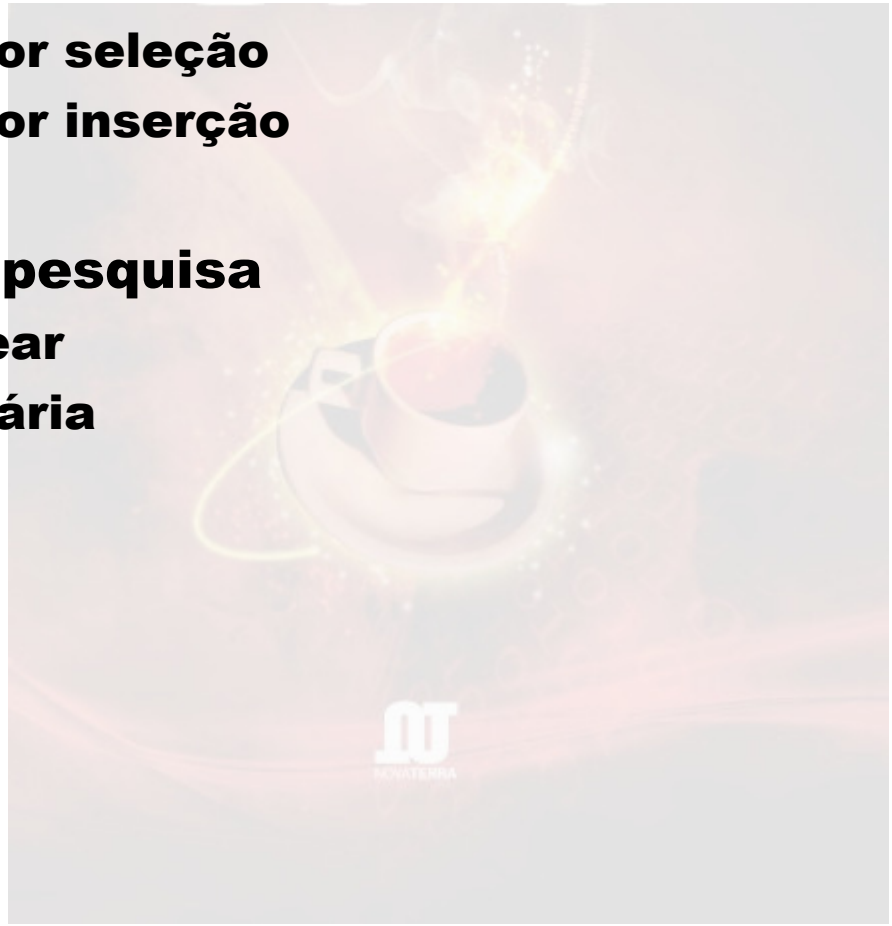
- **Ordenação**
  - **Operação realizada sobre registros**
  - **Objetivo: organizá-los sob determinada ordem**

- **Pesquisa**
  - **Operação realizada sobre registros**
  - **Objetivo: localizar um registro**
  - **Chave: campo ou chave presente em todos os registros**

Introdução

- **Algoritmos de ordenação**
  - **Ordenação por troca**
  - **Ordenação por seleção**
  - **Ordenação por inserção**

- **Algoritmos de pesquisa**
  - **Pesquisa linear**
  - **Pesquisa binária**



Ordenação por Troca

- **Características**
  - **O mais simples**
  - **O mais lento**

- **Método: bolha (bubble sort)**
  - **Compare dois elementos**
  - **Se o elemento da esquerda for maior, troque-os.**
  - **Mova-se uma posição à direita.**

  - **Obs.: acumula os elementos ordenados no final do vetor.**

Ordenação por Troca

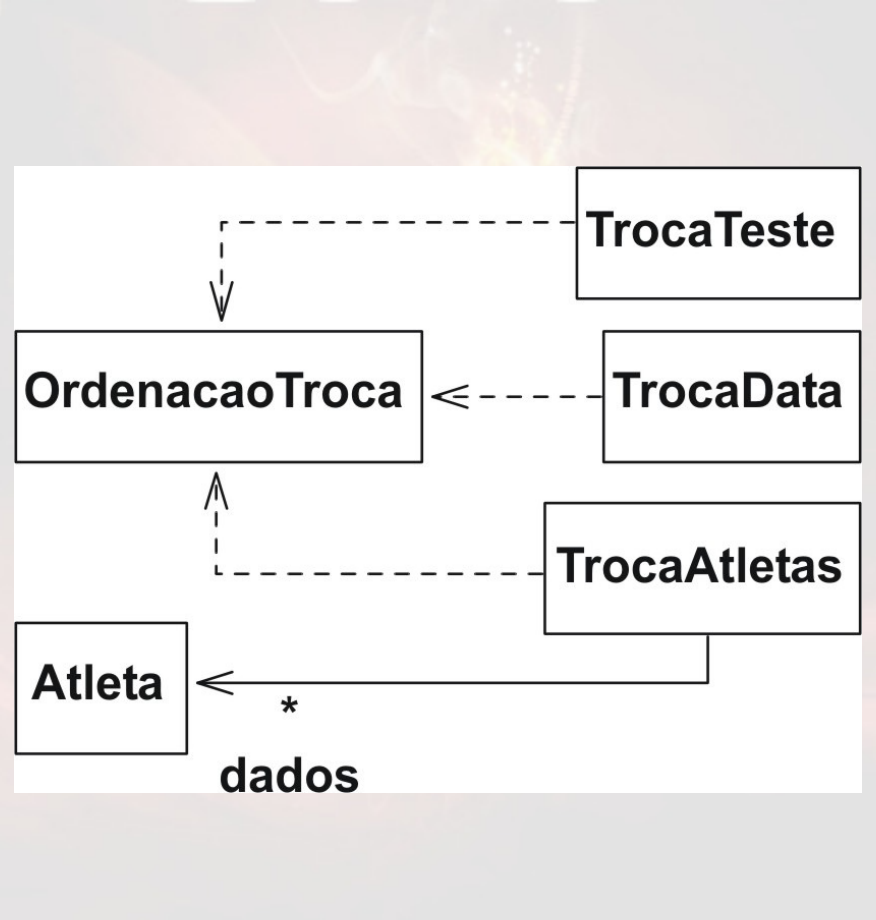
- **Modelo:**

```
int[] dados = new int[]{45,23,78,35,15};

for(int out = dados.length - 1; out > 0; out--)
  for(int in = 0; in < out; in++)
    if(dados[in] > dados[in+1]) {
      int temp = dados[in];
      dados[in] = dados[in+1];
      dados[in+1] = temp;
    }
```

Ordenação por Troca

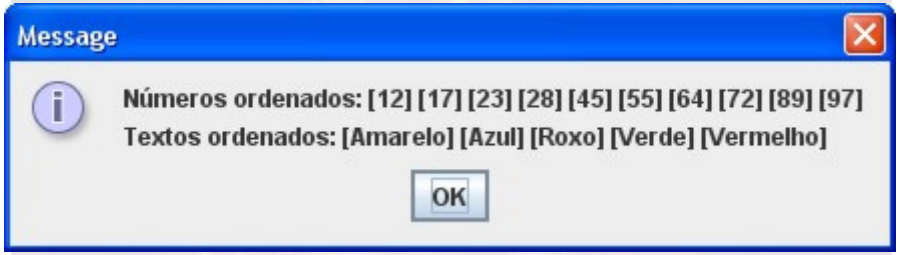
- **Exemplos:**



Ordenação por Troca

- Código 19.1 – OrdenacaoTroca.java

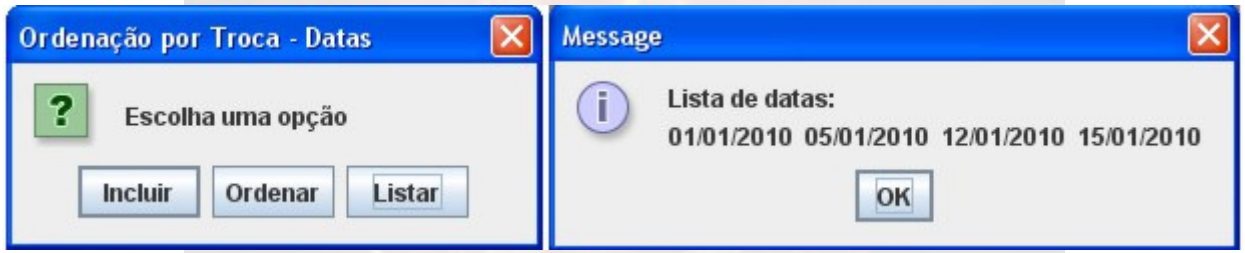
- Código 19.2 –TrocaTeste.java



Ordenação por Troca

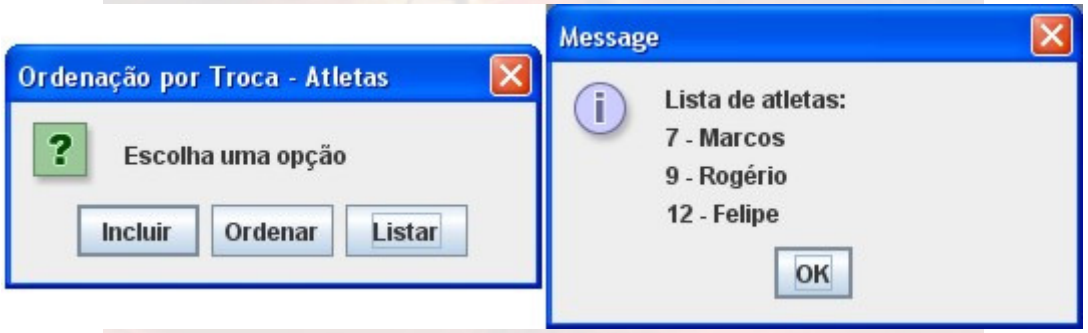
- **Código 19.1 – OrdenacaoTroca.java**

- **Código 19.3 –TrocaData.java**



Ordenação por Troca

- Código 19.1 – OrdenacaoTroca.java
- Código 19.4 –Atleta.java
- Código 19.5 –TrocaAtletas.java



Ordenação por Troca

- **Análise de eficiência: número de comparações constante**
  - **Fórmula:**

    C = E * (E – 1) / 2

    Onde:
    C = Número total de comparações
    E = Número de elementos

  - **Exemplo: vetor com 10 elementos**

    C = 10 * (10 - 1) / 2
    C = 45

Ordenação por Troca

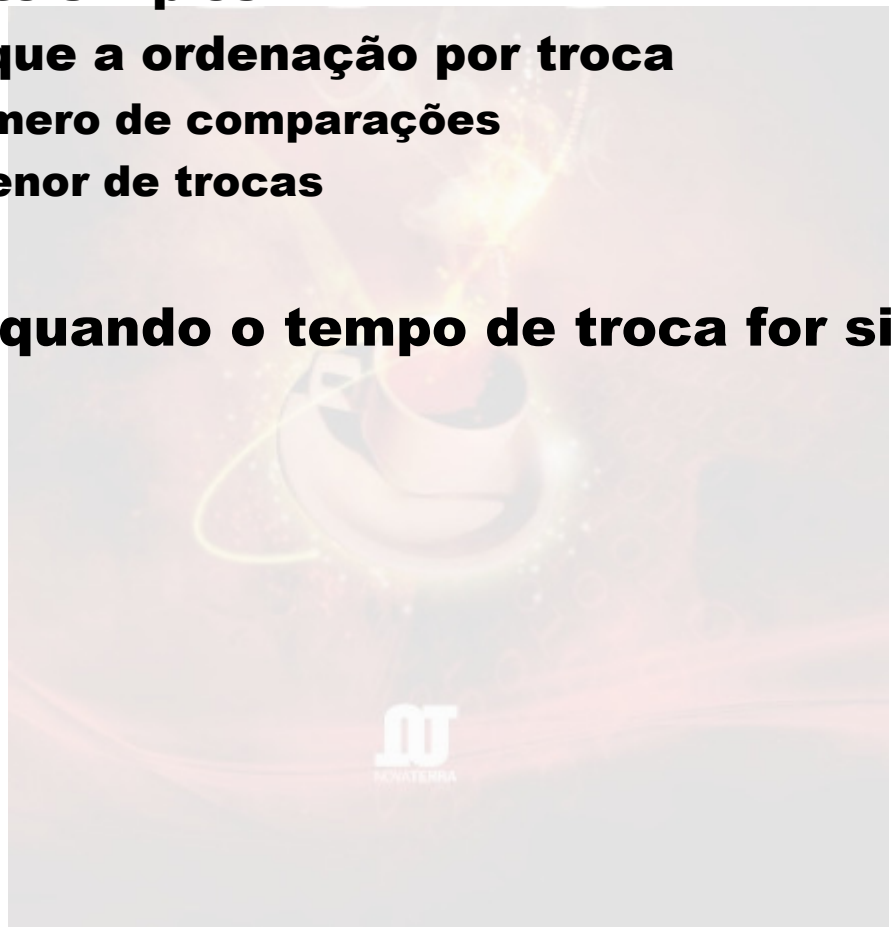
- **Análise de eficiência: número de trocas variável**
  - **Cenários:**
    - ✓ **Melhor: nenhuma troca**
    - ✓ **Pior: uma troca a cada comparação**
    - ✓ **Intermediário: uma troca a cada duas comparações**
  - **Fórmulas:**
    - ✓ **Pior cenário: T = E * (E – 1) / 2**
    - ✓ **Cenário intermediário: T = E * (E – 1) / 4**

    Onde:
    T = Número total de trocas
    E = Número de elementos
  - **Exemplo: vetor com 10 elementos**
    - ✓ **Pior cenário: 10 * (10 - 1) / 2 = 45**
    - ✓ **Cenário intermediário: 10 * (10 – 1) / 4 = 22,5 (22 ou 23 trocas)**

Ordenação por Seleção

- **Características**
  - **Relativamente simples**
  - **Mais rápida que a ordenação por troca**
    - ✓ **Mesmo número de comparações**
    - ✓ **Número menor de trocas**

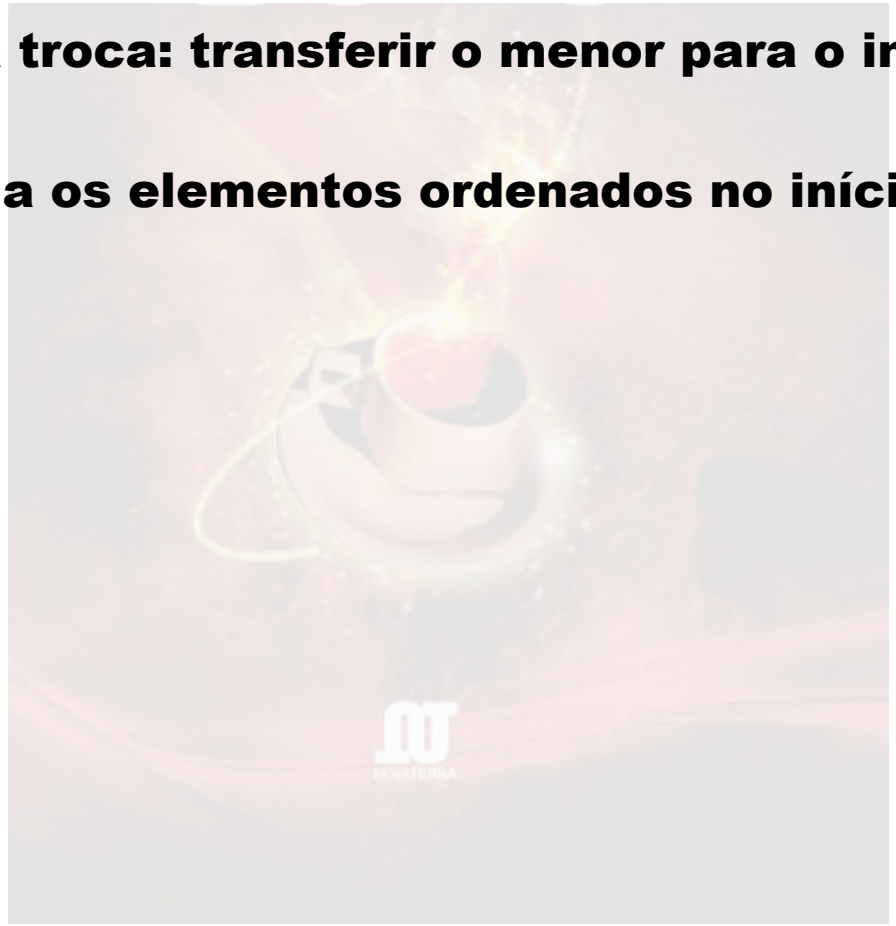
- **Mais indicada quando o tempo de troca for significativo**



Ordenação por Seleção

- **Método**
  - **Percorrer o vetor e selecionar o menor elemento**
  - **Realizar uma troca: transferir o menor para o início**

  - **Obs.: acumula os elementos ordenados no início do vetor**



Ordenação por Seleção

- **Modelo:**

```
int[] dados = new int[]{45,23,78,35,15};

for(int out = 0; out < dados.length - 1; out++) {
  int min = out;

  for(int in = out + 1; in < dados.length; in++)
    if(dados[in] < dados[min] )
      min = in;

  int temp = dados[out];
  dados[out] = dados[min];
  dados[min] = temp;
}
```

Ordenação por Seleção

- Código 19.6 – OrdenacaoSelecao.java
- Código 19.7 – SelecaoTeste.java

Ordenação por Seleção

- **Análise de eficiência: número de comparações constante**
  - **Fórmula:**

    C = E * (E – 1) / 2

    Onde:
    C = Número total de comparações
    E = Número de elementos

  - **Exemplo: vetor com 10 elementos**

    C = 10 * (10 - 1) / 2
    C = 45

Ordenação por Seleção

- **Análise de eficiência: número de trocas constante**
  - **Fórmula:**

    T = E – 1

    Onde:
    T = Número total de trocas
    E = Número de elementos

  - **Exemplo: vetor com 10 elementos**

    T = 10 – 1
    T = 9

Ordenação por Seleção

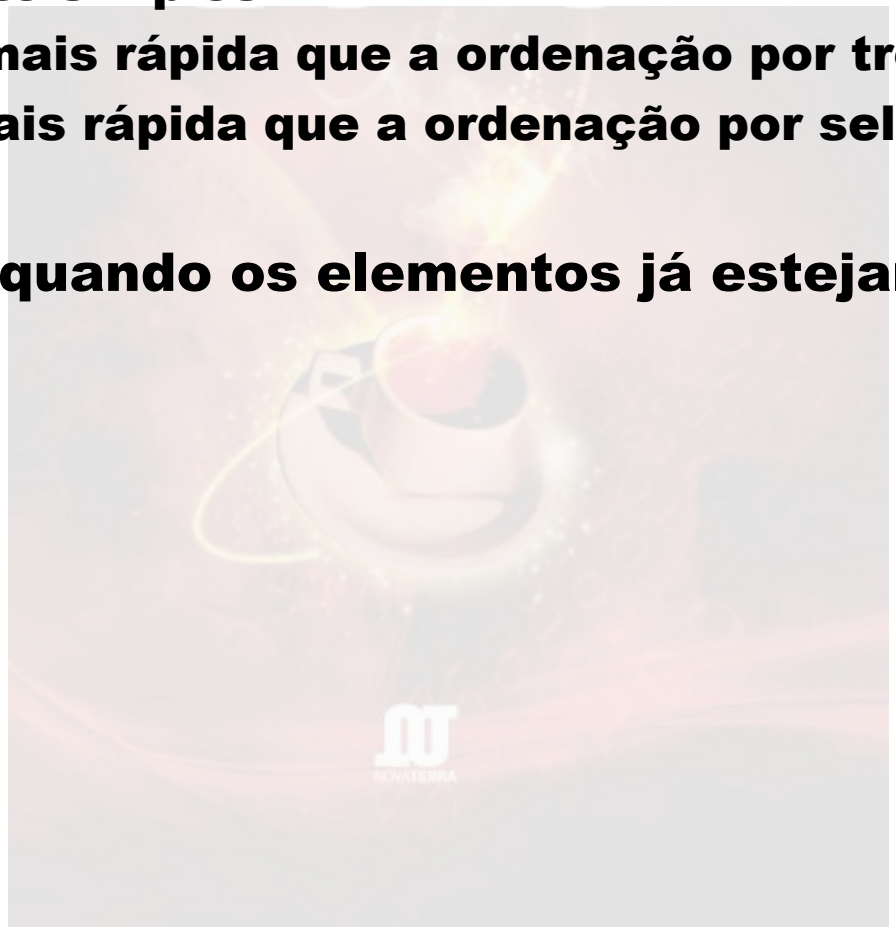
- Comparação com a ordenação por troca
  - Número de comparações idêntico
  - Número de trocas inferior

- Análise de caso: vetor com 100 elementos
  - Ordenação por troca: 4950 trocas
  - Ordenação por seleção: 99 trocas

Ordenação por Inserção

- **Características**
  - **Relativamente simples**
  - **Duas vezes mais rápida que a ordenação por troca**
  - **Um pouco mais rápida que a ordenação por seleção**

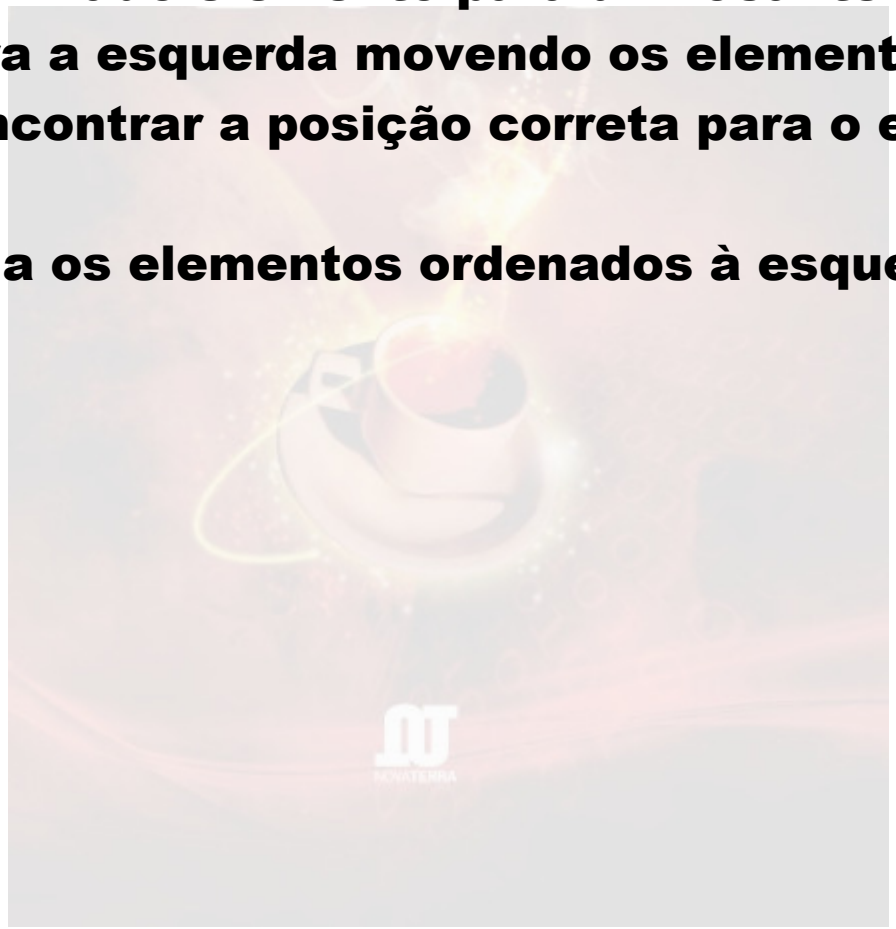
- **Mais indicada quando os elementos já estejam quase ordenados**



Ordenação por Inserção

- **Método**
  - **Copiar determinado elemento para um local temporário**
  - **Dirigir-se para a esquerda movendo os elementos para a direita**
  - **Seguir até encontrar a posição correta para o elemento**

  - **Obs.: acumula os elementos ordenados à esquerda**



Ordenação por Inserção

- **Modelo:**

```
int[] dados = new int[]{45,23,78,35,15};

for(int out = 1; out < dados.length; out++) {
  int temp = dados[out];
  int in = out;

  while(in > 0 && dados[in-1] >= temp) {
    dados[in] = dados[in-1];
    in--;
  }

  dados[in] = temp;
}
```

Ordenação por Inserção

- Código 19.8 – OrdenacaoInsercao.java
- Código 19.9 – InsercaoTeste.java

Ordenação por Inserção

- **Análise de eficiência: número de comparações variável**
  - **Cenários:**
    - ✓ **Pior: C = E * (E – 1) / 2**
    - ✓ **Intermediário: C = E * (E – 1) / 4**

    Onde:

    C = Número total de comparações

    E = Número de elementos

  - **Exemplo: vetor com 10 elementos**

    - ✓ **Número máximo de comparações: 10 * (10 - 1) / 2 = 45**
    - ✓ **Número médio de comparações: 10 * (10 – 1) / 4 = 22,5 (22 ou 23)**

Ordenação por Inserção

- **Análise de eficiência: número de trocas variável**
  - **Cenários:**
    - ✓ **Pior: T = E * (E – 1) / 2**
    - ✓ **Intermediário: T = E * (E – 1) / 4**

    Onde:

    T = Número total de trocas

    E = Número de elementos

  - **Exemplo: vetor com 10 elementos**

    - ✓ **Número máximo de trocas: 10 * (10 - 1) / 2 = 45**
    - ✓ **Número médio de trocas: 10 * (10 – 1) / 4 = 22,5 (22 ou 23)**

Ordenação por Inserção

- Comparação com a ordenação por troca
  - Em média, realiza metade do número de comparações
  - Em média, realiza o mesmo número de trocas

- Análise de caso: vetor com 100 elementos
  - Ordenação por inserção:
    - ✓ Comparações (média): 2.475
    - ✓ Trocas (média): 2.475
  - Ordenação por troca:
    - ✓ Comparações (constante): 4.950
    - ✓ Trocas (média): 2.475

Ordenação por Inserção

- Comparação com a ordenação por seleção
  - Em média, realiza metade do número de comparações
  - Em média, realiza um número maior de trocas

- Análise de caso: vetor com 100 elementos
  - Ordenação por inserção:
    - ✓ Comparações (média): 2.475
    - ✓ Trocas (média): 2.475
  - Ordenação por seleção:
    - ✓ Comparações (constante): 4.950
    - ✓ Trocas (constante): 99

Pesquisa Linear

- Objetivo: localizar um elemento em um conjunto

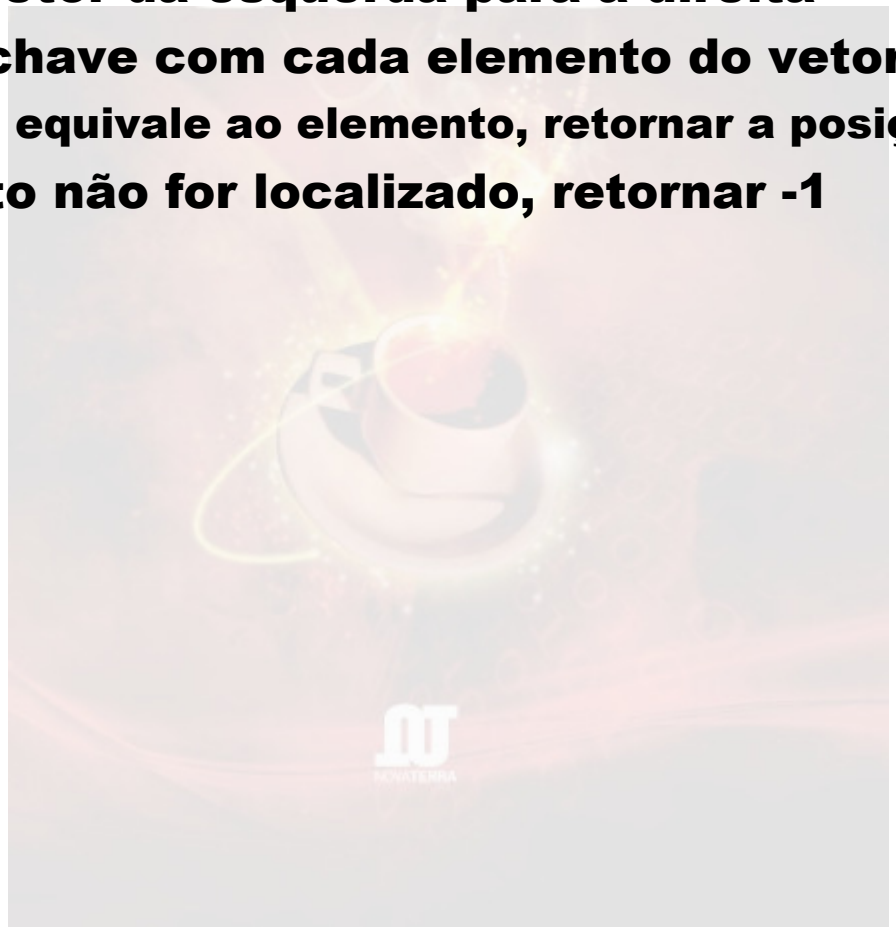
- Exemplos de conjuntos:
  - Vetores de tipos primitivos
  - Vetores de objetos
  - Coleções

- Chave: identifica o elemento a ser localizado

- Requisito:
  - Deve haver somente um elemento com a chave procurada

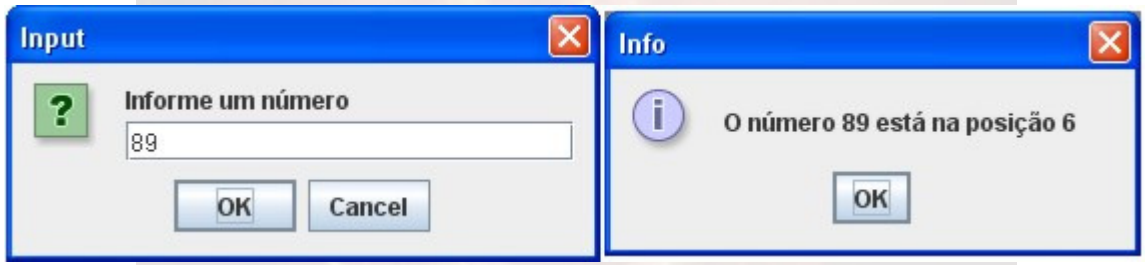
Pesquisa Linear

- **Método:**
  - **Percorrer o vetor da esquerda para a direita**
  - **Comparar a chave com cada elemento do vetor**
    - ✓ **Se a chave equivale ao elemento, retornar a posição do mesmo**
  - **Se o elemento não for localizado, retornar -1**



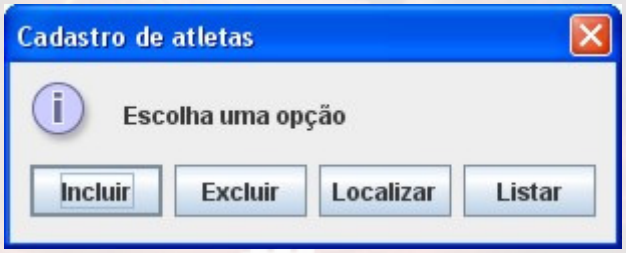
Pesquisa Linear

- **Código 19.10 – PesquisaLinear.java**
- **Código 19.11 – PLTeste.java**



Pesquisa Linear

- Código 19.4 – Atleta.java
- Código 19.10 – PesquisaLinear.java
- Código 19.12 – PLAtletas.java



Pesquisa Linear

- **Análise de eficiência: número de passos**
  - **Cenários:**
    - ✓ **Melhor: a chave está na primeira posição**
    - ✓ **Pior: a chave está na última posição**
    - ✓ **Média: P = E / 2**

    Onde:

    P = Número total de passos

    E = Número de elementos

  - **Exemplo: vetor com 100 elementos**

    - ✓ **Melhor cenário: 1 passo**
    - ✓ **Pior cenário: 100 passos**
    - ✓ **Média: 100 / 2 = 50 passos**

Pesquisa Binária

- Objetivo: localizar um elemento em um conjunto

- Exemplos de conjuntos:
  - Vetores de tipos primitivos
  - Vetores de objetos
  - Coleções

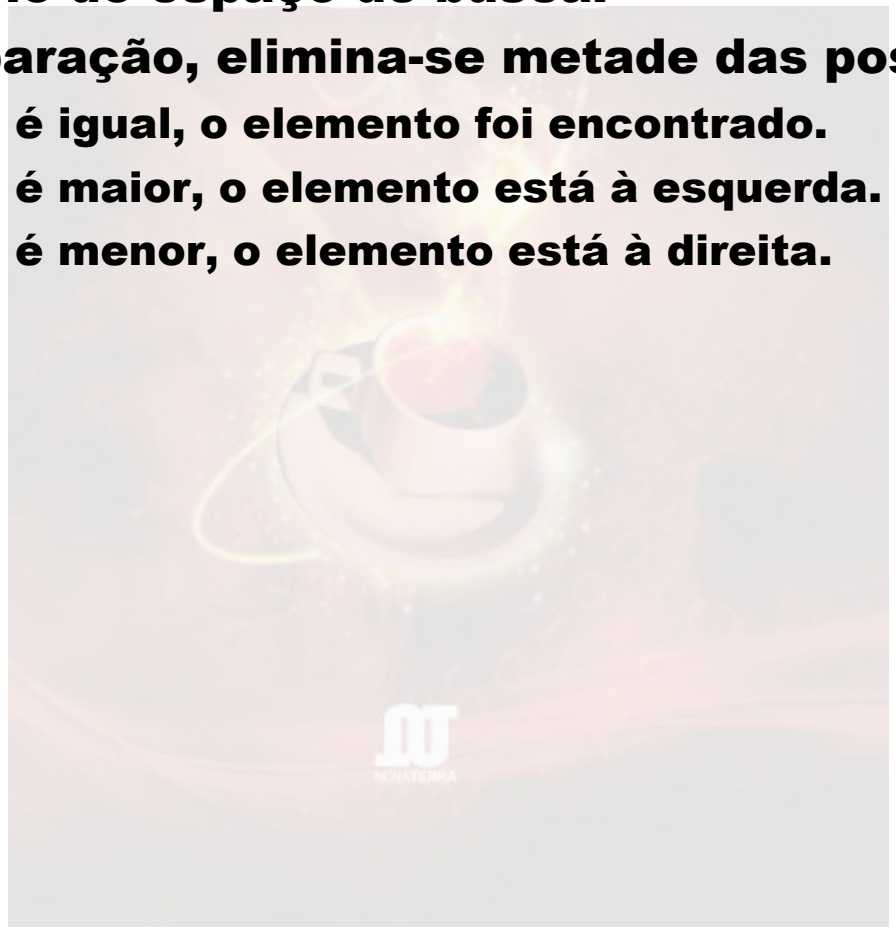
- Chave: identifica o elemento a ser localizado

- Mais eficiente que a pesquisa linear.

- Requisitos:
  - Deve haver somente um elemento com a chave procurada.
  - Os elementos devem estar ordenados.

Pesquisa Binária

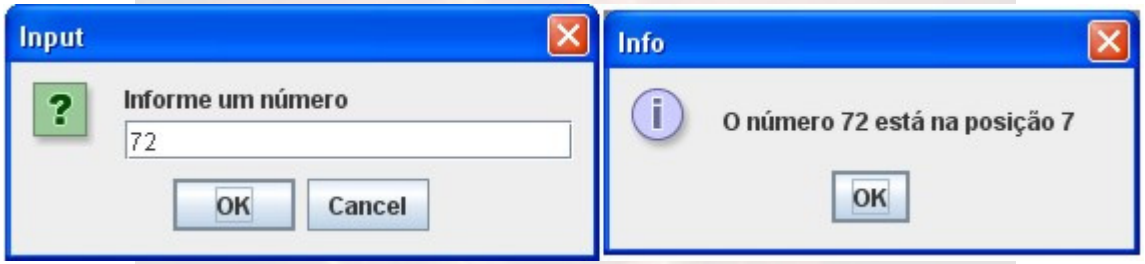
- **Método:**
  - **Iniciar no meio do espaço de busca.**
  - **A cada comparação, elimina-se metade das possibilidades.**
    - ✓ **Se a chave é igual, o elemento foi encontrado.**
    - ✓ **Se a chave é maior, o elemento está à esquerda.**
    - ✓ **Se a chave é menor, o elemento está à direita.**



Pesquisa Binária

- **Código 19.13 – PesquisaBinaria.java**

- **Código 19.14 – PBTeste.java**



Pesquisa Binária

- **Análise de eficiência: número de passos**
  - **Cenários:**
    - ✓ **Melhor (a chave está na primeira posição analisada): 1 passo**
    - ✓ **Pior (a chave está na última posição analisada): $P = \log_2(E)$**
    - ✓ **Média: $P = \log_2(E) / 2$**

    Onde:

    P = Número total de passos

    E = Número de elementos

  - **Exemplo: vetor com 100 elementos**

    - ✓ **Melhor cenário: 1 passo**
    - ✓ **Pior cenário: $\log_2(100) = 6{,}644$ (7 passos)**
    - ✓ **Média: $\log_2(100) / 2 = 3{,}322$ (4 passos)**

Pesquisa Binária

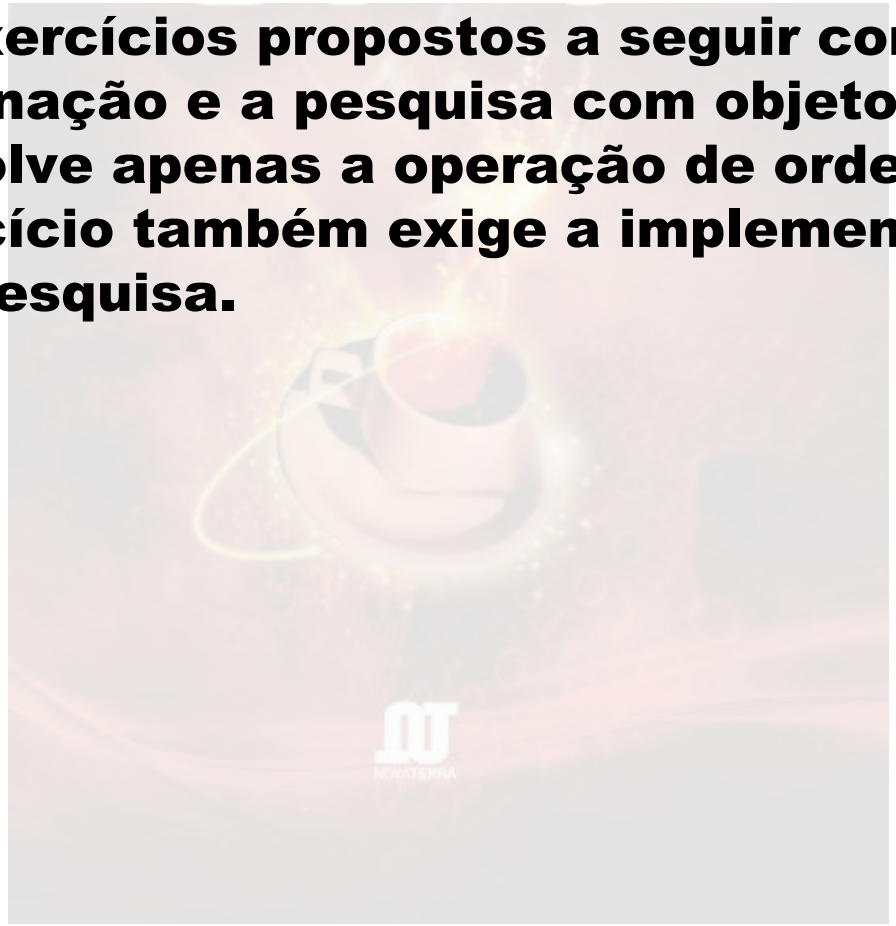
- **Análise de eficiência: a função logarítmica**
  - **Fórmula:**
    - ✓ $P = \log_2(N)$
  - **Notas**
    - ✓ **O logaritmo na base 2 de um número N indica a quantidade de vezes que o 2 precisa ser multiplicado por ele próprio até que o resultado seja igual a este número N.**
    - ✓ **A maioria das calculadoras possuem uma função para calcular o logaritmo de um número na base 10. A conversão desse resultado para a base 2 pode ser feita multiplicando-o pela constante 3,322.**

$\log_{10}(100) = 2$

$\log_2(100) = 2 * 3{,}322 = 6{,}644$

Exercícios

- **A maioria das operações de ordenação e de pesquisa que precisam ser realizadas em sistemas reais envolve objetos. Por isso, os exercícios propostos a seguir contemplam apenas a ordenação e a pesquisa com objetos. O primeiro exercício envolve apenas a operação de ordenação e o segundo exercício também exige a implementação de uma operação de pesquisa.**



Exercício 1

- Crie um aplicativo que permita realizar cadastros de clientes.
  - Cada cadastro de cliente deve ser composto pelos seguintes dados: código, nome, CPF e telefone.
  - Além de permitir o cadastro de novos clientes, este aplicativo também deve permitir a exclusão de cadastros, a sua ordenação e a produção de uma lista com todos eles.
- O aplicativo deve permitir que o usuário escolha entre ordenar os cadastros dos clientes com base no código ou no nome dos mesmos.
  - Para realizar a ordenação, utilize o algoritmo que for mais eficiente para este cenário.

Exercício 2

- Crie um aplicativo que permita realizar cadastros de funcionários.
  - Cada cadastro de funcionário deve ser composto pelos seguintes dados: matrícula, nome, nascimento e salário.
  - Além de permitir o cadastro de novos funcionários, este aplicativo também deve permitir a exclusão de qualquer cadastro, a realização de pesquisas e a geração de um relatório.
- Logo depois de gravar cada novo cadastro, este aplicativo deve realizar a ordenação dos mesmos e mantê-los sempre em ordem crescente de matrícula.
  - Escolha o algoritmo de ordenação que for mais apropriado para realizar esta tarefa.
- Deve ser permitido ao usuário realizar pesquisas nos cadastros de funcionários.
  - O usuário deve informar a matrícula do funcionário que deseja localizar e o aplicativo deve utilizar o algoritmo de pesquisa binária para procurar pelo cadastro correspondente.

Contato

Com o autor:

Rui Rossi dos Santos
E-mail: livros@ruirossi.pro.br
Web Site: http://www.ruirossi.pro.br

Com a editora:

Editora NovaTerra
Telefone: (21) 2218-5314
Web Site: http://www.editoranovaterra.com.br