

EDCO4B

ESTRUTURAS DE DADOS 2

Aula 05 - Quick Sort

Prof. Rafael G. Mantovani

Licença

Este trabalho está licenciado com uma Licença CC BY-NC-ND 4.0:



maiores informações:

https://creativecommons.org/licenses/by-nc-nd/4.0/deed.pt_BR

Roteiro



- 1** Introdução
- 2** Quick Sort
- 3** Exemplo
- 4** Exercício
- 5** Referências

Roteiro

- 1** Introdução
- 2** Quick Sort
- 3** Exemplo
- 4** Exercício
- 5** Referências

Introdução



**Algoritmos de
Ordenação**

Introdução

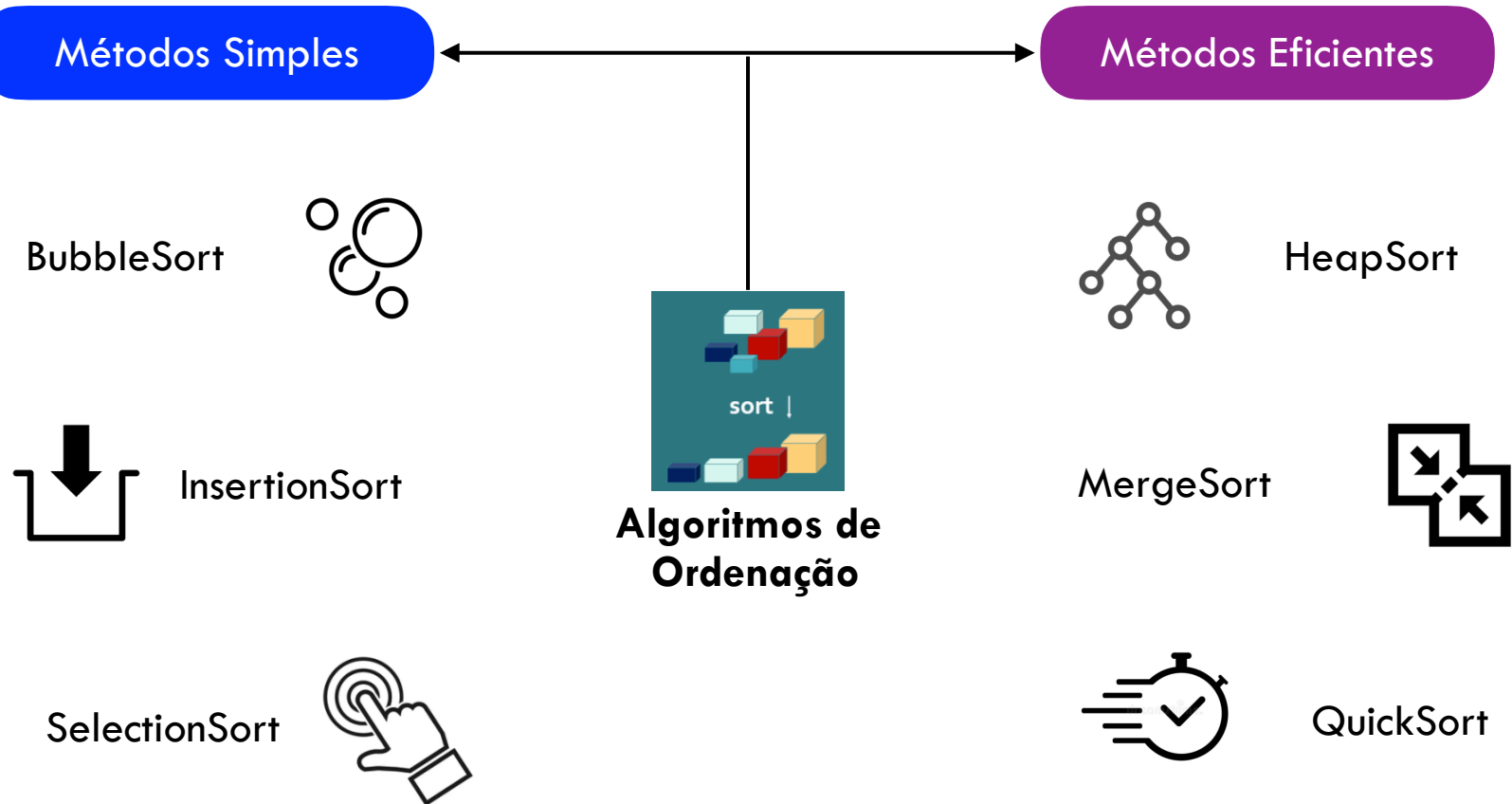
Métodos Simples

Métodos Eficientes

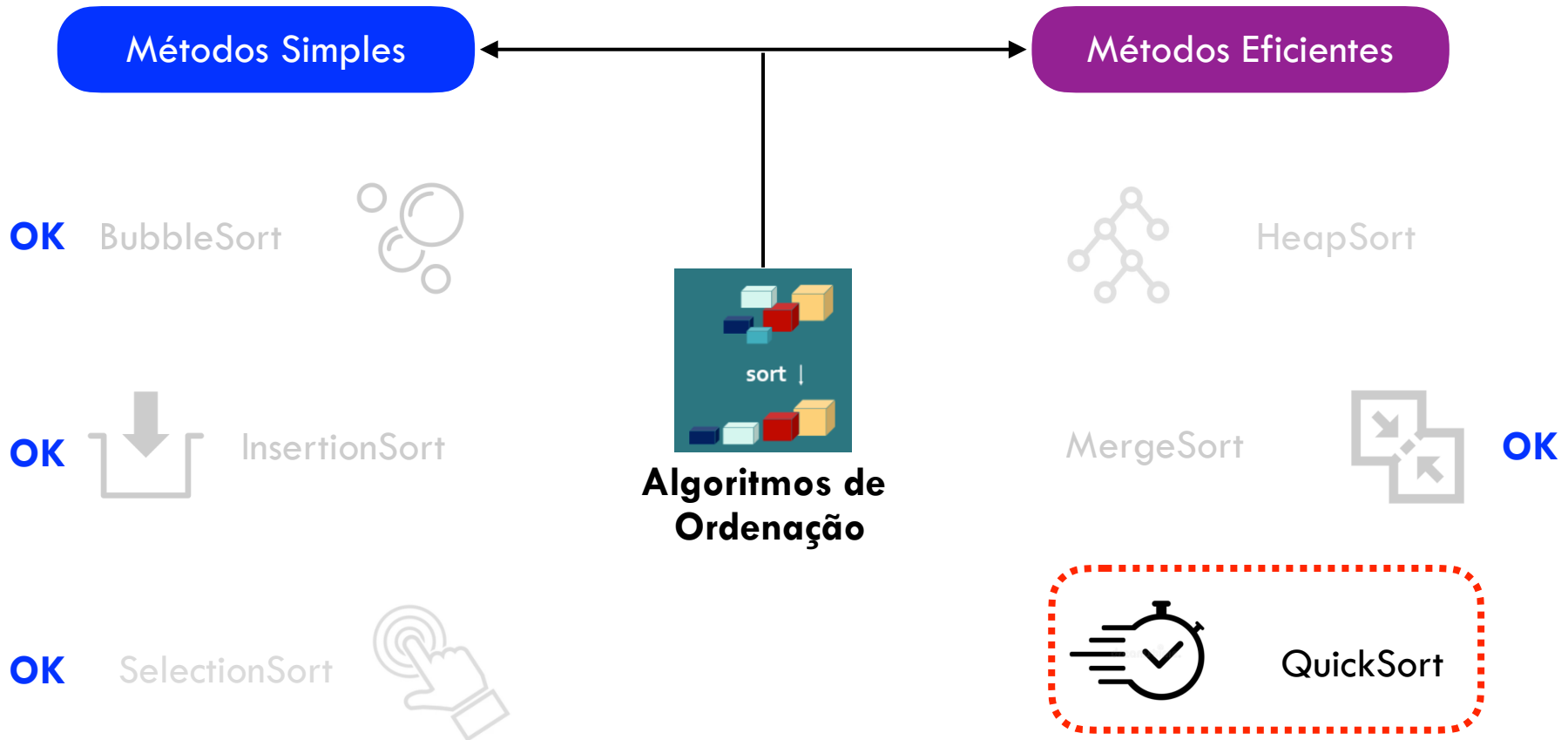


**Algoritmos de
Ordenação**

Introdução



Introdução



Roteiro



- 1 Introdução
- 2 Quick Sort
- 3 Exemplo
- 4 Exercício
- 5 Referências

Quick Sort

□ Ordenação por Troca de Partições

- * ideia básica: **dividir e conquistar**
- * divide recursivamente o conjunto de dados até que cada subconjunto possua um elemento

Quick Sort

□ **Funcionamento**

* Dividir e conquistar:

1. Um elemento é escolhido como **pivô**

Quick Sort

□ Funcionamento

* Dividir e conquistar:

1. Um elemento é escolhido como **pivô**
2. Função auxiliar chamada **particionar**: os dados são rearranjados

Quick Sort

□ Funcionamento

* Dividir e conquistar:

1. Um elemento é escolhido como **pivô**
2. Função auxiliar chamada **particionar**: os dados são rearranjados
 - a. valores menores que o pivô são colocados antes dele
 - b. valores maiores que o pivô são colocados depois dele

Quick Sort

□ Funcionamento

* Dividir e conquistar:

1. Um elemento é escolhido como **pivô**
2. Função auxiliar chamada **particionar**: os dados são rearranjados
 - a. valores menores que o pivô são colocados antes dele
 - b. valores maiores que o pivô são colocados depois dele
3. Recursivamente ordena as duas partições

Quick Sort

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

QuickSort(V, 0, 6)

Quick Sort

- Encontrar um pivô:

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

pivô
(pivô = início)

QuickSort(V, 0, 6)

Quick Sort

- Posicionar o pivô

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

pivô

0	1	2	3	4	5	6
-8	4	21	23	90	54	67

pivô

QuickSort(V, 0, 6)

Quick Sort

- Posicionar o pivô

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

pivô

0	1	2	3	4	5	6
-8	4	21	23	90	54	67

A red dashed line encloses the elements at indices 0, 1, and 2 (-8, 4, 21). A blue dotted arrow points from the pivot (23 at index 3) to the first element of the dashed box (-8 at index 0). The word "pivô" is written in red below the pivot element.

QuickSort(V, 0, 6)

Valores menores que o pivô ficam à esquerda

Quick Sort

- Posicionar o pivô

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

pivô

0	1	2	3	4	5	6
-8	4	21	23	90	54	67

A red dashed box encloses the elements from index 3 to 6 (23, 90, 54, 67). A blue dotted arrow points from the pivot 23 at index 3 to the element -8 at index 0.

pivô

Valores maiores que o pivô ficam à direita

Quick Sort

- Chamar recursivamente, desconsiderando o pivô:

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

pivô

QuickSort(V, 0, 6)

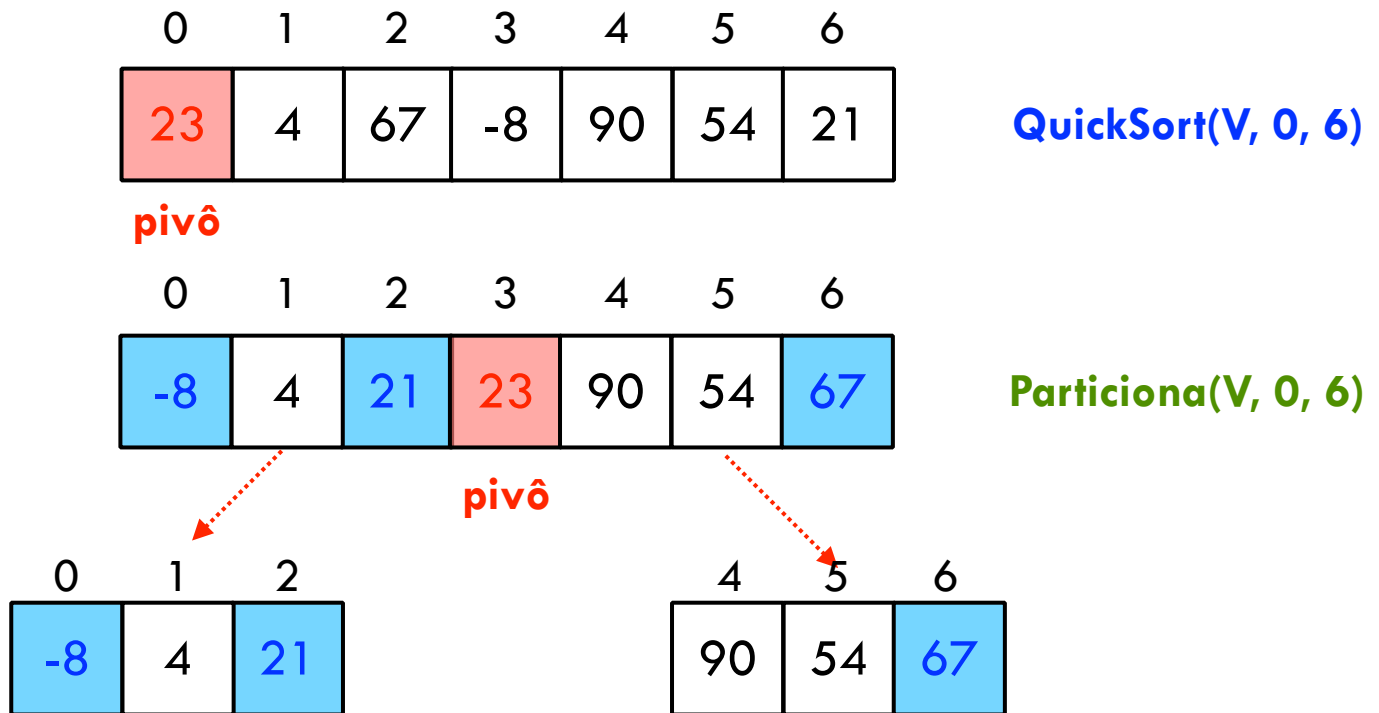
0	1	2	3	4	5	6
-8	4	21	23	90	54	67

pivô



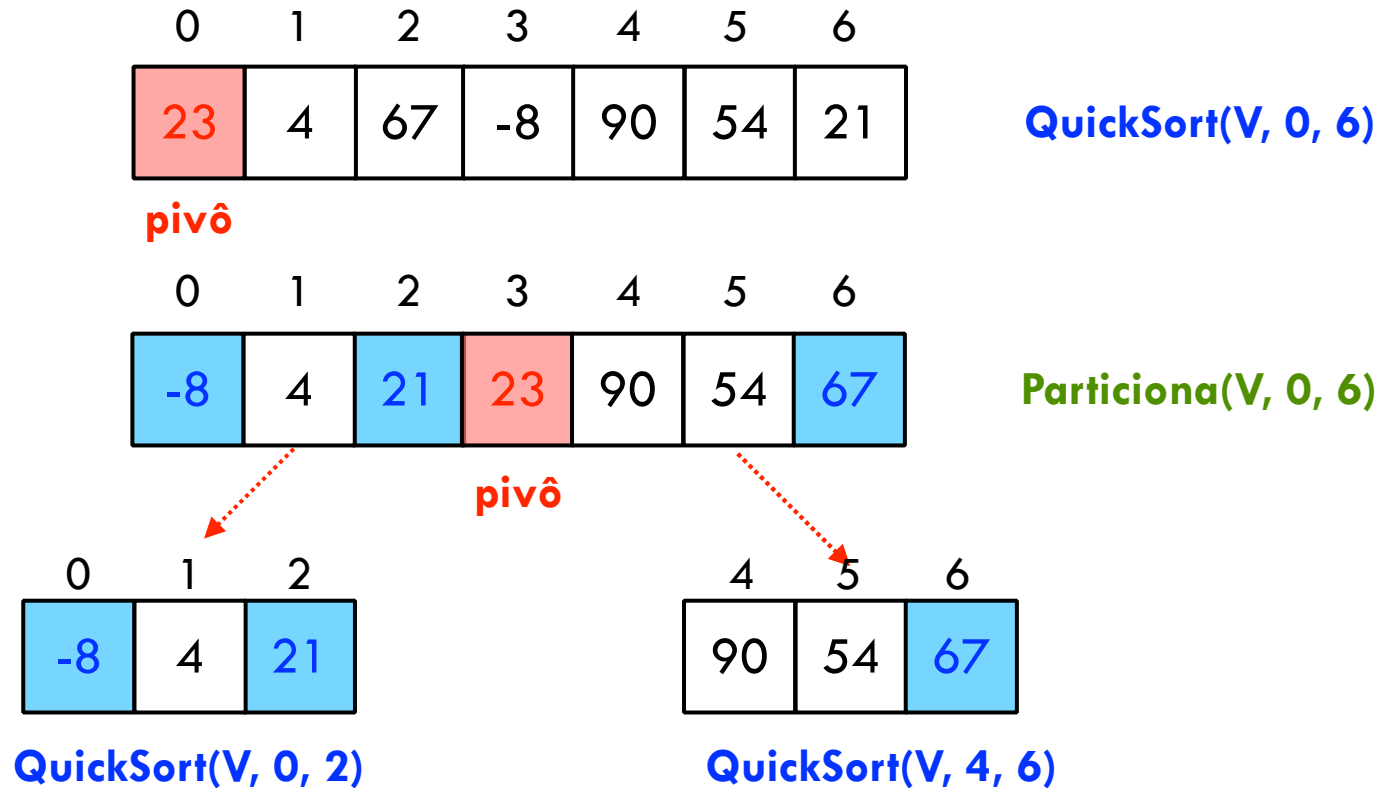
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



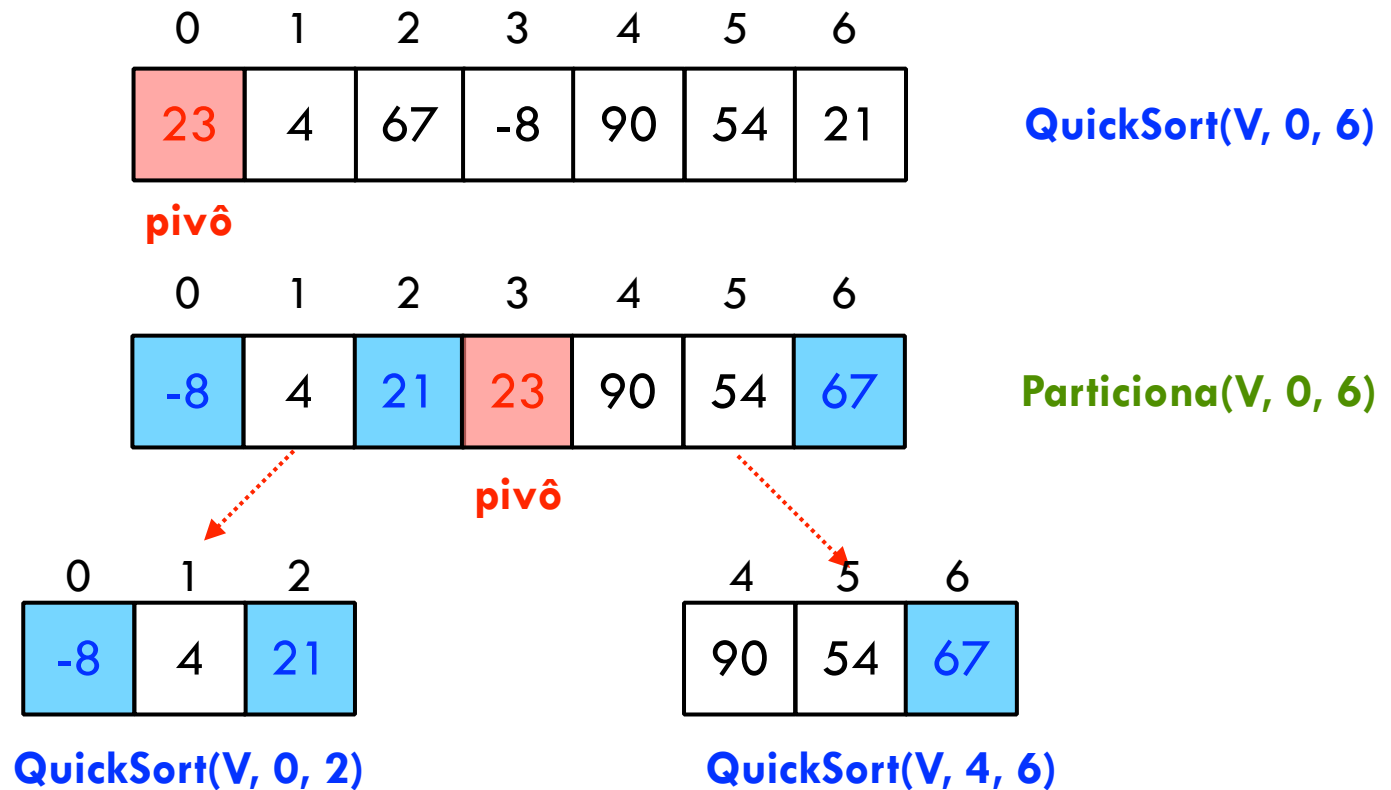
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



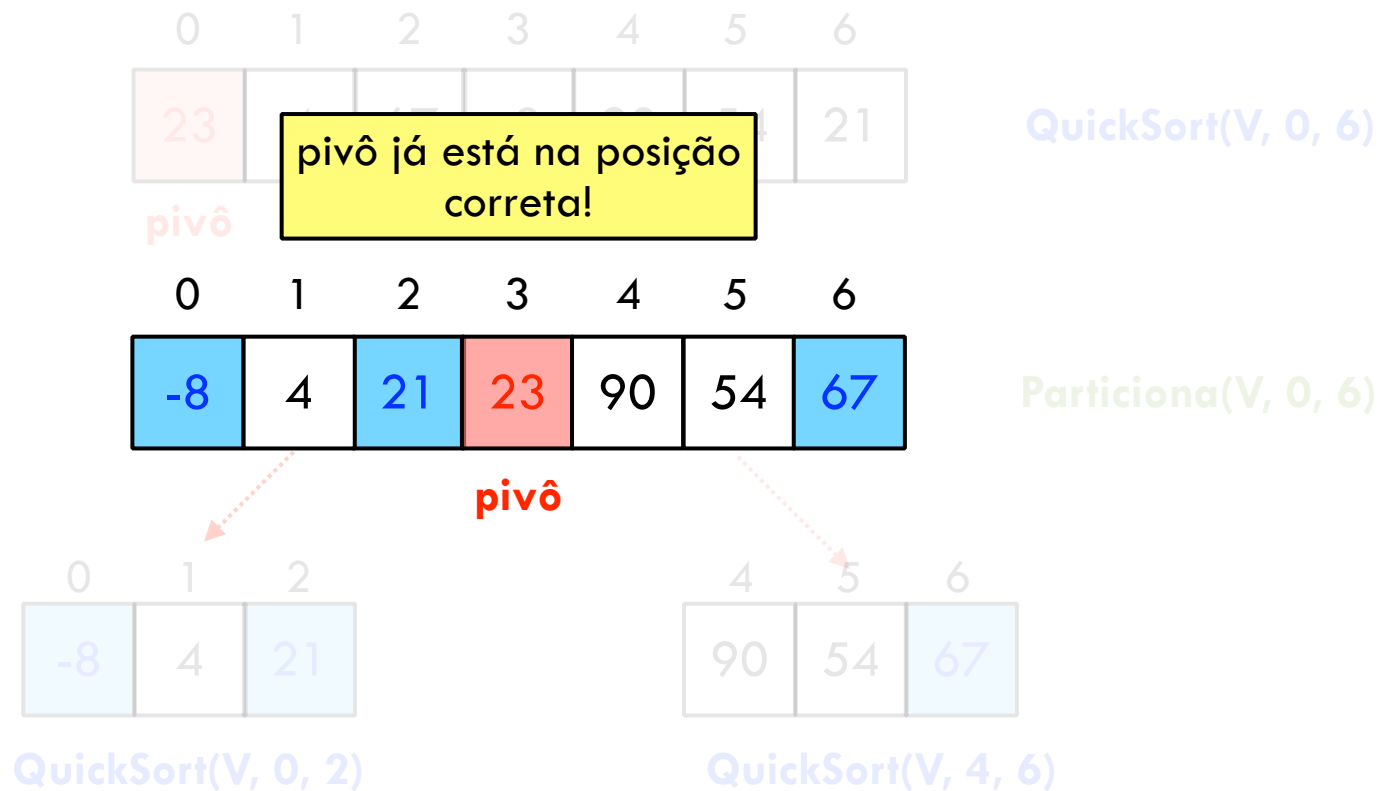
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



Quick Sort

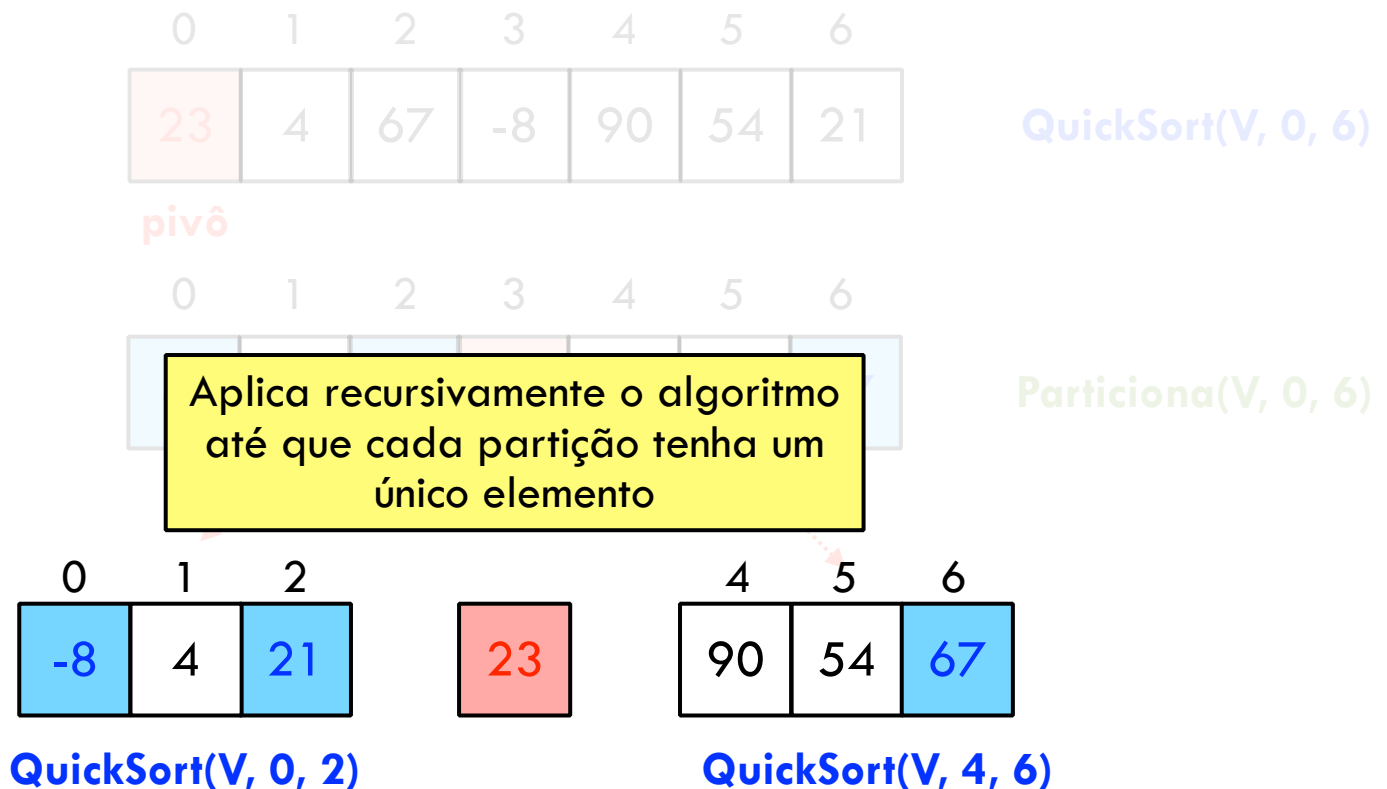
- Chamar recursivamente, desconsiderando o pivô:



...

Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



...

Quick Sort

□ Desempenho

- * **melhor caso:** $O(N \log N)$
- * **pior caso:** $O(N^2)$, mas é muito raro
- * **caso médio:** $O(N \log N)$

Quick Sort

□ Desempenho

- * **melhor caso:** $O(N \log N)$
- * **pior caso:** $O(N^2)$, mas é muito raro
- * **caso médio:** $O(N \log N)$

Obs: O pior caso do Quick Sort ocorre quando o vetor já está ordenado.
Nesse caso, a complexidade do Insertion Sort é $O(N)$

Quick Sort

□ Pseudocódigo

1. **QuickSort:** divide os dados em vetores cada vez menores
2. **Particiona:** elege um pivô e particiona de maneira que ...
 - * todos os elementos menores que o pivô estão antes dele
 - * todos os elementos maiores que o pivô estão depois dele

Quick Sort

□ Pseudocódigo (função principal)

1. QuickSort (V, Inicio, Fim)
2. Se (Inicio < Fim), então:
3. Pivo = Particiona(V, Inicio, Fim)
4. QuickSort(V, Inicio, Pivo-1)
5. QuickSort(V, Pivo+1, Fim)

Quick Sort

□ Pseudocódigo (função auxiliar)

```
1.  Particiona (V, Inicio, Fim)
2.      Esq = Inicio
3.      Dir  = Fim
4.      Pivo = V[Inicio]
5.      Enquanto (Esq < Dir) faça:
6.          Enquanto (V[Esq] ≤ Pivo & Esq ≤ final) faça:
7.              Incrementa Esq
8.          Enquanto (V[Dir] > Pivo & Dir ≥ Inicio) faça:
9.              Decrementa Dir
10.         Se Esq < Dir então:
11.             Troca V[Esq] e V[Dir]
12.         Troca V[Dir] com V[Inicio]
13.     Return (Dir)
```

Roteiro



- 1 Introdução
- 2 Quick Sort
- 3 Exemplo
- 4 Exercícios
- 5 Referências

Exemplo

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

vetor não ordenado

Exemplo

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

QuickSort(V, 0, 6)

Exemplo

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

pivô

QuickSort(V, 0, 6)

Particiona(V, 0, 6)

Exemplo

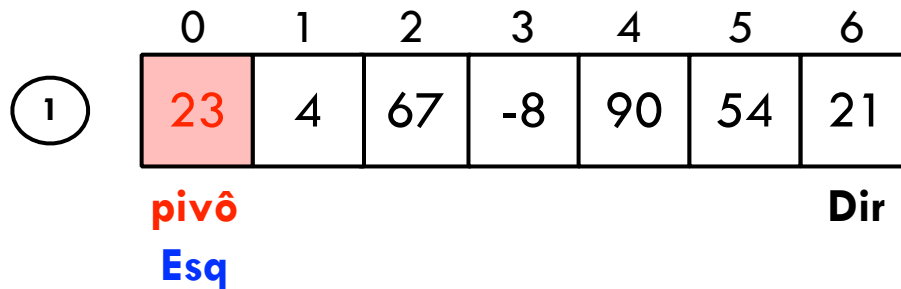
Particiona(V, 0, 6)

	0	1	2	3	4	5	6
①	23	4	67	-8	90	54	21
	pivô						Dir

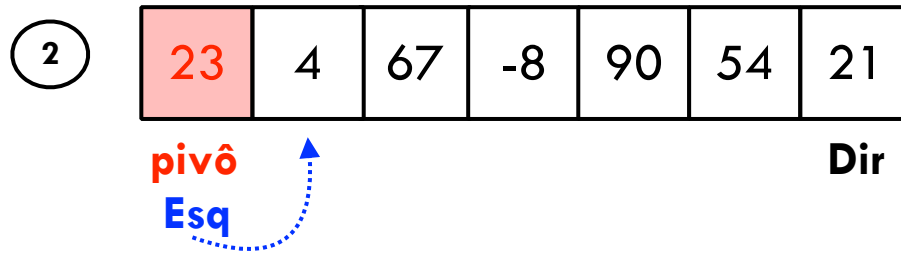
Esq = Inicio = 0
Dir = Fim = 6
pivô = v[Inicio] = 23

Exemplo

Particiona(V, 0, 6)



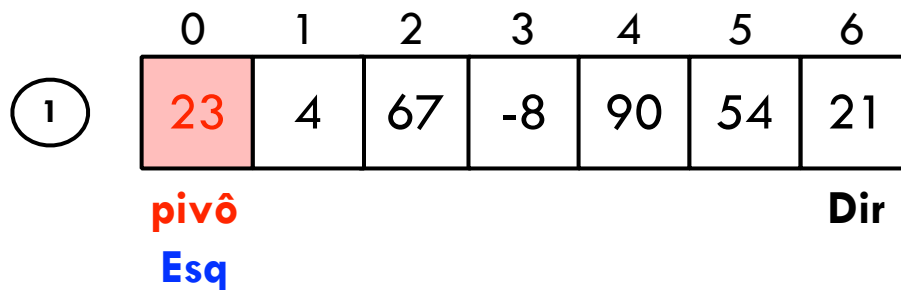
Esq = Inicio = 0
Dir = Fim = 6
pivô = v[Inicio] = 23



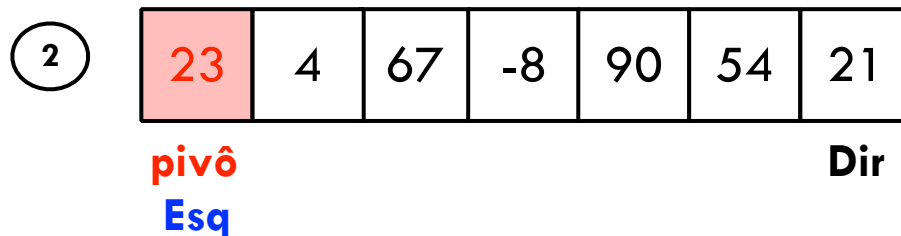
$V[Esq] \leq \text{Pivô} : Esq++$

Exemplo

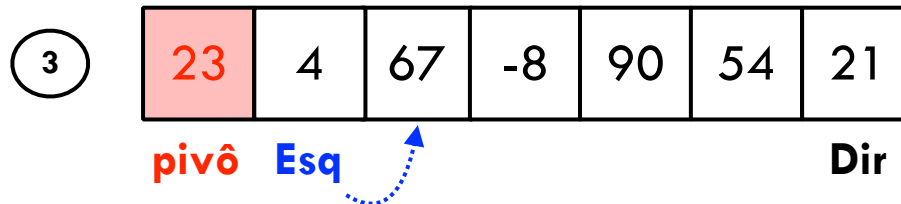
Particiona(V, 0, 6)



Esq = Inicio = 0
Dir = Fim = 6
pivô = v[Inicio] = 23



V[Esq] <= Pivô: Esq++



V[Esq] <= Pivô: Esq++

Exemplo

Particiona(V, 0, 6)

	0	1	2	3	4	5	6
④	23	4	67	-8	90	54	21
	pivô		Esq				Dir

$V[\text{Esq}] > \text{Pivô}$: Comparar Dir

Exemplo

Particiona(V, 0, 6)

	0	1	2	3	4	5	6
④	23	4	67	-8	90	54	21
	pivô		Esq				Dir

$V[\text{Esq}] > \text{Pivô}$: Comparar Dir

	0	1	2	3	4	5	6
⑤	23	4	67	-8	90	54	21
	pivô		Esq				Dir

$V[\text{Dir}] < \text{Pivô}$

$\text{Esq} < \text{Dir}$: Trocar $V[\text{Esq}]$, $V[\text{Dir}]$

Exemplo

Particiona(V, 0, 6)

	0	1	2	3	4	5	6
4	23	4	67	-8	90	54	21
	pivô		Esq				Dir

$V[Esq] > Pivô$: Comparar Dir

	0	1	2	3	4	5	6
5	23	4	67	-8	90	54	21
	pivô		Esq				Dir

$V[Dir] < Pivô$

Esq < Dir: Trocar $V[Esq]$, $V[Dir]$

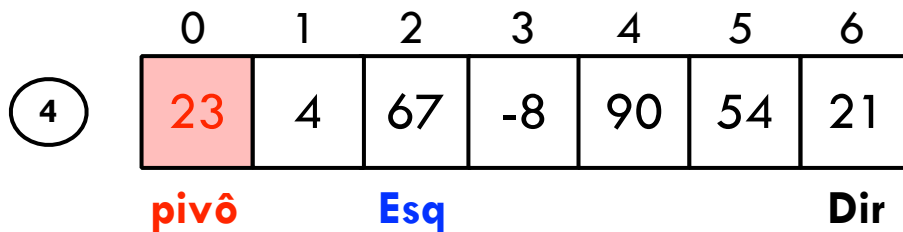
	0	1	2	3	4	5	6
6	23	4	21	-8	90	54	67
	pivô		Esq				Dir

Esq < Dir? Sim

continua o while (iteração)

Exemplo

Particiona(V, 0, 6)



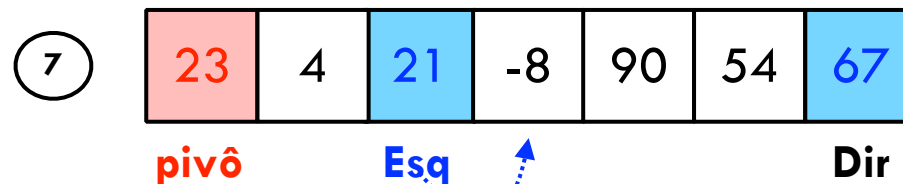
$V[Esq] > Pivô$: Comparar Dir



$V[Dir] < Pivô$
 $Esq < Dir$: Trocar $V[Esq]$, $V[Dir]$



$Esq < Dir$? Sim
continua o while (iteração)



$V[Esq] \leq Pivô$: $Esq++$

Exemplo

Particiona(V, 0, 6)



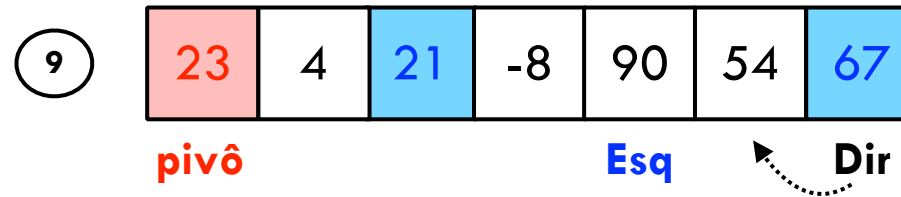
$V[\text{Esq}] > \text{Pivô}$: Comparar Dir

Exemplo

Particiona(V, 0, 6)



$V[\text{Esq}] > \text{Pivô}$: Comparar Dir



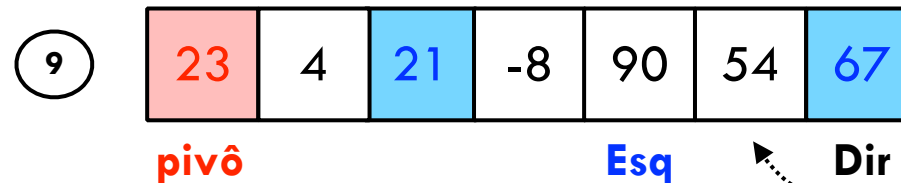
$V[\text{Dir}] > \text{Pivô}$: Dir--

Exemplo

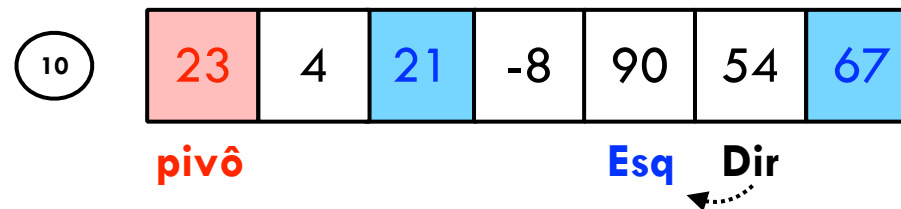
Particiona(V, 0, 6)



$V[\text{Esq}] > \text{Pivô}$: Comparar Dir



$V[\text{Dir}] > \text{Pivô}$: Dir--



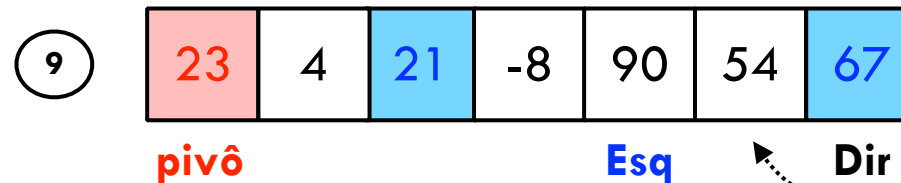
$V[\text{Dir}] > \text{Pivô}$: Dir--

Exemplo

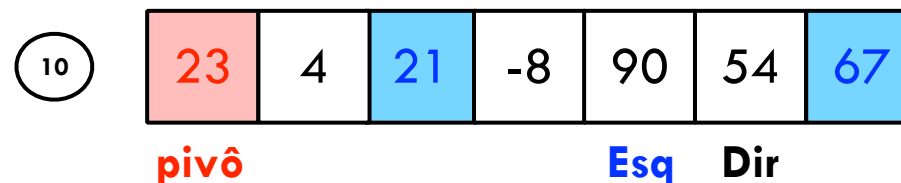
Particiona(V, 0, 6)



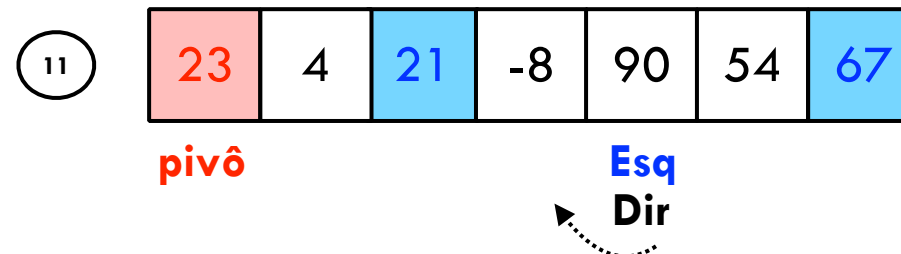
$V[Esq] > Pivô$: Comparar Dir



$V[Dir] > Pivô$: Dir--



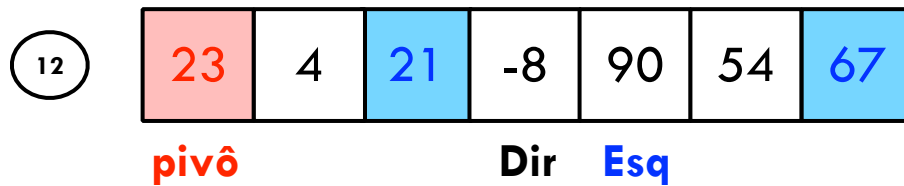
$V[Dir] > Pivô$: Dir--



$V[Dir] > Pivô$: Dir--

Exemplo

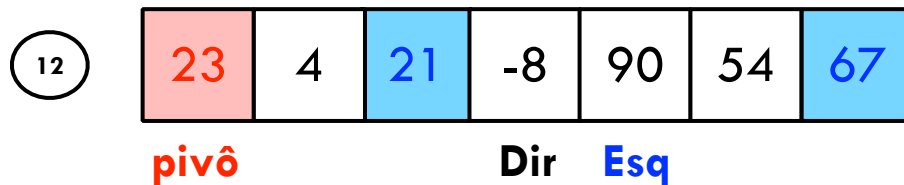
Particiona(V, 0, 6)



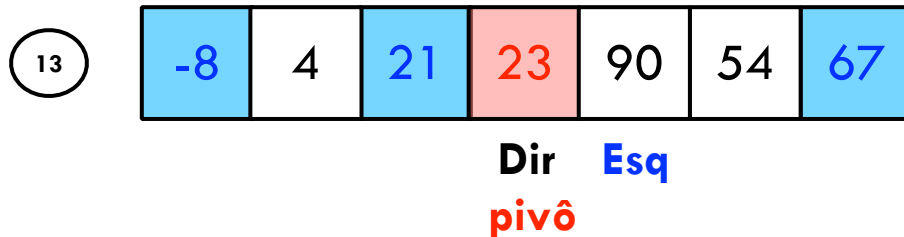
$V[Dir] < Pivô$
Dir < Esq: terminar while

Exemplo

Particiona(V, 0, 6)



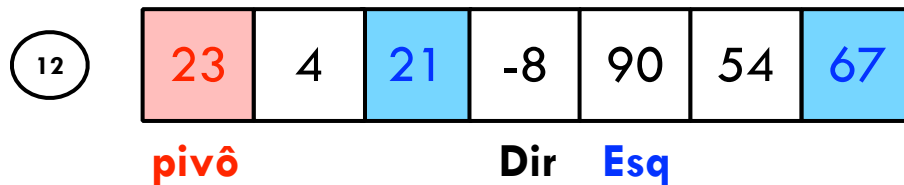
$V[Dir] < \text{Pivô}$
 $Dir < \text{Esq}$: terminar while



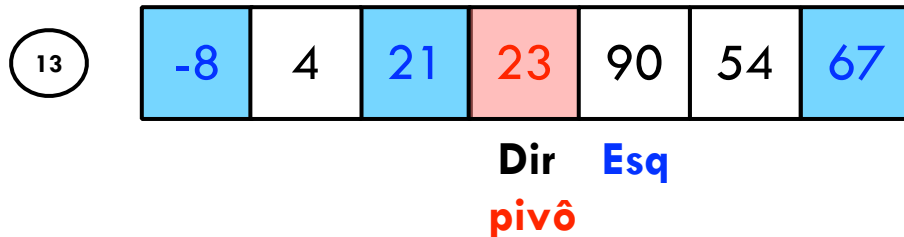
Trocar $V[Dir]$ e $V[\text{Inicio}]$

Exemplo

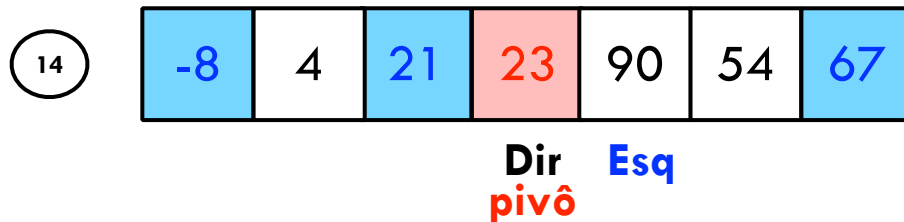
Particiona(V, 0, 6)



$V[Dir] < \text{Pivô}$
Dir < Esq: terminar while



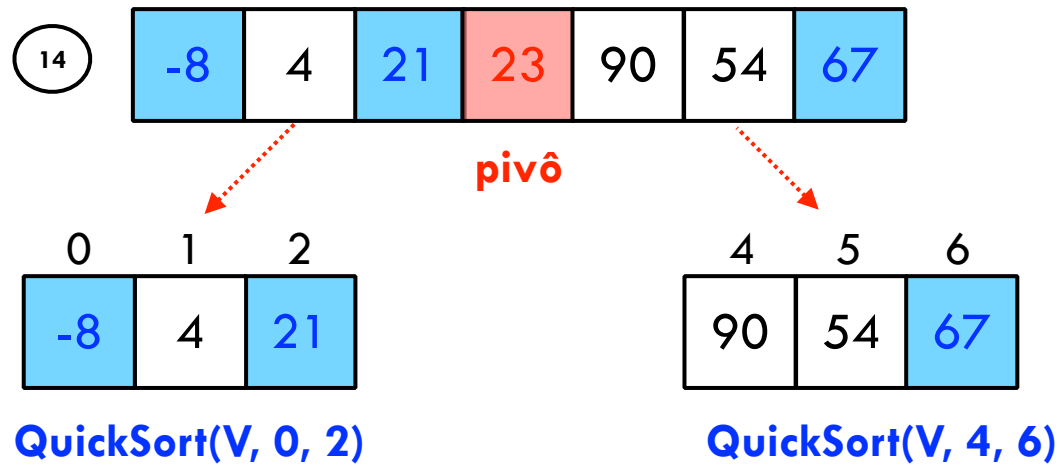
Trocar $V[Dir]$ e $V[\text{Inicio}]$
 $V[dir]$ é o novo pivô



Chamar recursivamente
para as duas partições:
QuickSort(0,2)
QuickSort(4,6)

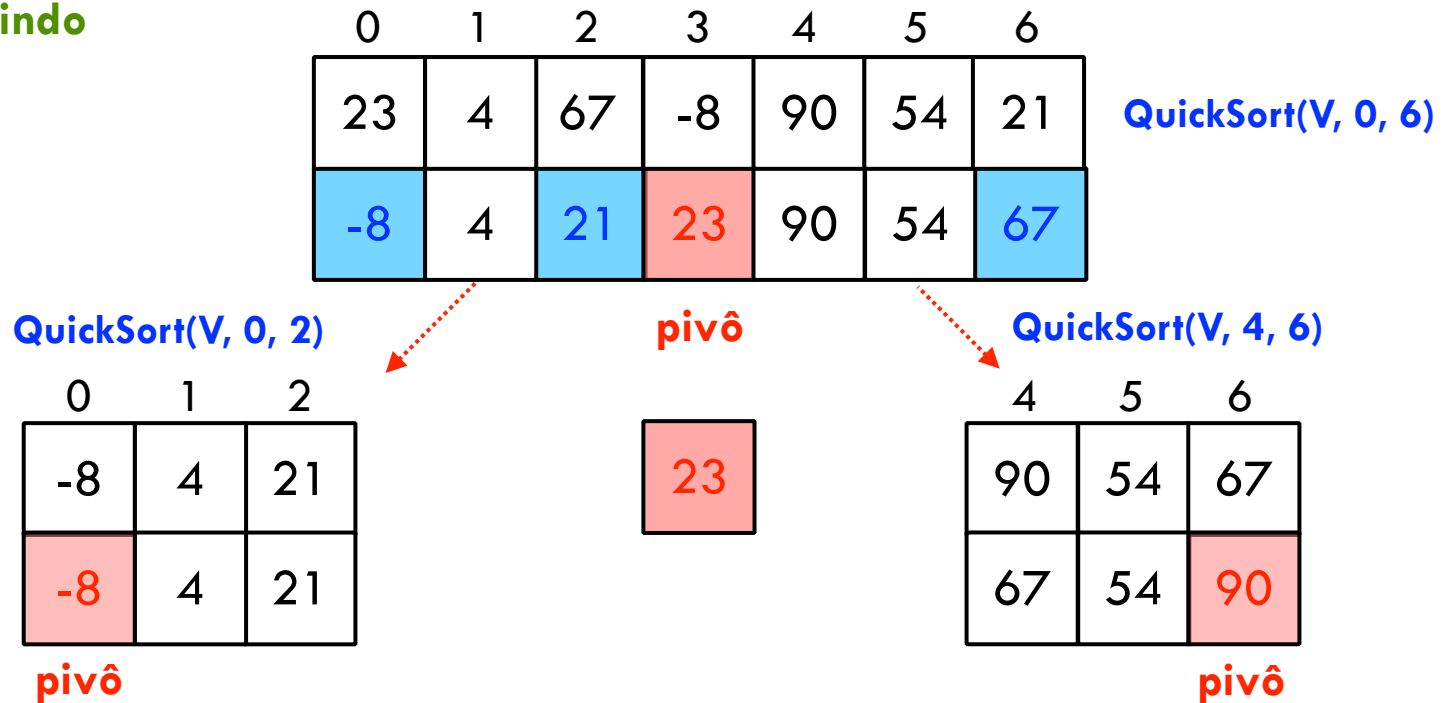
Exemplo

Particiona(V, 0, 6)

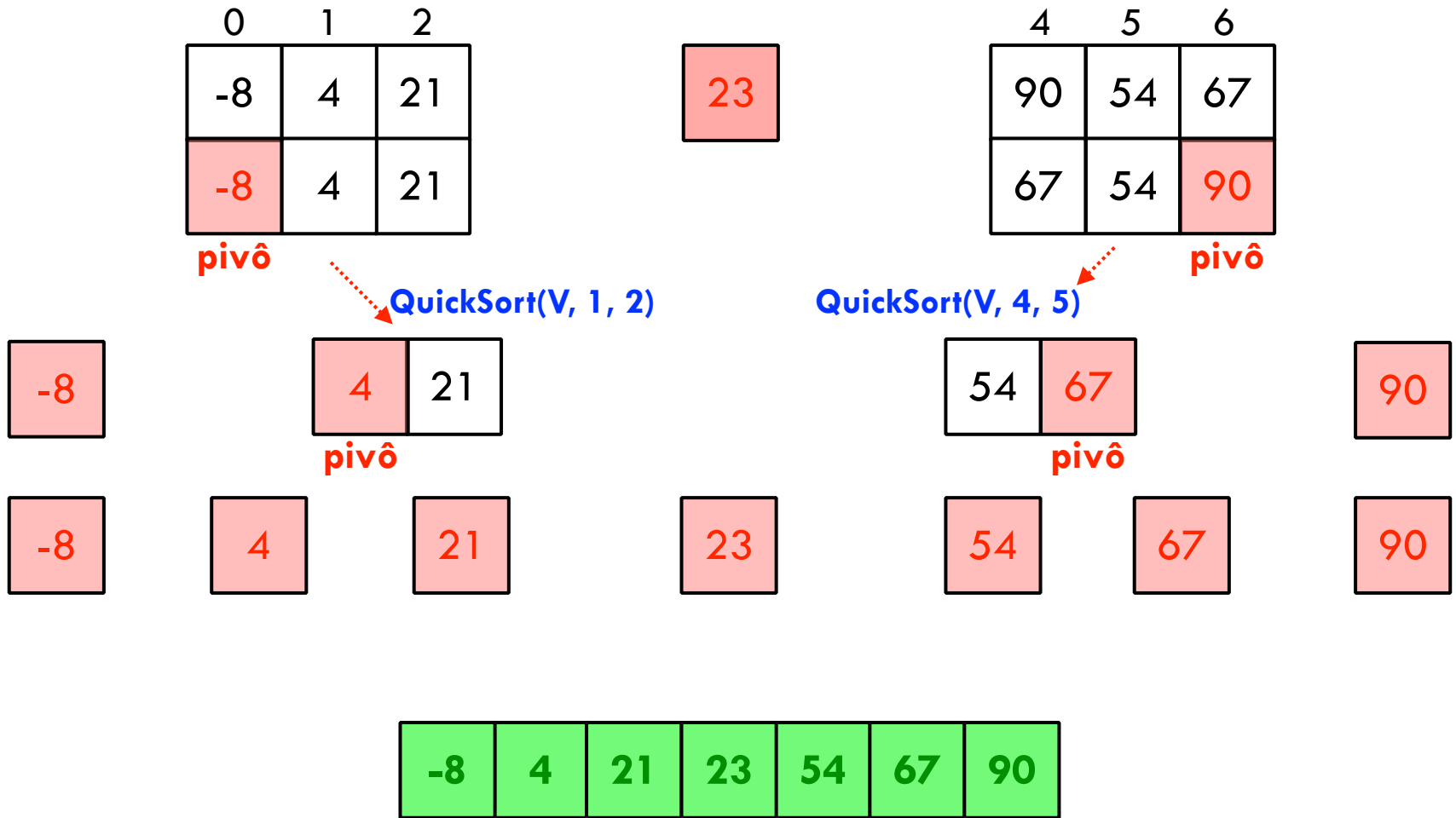


Exemplo

Resumindo



Exemplo



Vetor Ordenado

Quick Sort

Vantagens

- * elegante e eficiente
- * costuma ser a melhor opção prática para a ordenação de grandes conjuntos de dados

Desvantagens

- * Recursivo
- * não é estável
- * escolha do pivô (particionamento não balanceado)
 - partições com 0 e $n-1$ elementos

Roteiro



- 1 Introdução
- 2 Quick Sort
- 3 Exemplo
- 4 Exercícios
- 5 Referências

Exercícios



HANDS ON :)))

Exercícios

1) Implemente o **quickSort** em C considerando a seguinte assinatura de função:

```
/* Ordena o vetor usando QuickSort
```

```
Parâmetros:
```

```
v: vetor a ser ordenado
```

```
inicio: índice do primeiro elemento do vetor
```

```
fim: índice do último elemento do vetor */
```

```
void quickSort(int *v, int inicio, int fim);
```

Exercícios

2) Implemente a função auxiliar **particiona** para o correto funcionamento do algoritmo de Quick Sort.

/ Parâmetros:*

v: vetor a ser particionado

inicio: índice do primeiro elemento do vetor

fim: índice do último elemento do vetor

Retorno:

índice do elemento pivô

*Particiona um vetor. Ao final da função, todos os elementos menores que o pivô devem estar antes dele, e todos os elementos maiores depois */*

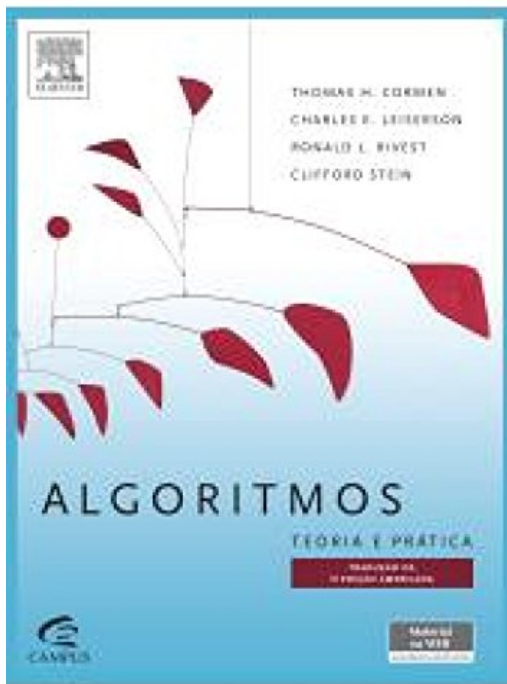
int **particiona**(**int** *v, **int** inicio, **int** fim);

Roteiro



- 1** Introdução
- 2** Quick Sort
- 3** Exemplo
- 4** Exercícios
- 5** Referências

Referências sugeridas

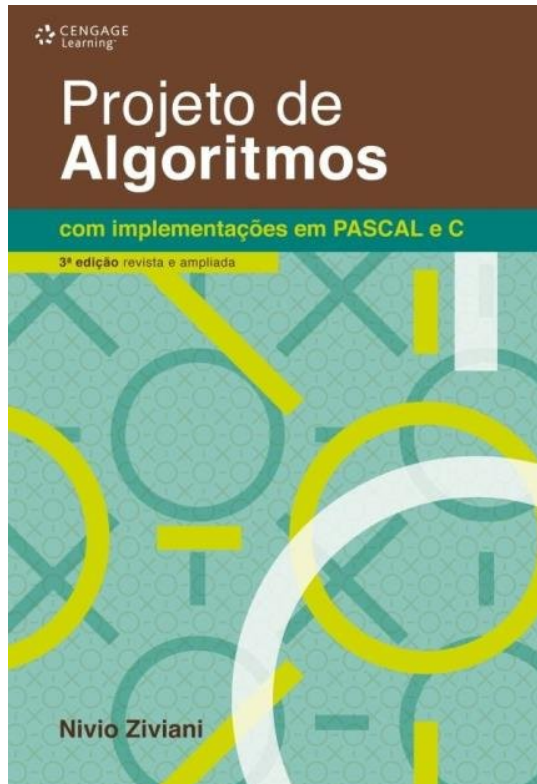


[Cormen et al, 2018]

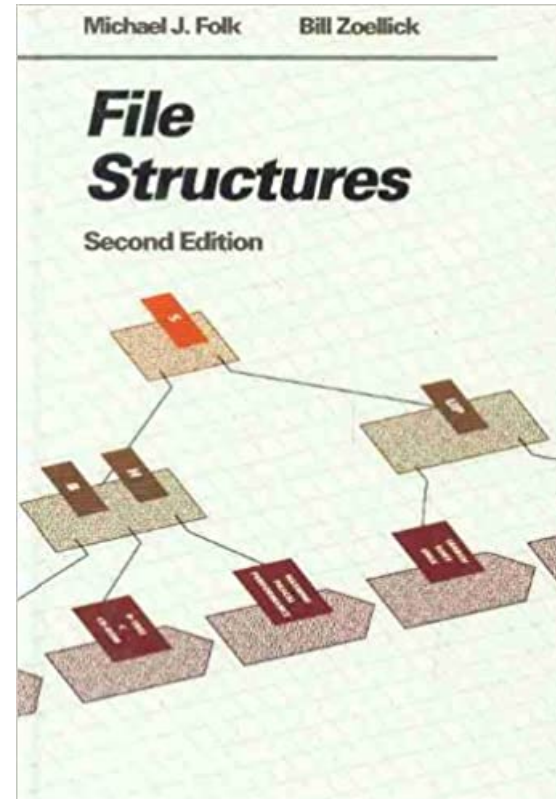


[Drozdek, 2017]

Referências sugeridas



[Ziviani, 2010]



[Folk & Zoellick, 1992]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br