

EDCO4B

ESTRUTURAS DE DADOS 2

Aula 01 - Ordenação

Prof. Rafael G. Mantovani
Prof. Luiz Fernando Carvalho

Roteiro

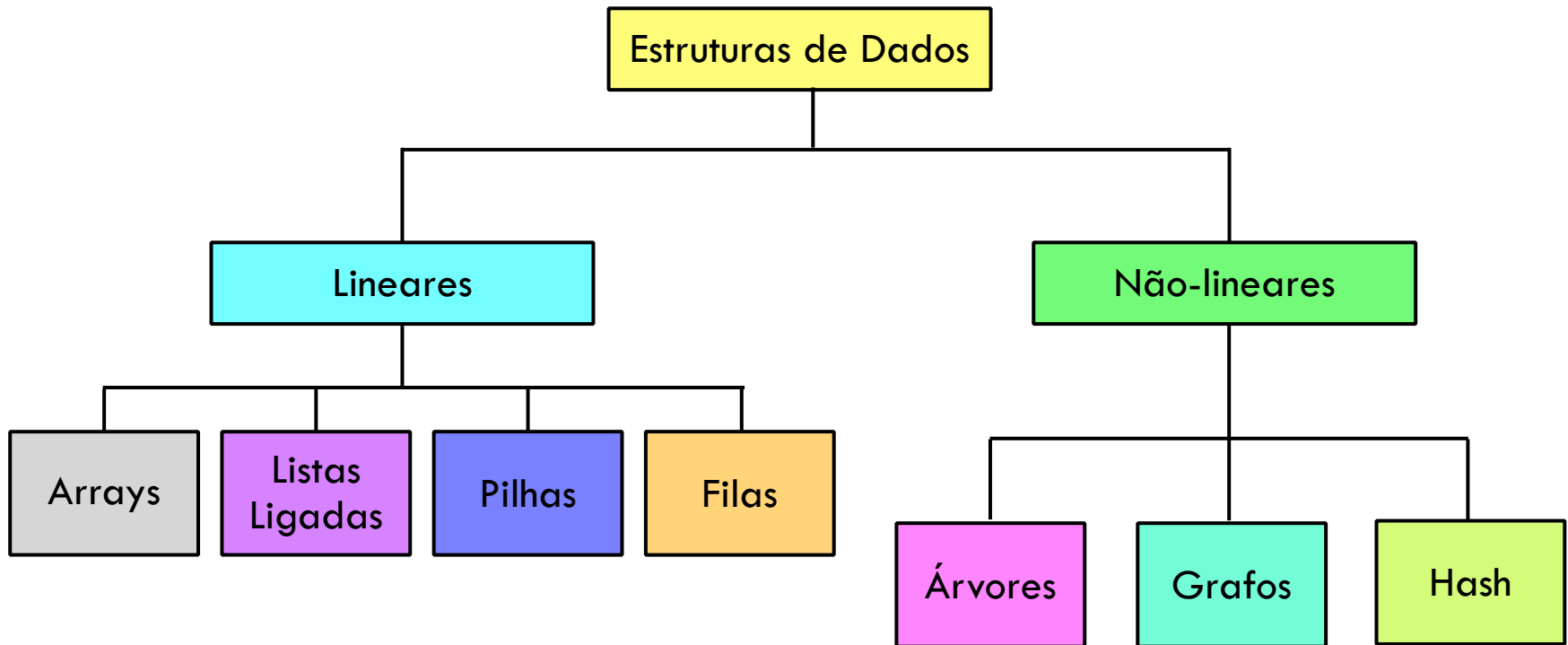


- 1 Introdução**
- 2 Ordenação**
- 3 Tipos de Algoritmos de Ordenação**
- 4 Tipos de Busca + Hands On**
- 5 Referências**

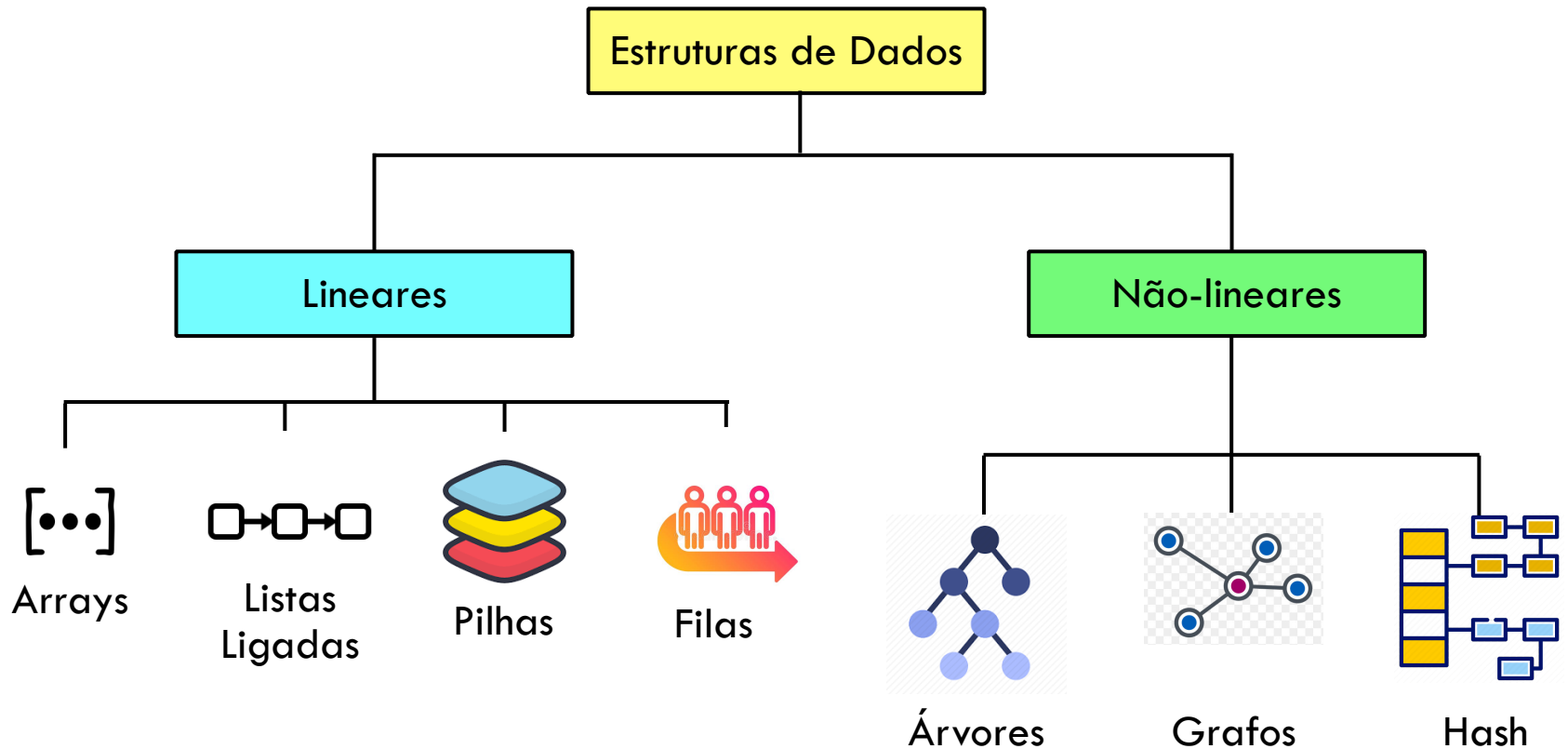
Roteiro

- 1 Introdução**
- 2 Ordenação**
- 3 Tipos de Algoritmos de Ordenação**
- 4 Tipos de Busca + Hands On**
- 5 Referências**

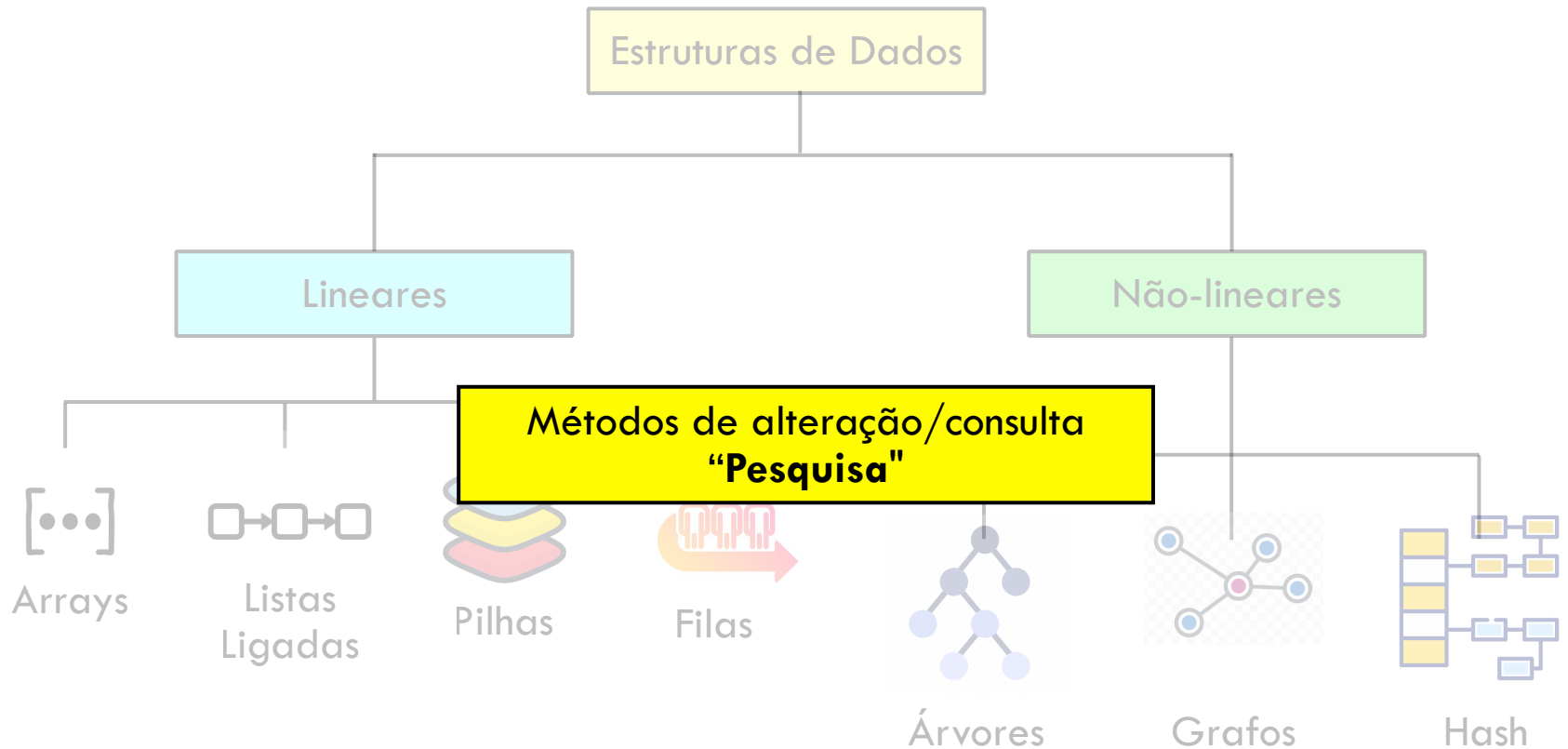
Introdução



Introdução



Introdução



Introdução



Estruturas de Dados



Pesquisa

Introdução



Estruturas de Dados



Pesquisa

Busca: “Recuperação de dados armazenados em um repositório ou banco de dados ...”

Introdução



Estruturas de Dados



Pesquisa

Busca: “Recuperação de dados armazenados em um repositório ou banco de dados ...”

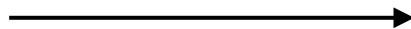
□ Eficiência depende:

1. Dados estruturados (vetor, lista, árvore?)
2. Dados são/estão ordenados?
3. Existem valores duplicados?

Introdução



Estruturas de Dados



Pesquisa

Busca: “Recuperação de dados armazenados em um repositório ou banco de dados ...”

□ Eficiência depende:

1. Dados estruturados (vetor, lista, árvore?)

2. Dados são/estão ordenados?

3. Existem valores duplicados?

Algoritmos de
Ordenação (Sort)

Roteiro



- 1 Introdução
- 2 Ordenação
- 3 Tipos de Algoritmos de Ordenação
- 4 Tipos de Busca + Hands On
- 5 Referências

Ordenação



Algoritmos de Ordenação

Ordenação



Algoritmos de Ordenação

... são bons exemplos de como resolver problemas práticos nos computadores:

Ordenação



Algoritmos de Ordenação

... são bons exemplos de como resolver problemas práticos nos computadores:

* q1: **vários algoritmos** para resolver uma **mesma tarefa**

Ordenação

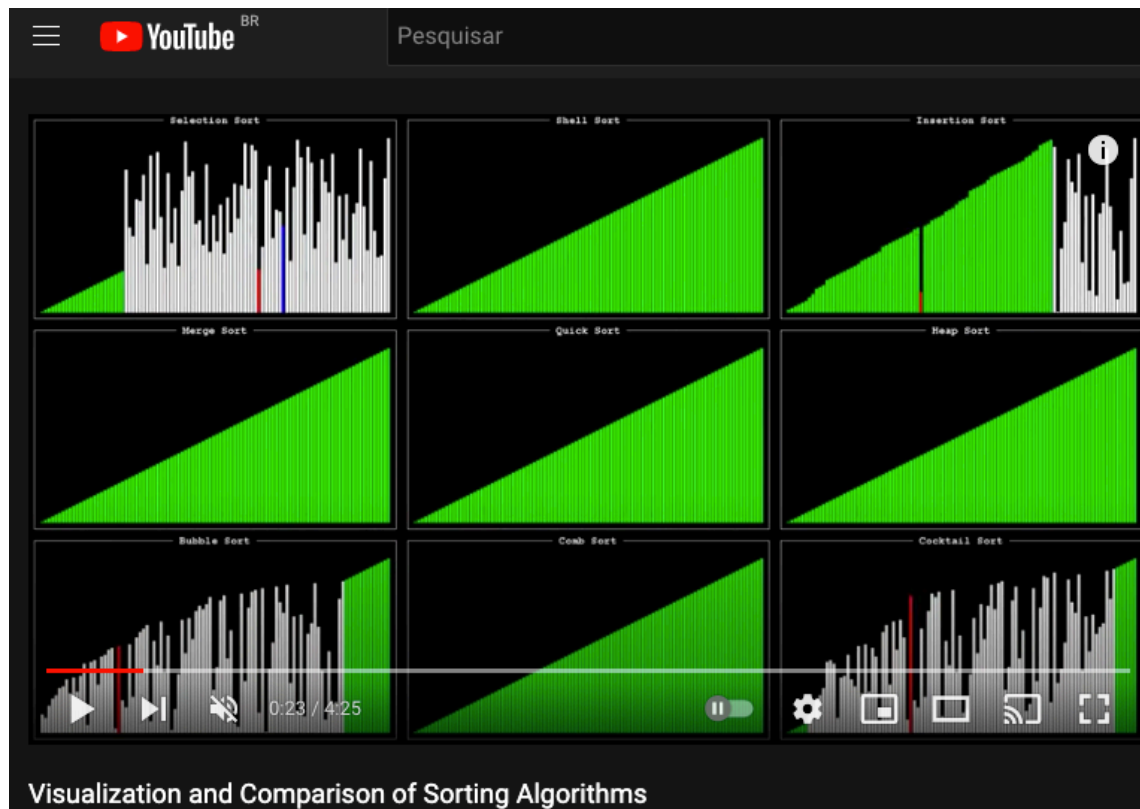


Algoritmos de Ordenação

... são bons exemplos de como resolver problemas práticos nos computadores:

- * q1: **vários algoritmos** para resolver uma **mesma tarefa**
- * q2: **quando** usar depende da **aplicação**

Ordenação



Compartivo de algoritmos de ordenação
Link: <https://www.youtube.com/watch?v=ZZuD6iUe3Pc>

Ordenação



Algoritmos de
Ordenação

Ordenação



Algoritmos de
Ordenação

Objetivo:
rearranjar ...



ordem ascendente

R, A, F, A, E, L> A, A, E, F, L, R

Ordenação



Algoritmos de
Ordenação

Objetivo:
rearranjar ...



ordem ascendente

R, A, F, A, E, L> A, A, E, F, L, R

Objetivo:
rearranjar ...



ordem descendente

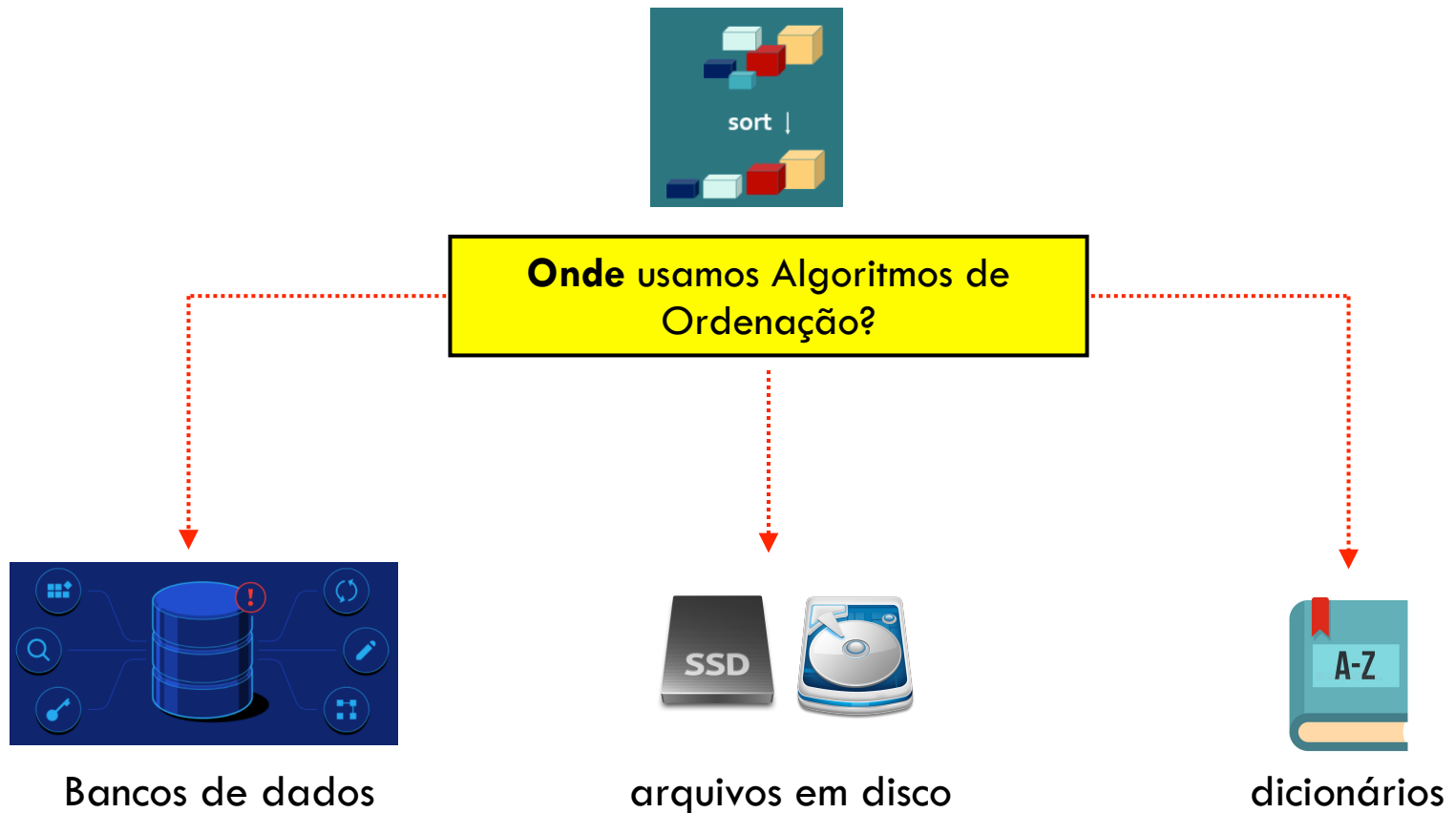
10, 5, 6, 7, 54> 54, 10, 7, 6, 5

Ordenação



Onde usamos Algoritmos de Ordenação?

Ordenação



Roteiro

- 1 Introdução
- 2 Ordenação
- 3 Tipos de Algoritmos de Ordenação
- 4 Tipos de Busca + Hands On
- 5 Referências

Tipos de Ordenação



Algoritmos
baseados em
comparação

Tipos de Ordenação

Algoritmos
baseados em
comparação

Compara dois valores ou duas posições o arranjo ...

? ↓

3	4	1	5	2
---	---	---	---	---

Sorting index 0
Smallest element at 2
Swap them

? ↓

1	4	3	5	2
---	---	---	---	---

Sorting index 1
Smallest element at 4
Swap them

? ↓

1	2	3	5	4
---	---	---	---	---

Sorting index 2
Smallest element at 2
Leave it

? ↓

1	2	3	5	4
---	---	---	---	---

Sorting index 3
Smallest element at 4
Swap them

1	2	3	4	5
---	---	---	---	---

Array is now sorted
☺

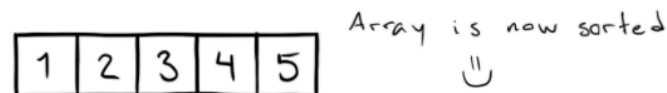
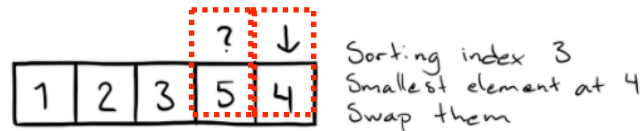
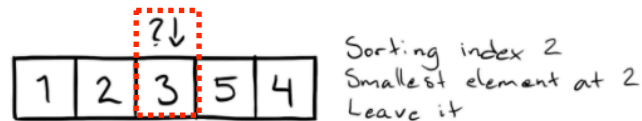
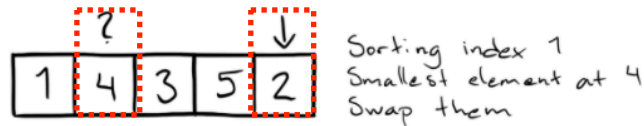
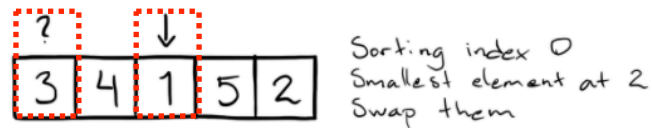
Link: shorturl.at/iBER1

Tipos de Ordenação

Algoritmos
baseados em
comparação

Iterativamente compara os
elementos (Selection Sort)

Compara dois valores ou duas posições o arranjo ...



Link: shorturl.at/iBER1

Tipos de Ordenação



Algoritmos
baseados em
distribuição

Distribui e reorganiza os dados

Tipos de Ordenação

Algoritmos
baseados em
distribuição

Distribui e reorganiza os dados
exemplo: baralho de cartas



Considerando
símbolos e naipes

Tipos de Ordenação

Algoritmos
baseados em
distribuição

Distribui e reorganiza os dados
exemplo: baralho de cartas



Considerando
símbolos e naipes

1. criar 13 montes, um para cada símbolo e distribuir as cartas
2. juntar o montes na ordem dos símbolos
3. criar 4 montes, um para cada naipe e distribuir as cartas
4. juntar os montes na ordem dos naipes
5. saída: conjunto ordenado

Tipos de Ordenação

Algoritmos
baseados em
distribuição

Distribui e reorganiza os dados
exemplo: baralho de cartas



Considerando
símbolos e naipes

Radixsort

1. criar 13 montes, um para cada símbolo e distribuir as cartas
2. juntar o montes na ordem dos símbolos
3. criar 4 montes, um para cada naipe e distribuir as cartas
4. juntar os montes na ordem dos naipes
5. saída: conjunto ordenado

Não existem comparações !

Tipos de Ordenação

Resumindo:

Algoritmos
baseados em
distribuição

- * não existem comparações
- * pode acarretar num alto consumo de memória: $O(n)$

Algoritmos
baseados em
comparação

- * usam comparações
- * podem ser mais simples e econômicos

Tipos de Ordenação

Outra classificação:

Ordenação Interna

- * todos os dados cabem na memória (principal)
- * qualquer dado/registro pode ser acessado imediatamente

Ordenação Externa

- * os dados não cabem na memória (principal), por isso são armazenados em disco
- * dados/registros são acessados sequencialmente ou em grandes blocos

Ordenação Interna



Algoritmos:

Ordenação
Interna

Ordenação Interna

Algoritmos:

Ordenação
Interna

1. *Bubble sort*
2. *Selection sort*
3. *Insertion sort*

3 métodos **simples**:

- * Fácil implementação
- * Fácil entendimento
- * Bons para conjuntos pequenos
- * Em geral: $O(n^2)$

Ordenação Interna

Algoritmos:

Ordenação Interna

1. *Bubble sort*
2. *Selection sort*
3. *Insertion sort*

3 métodos **simples**:

- * Fácil implementação
- * Fácil entendimento
- * Bons para conjuntos pequenos
- * Em geral: $O(n^2)$

4. *Merge sort*
5. *Quick sort*
6. *Heap sort*

3 métodos **sofisticados**:

- * Mais complexos
- * Recursivos
- * Bons para conjuntos grandes
- * Em geral: $O(n \log n)$

Roteiro

- 1 Introdução
- 2 Ordenação
- 3 Tipos de Algoritmos de Ordenação
- 4 Tipos de Busca + Hands On
- 5 Referências

Tipos de Busca



1. Busca Linear
2. Busca Ordenada
3. Busca Binária

Busca Linear

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Busca Linear

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

vetor com elementos não ordenados

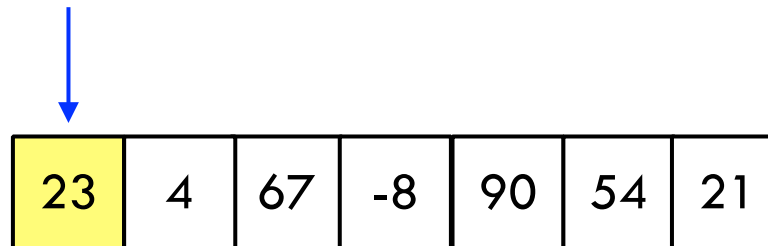
Busca Linear

* Procurar o valor: -8

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Busca Linear

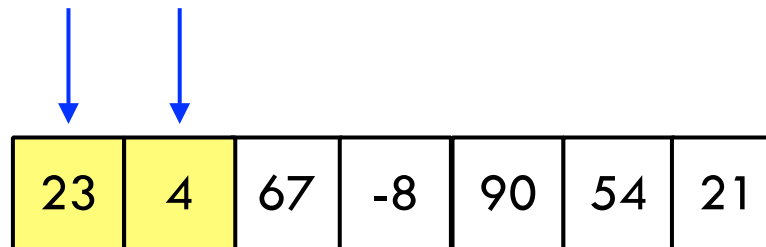
* Procurar o valor: -8



23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Busca Linear

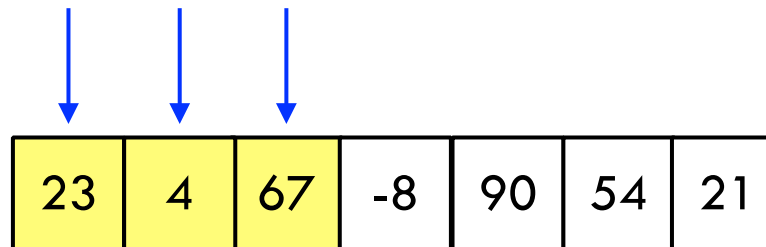
* Procurar o valor: -8



23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Busca Linear

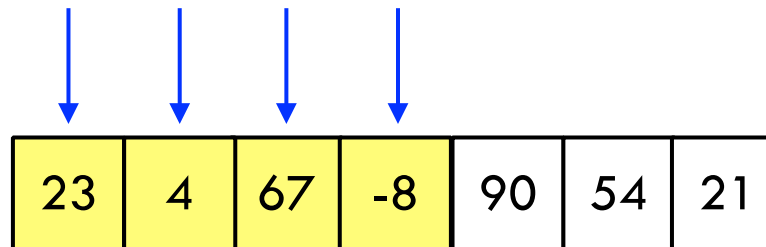
* Procurar o valor: -8



23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Busca Linear

* Procurar o valor: -8

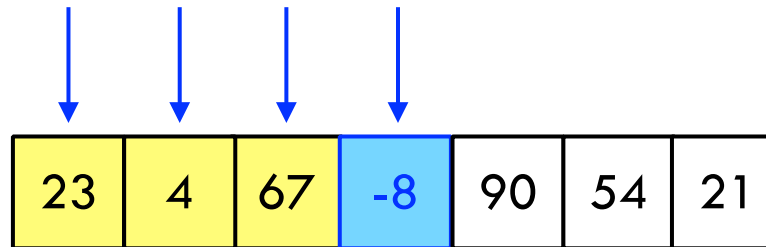


The diagram illustrates a linear search process on an array. The array contains seven elements: 23, 4, 67, -8, 90, 54, and 21. The first four elements (23, 4, 67, and -8) are highlighted in yellow, and blue arrows point down to each of them, indicating that the search algorithm has checked these values. The remaining three elements (90, 54, and 21) are in white boxes, indicating they have not yet been checked.

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Busca Linear

* Procurar o valor: -8



Valor encontrado!

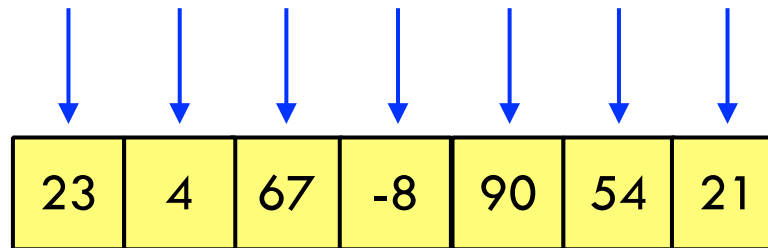
Busca Linear

* Procurar o valor: 80

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Busca Linear

* Procurar o valor: 80



23	4	67	-8	90	54	21
----	---	----	----	----	----	----

Valor não encontrado!

Busca Linear

/ testa todas as posições até encontrar o elemento desejado ou até chegar ao final do vetor*

Parametros:

***array** é o vetor*

***elemento** é o elemento que se deseja procurar*

**/*

def buscaLinear(array, elemento)

/ Retorna a posição do elemento ou -1 caso não encontre */*

Busca Ordenada

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

vetor com elementos ordenados

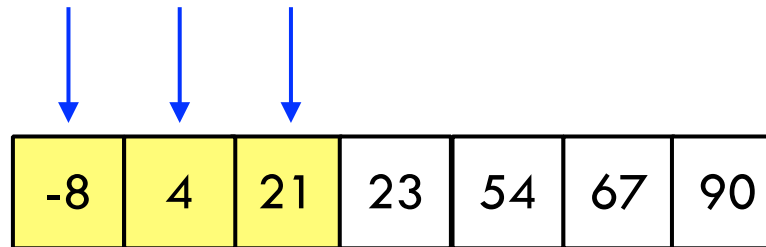
Busca Ordenada

* Procurar o valor: 21

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

Busca Ordenada

* Procurar o valor: 21



-8	4	21	23	54	67	90
----	---	----	----	----	----	----

Valor encontrado!

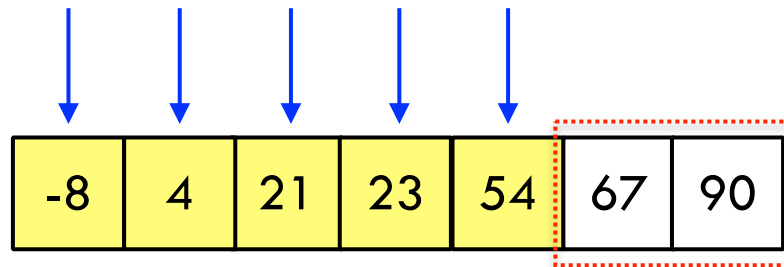
Busca Ordenada

* Procurar o valor: 40

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

Busca Ordenada

* Procurar o valor: 40



Valor não encontrado!

Busca Ordenada

/ testa todas as posições até encontrar o elemento desejado , ou até que o valor da posição testada for maior do que o elemento, ou chegou até o final do vetor*

Parametros:

***array** é o vetor*

***Elemento** é o elemento que se deseja procurar*

**/*

def estaOrdenado(array)

def buscaOrdenada(array, elemento)

/ Retorna a posição do elemento ou -1 caso não encontre */*

Busca Binária

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

vetor com elementos ordenados

Busca Binária

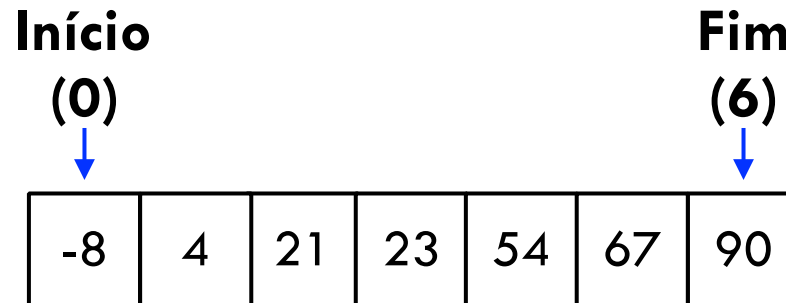
* Procurar o valor: 4

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

vetor com elementos ordenados

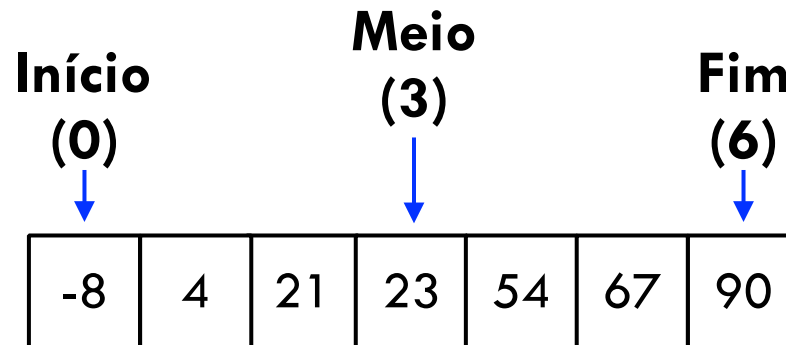
Busca Binária

* Procurar o valor: 4



Busca Binária

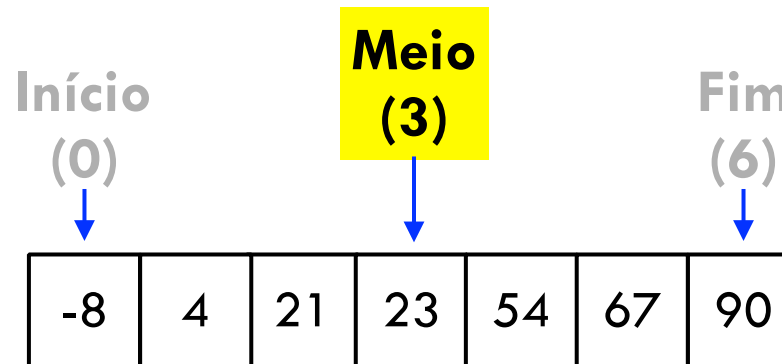
* Procurar o valor: 4



$$\text{Meio} = (\text{Inicio} + \text{Fim}) / 2$$

Busca Binária

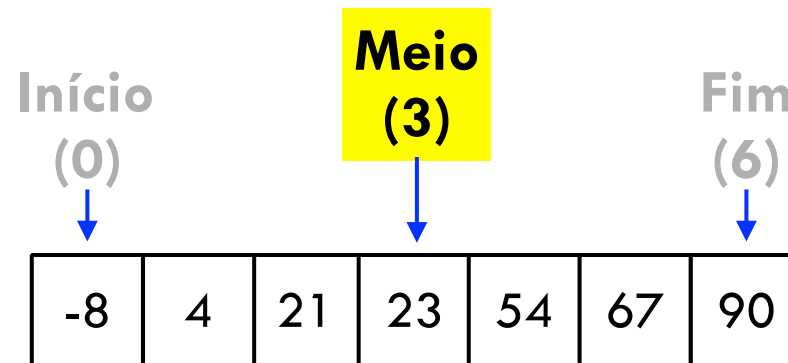
* Procurar o valor: 4



$$\text{Meio} = (\text{Inicio} + \text{Fim}) / 2$$

Busca Binária

* Procurar o valor: 4

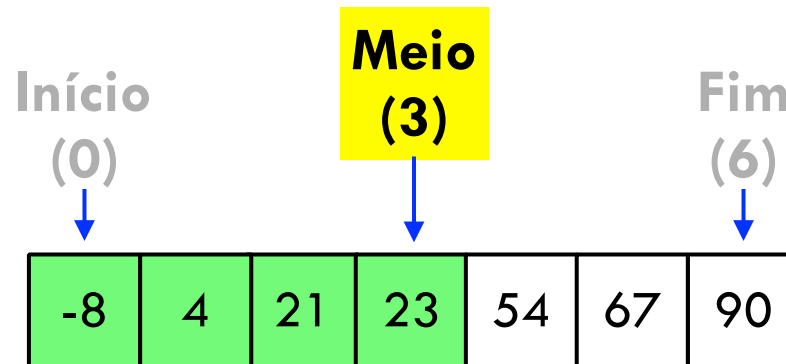


$$\text{Meio} = (\text{Início} + \text{Fim}) / 2$$

valor == vetor[meio] ???
valor > vetor[meio]: olhar p direita
valor < vetor[meio]: olhar p esquerda

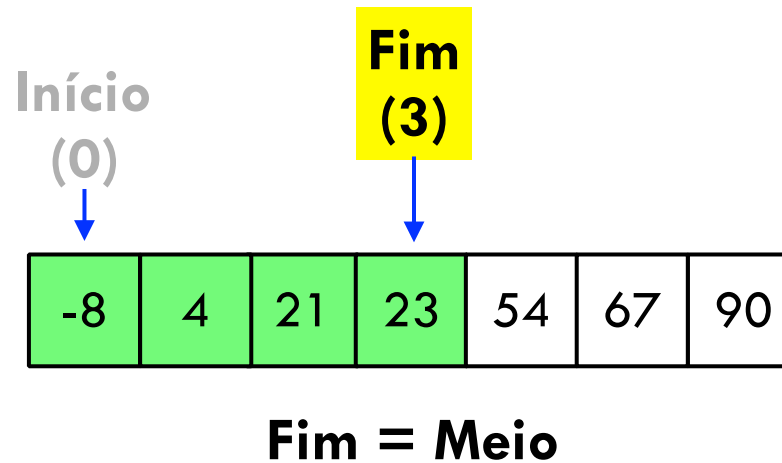
Busca Binária

* Procurar o valor: 4



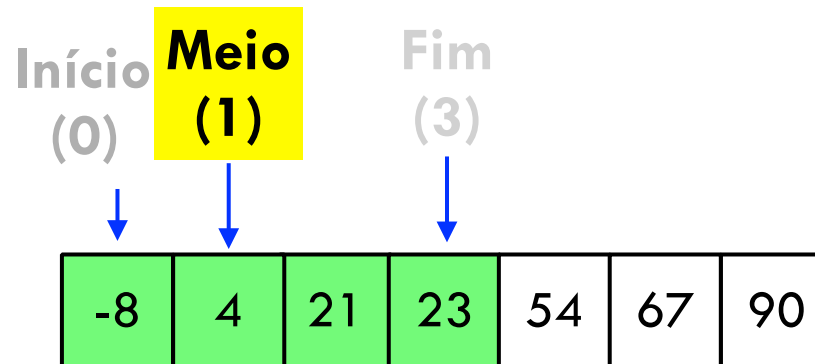
Busca Binária

* Procurar o valor: 4



Busca Binária

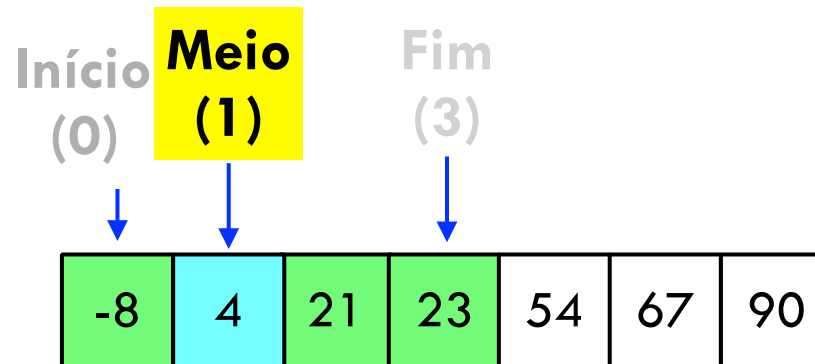
* Procurar o valor: 4



$$\text{Meio} = (\text{Início} + \text{Fim}) / 2$$

Busca Binária

* Procurar o valor: 4



$$\text{Meio} = (\text{Início} + \text{Fim}) / 2$$

valor == vetor[meio] ??? Valor encontrado!

Busca Binária

/ Utiliza a divisão e conquista. Testa sempre o elemento na metade do intervalo válido.*

Parametros:

***array** é o vetor*

***elemento** é o elemento que se deseja procurar*

**/*

def estaOrdenado(array)

def buscaBinaria(array, elemento)

/ Retorna a posição do elemento ou -1 caso não encontre */*

Hands On

□ Implementar e comparar três métodos de busca:

1. Busca Linear
2. Busca Ordenada
3. Busca Binária



Hands On :))

```
/*hands on. Fazer: */
```

```
def buscaLinear(array, elemento)
```

```
def buscaOrdenada(array, elemento)
```

```
def buscaBinaria(array, elemento)
```

```
/*
```

Existem condições para que os métodos funcionem?

Realizar testes com vetores de tamanhos diferentes.

Criar funções auxiliares se necessário.

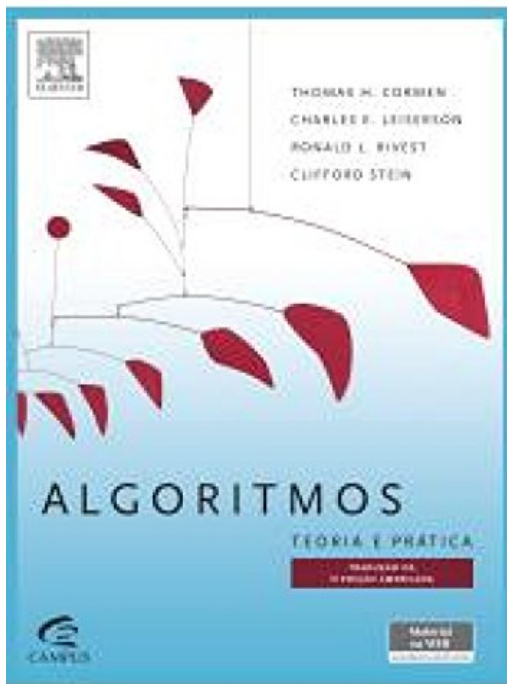
Computar o número de comparações realizados por cada função em cada caso.

*Qual é o mais eficiente? */*

Roteiro

- 1 Introdução
- 2 Ordenação
- 3 Tipos de Algoritmos de Ordenação
- 4 Tipos de Busca + Hands On
- 5 Referências

Referências sugeridas

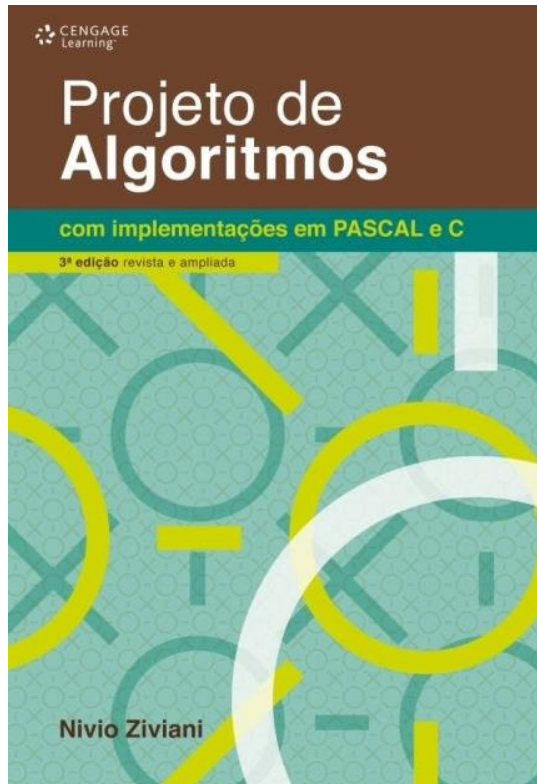


[Cormen et al, 2018]

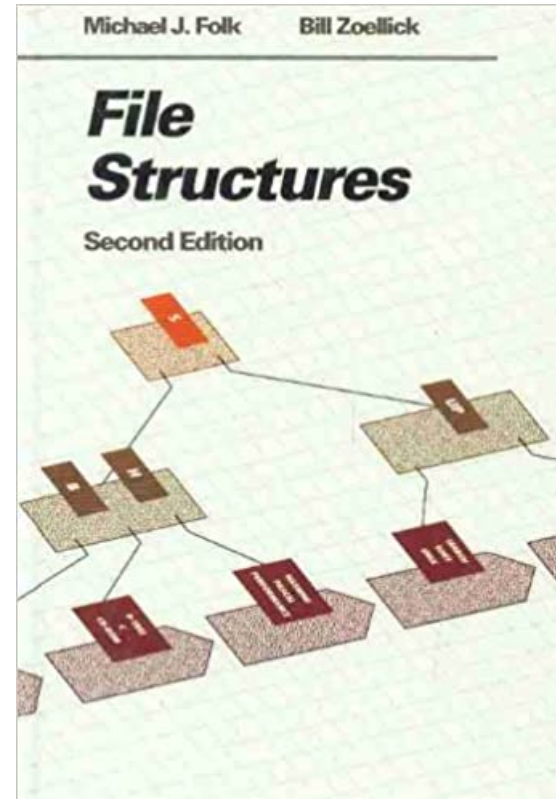


[Drozdek, 2017]

Referências sugeridas



[Ziviani, 2010]



[Folk & Zoellick, 1992]