EDCO4B
ESTRUTURAS DE DADOS 2

Aula 06 - Quick Sort

Prof. Rafael G. Mantovani

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Apucarana - PR, Brasil

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia de Computação

Licença

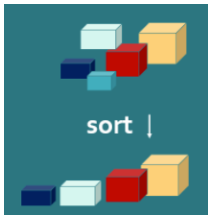
Roteiro

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

Roteiro

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

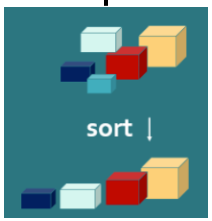
Introdução



Algoritmos de Ordenação

Introdução

Métodos Simples ⟷ Métodos Eficientes



Algoritmos de Ordenação

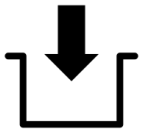
Introdução

Métodos Simples

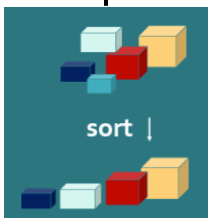
Métodos Eficientes

BubbleSort

InsertionSort



SelectionSort



Algoritmos de Ordenação

HeapSort

MergeSort

QuickSort

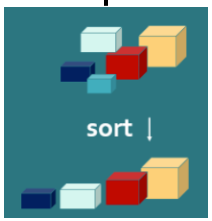
Introdução

Métodos Simples

Métodos Eficientes

OK BubbleSort

HeapSort



Algoritmos de Ordenação

OK InsertionSort

MergeSort OK

OK SelectionSort

QuickSort

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

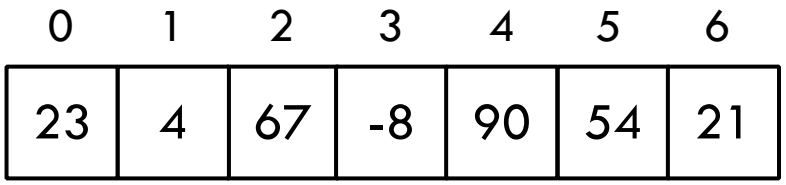
Quick Sort

- **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

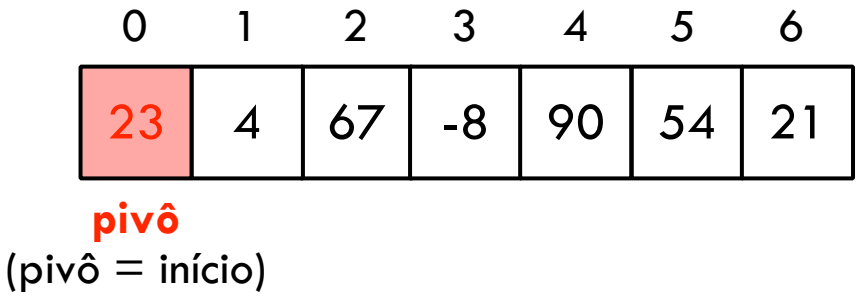
Quick Sort



QuickSort(V, 0, 6)

Quick Sort

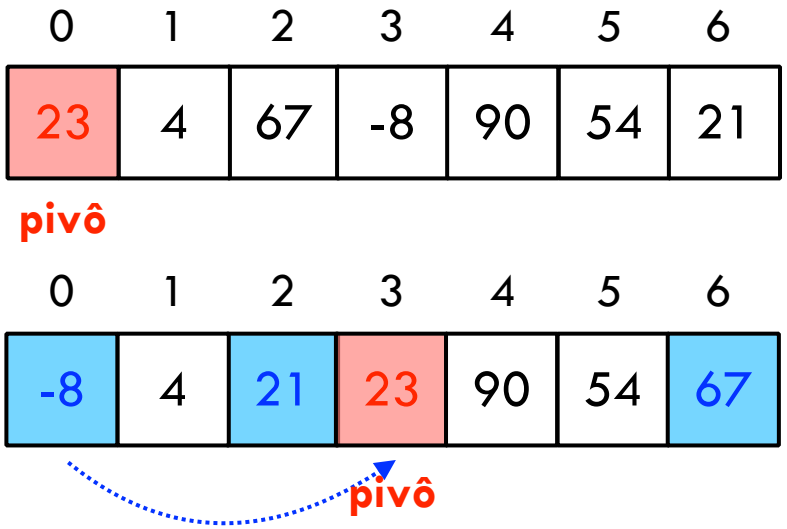
- Encontrar um pivô:



QuickSort(V, 0, 6)

Quick Sort

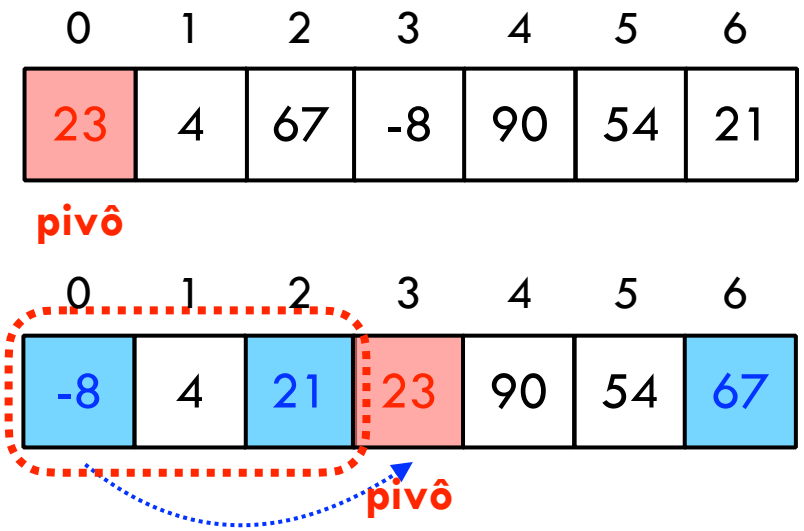
- Posicionar o pivô



QuickSort(V, 0, 6)

Quick Sort

- Posicionar o pivô

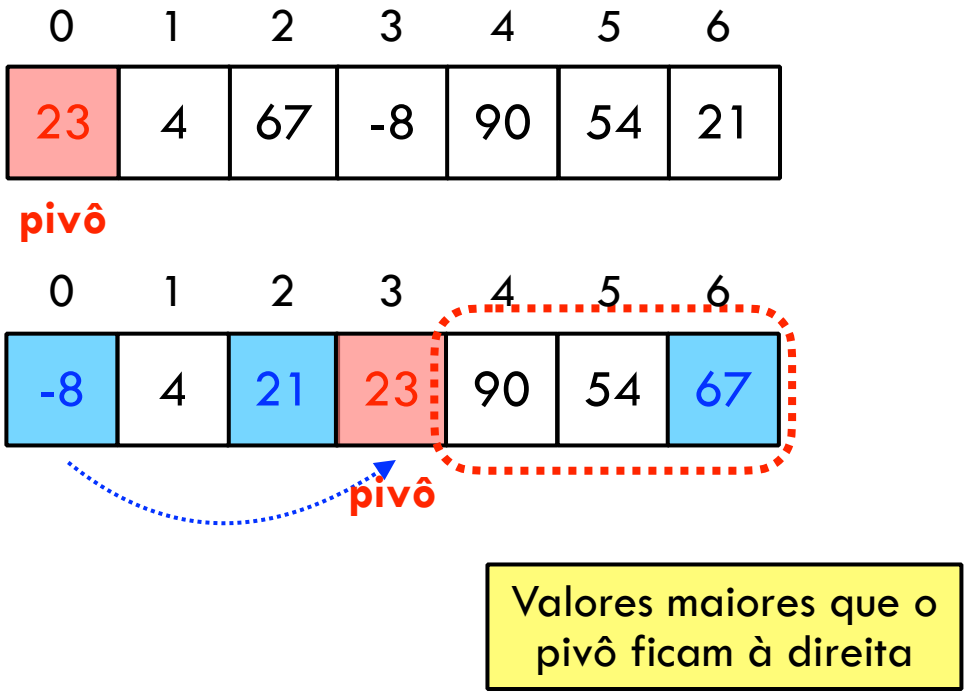


QuickSort(V, 0, 6)

Valores menores que o pivô ficam à esquerda

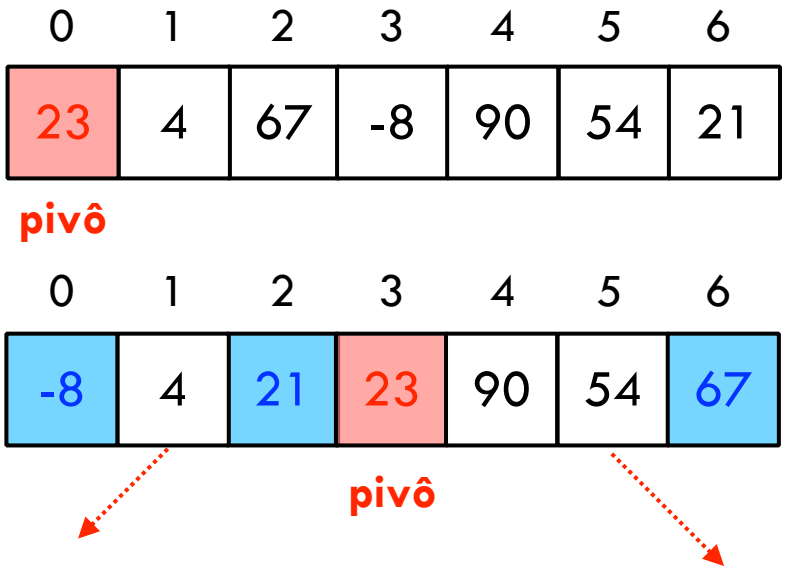
Quick Sort

- Posicionar o pivô



Quick Sort

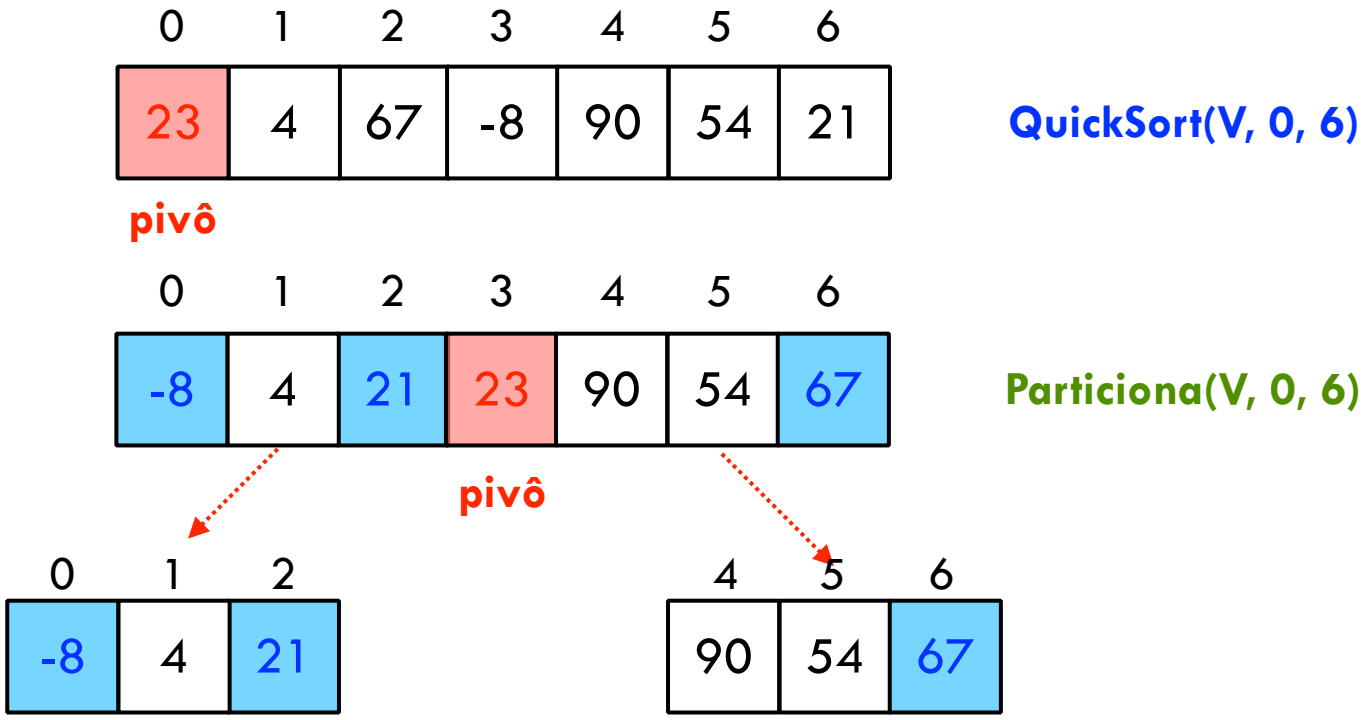
- Chamar recursivamente, desconsiderando o pivô:



QuickSort(V, 0, 6)

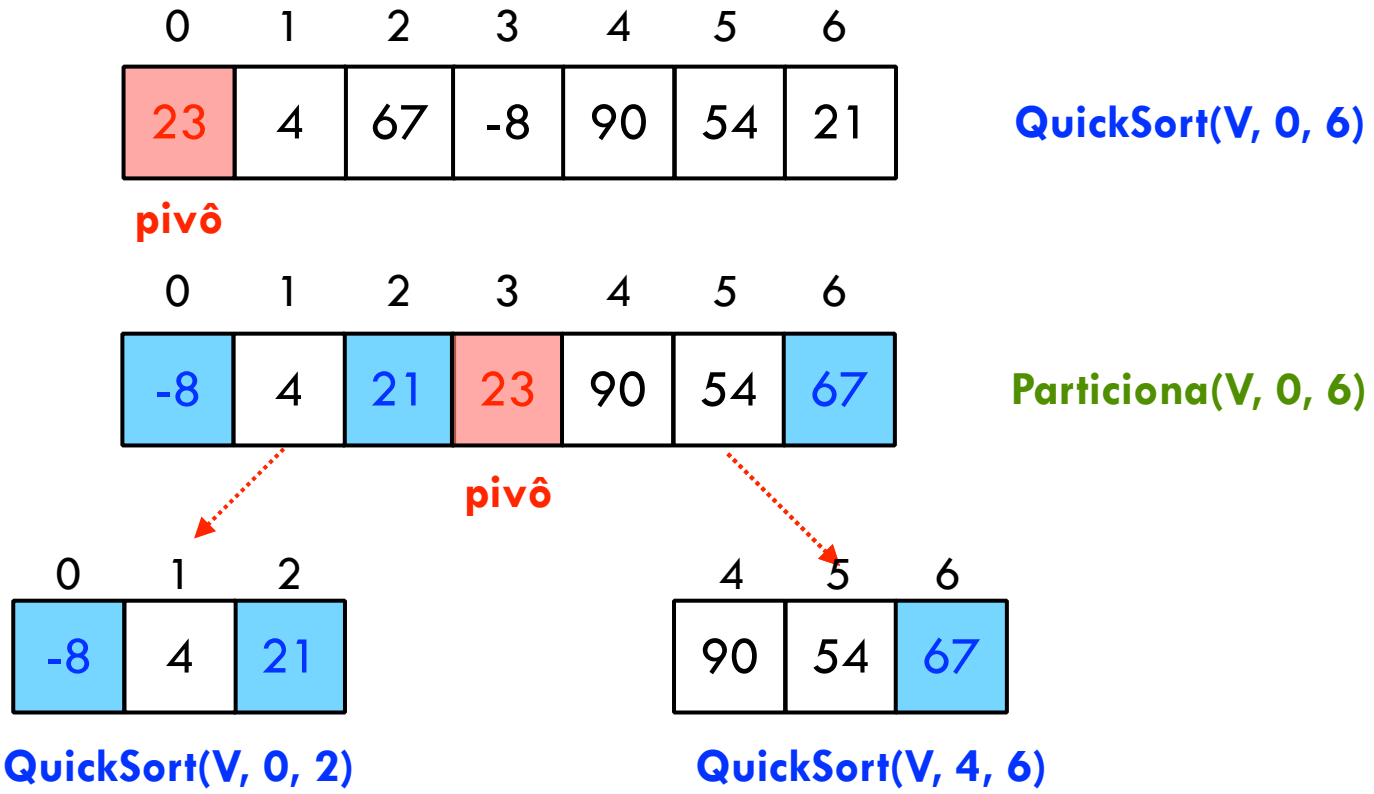
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



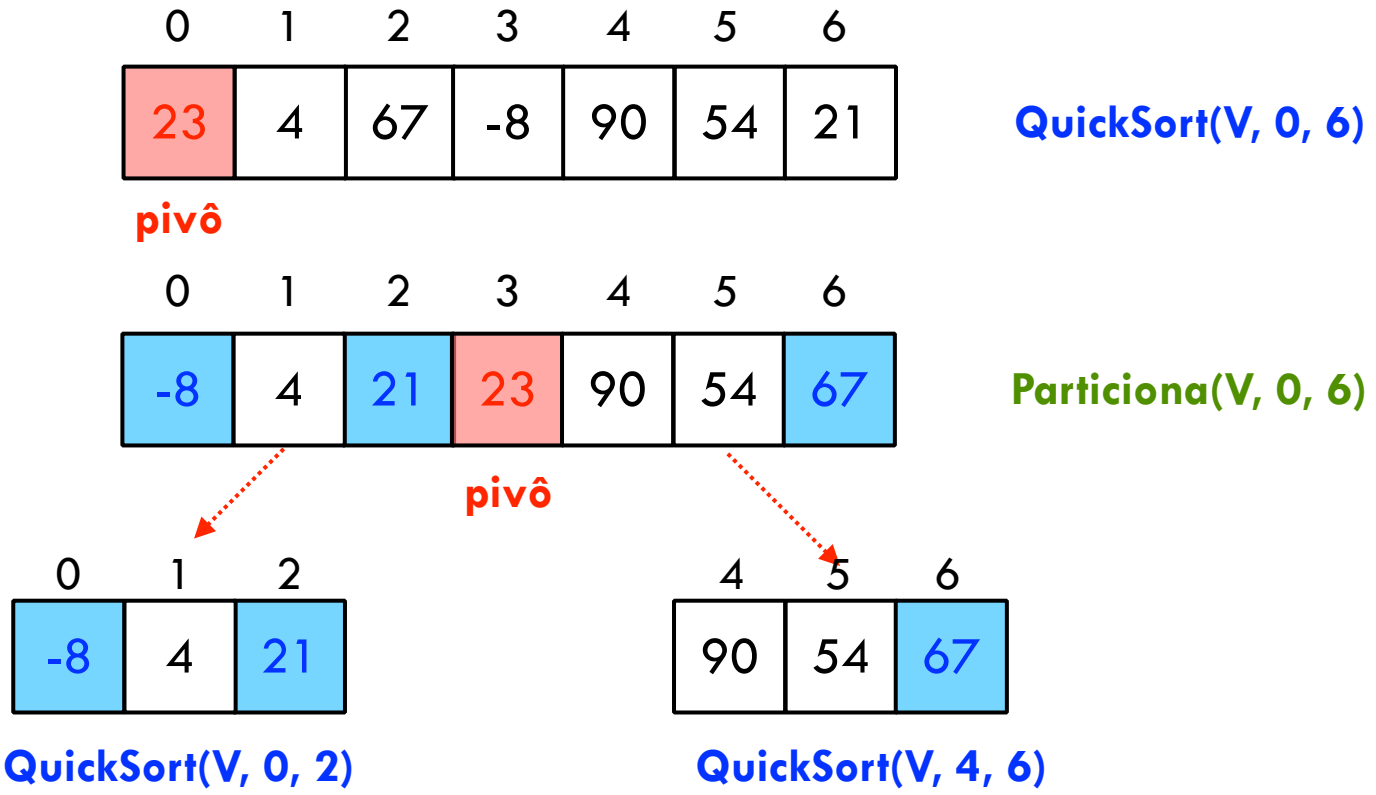
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



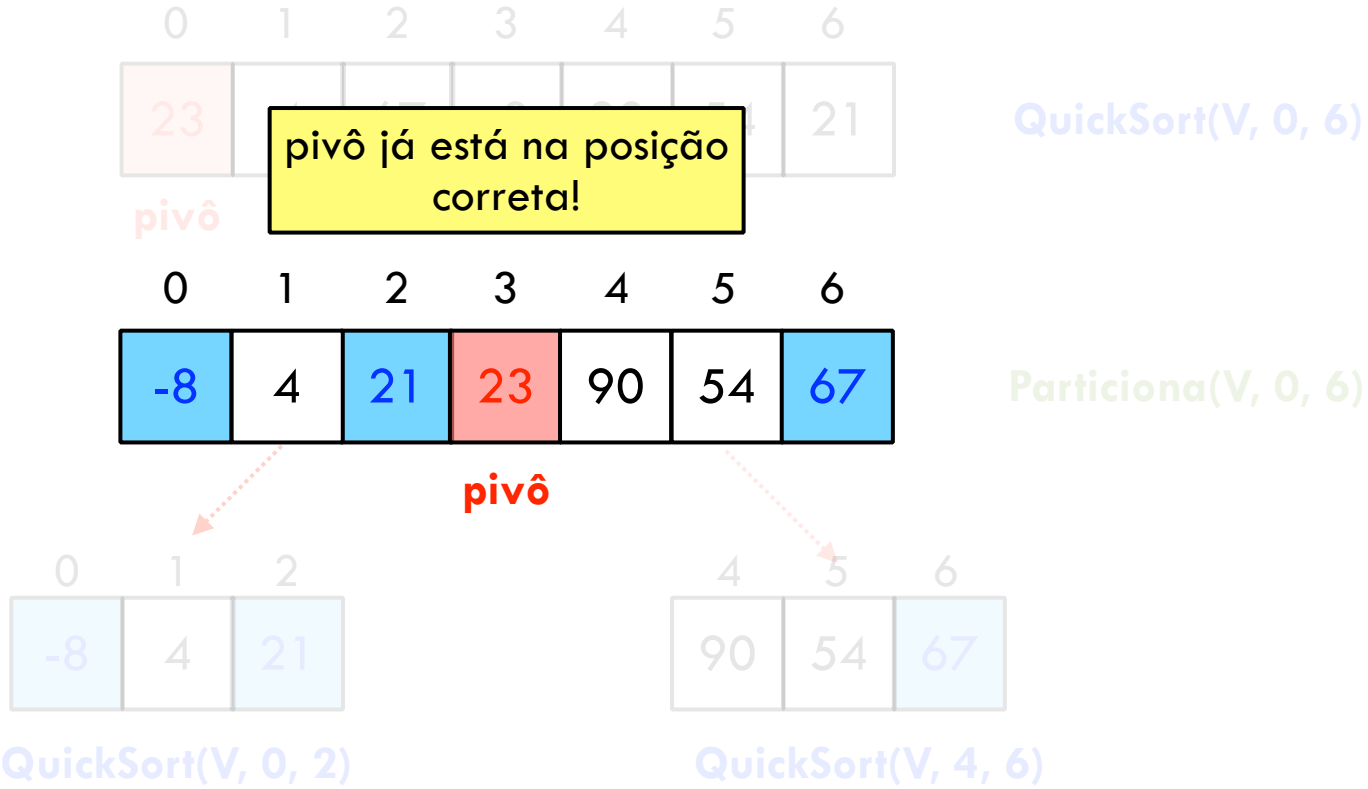
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



Quick Sort

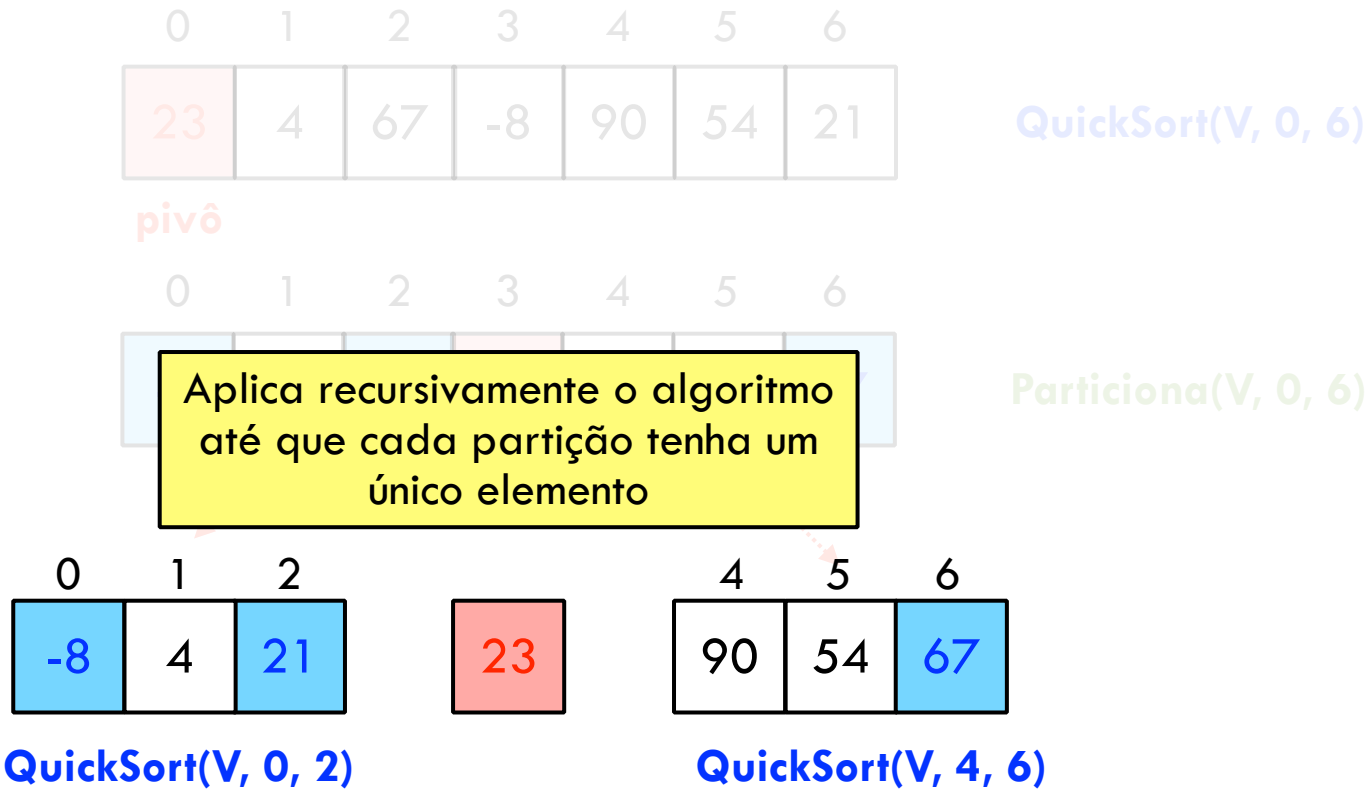
- Chamar recursivamente, desconsiderando o pivô:



...

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

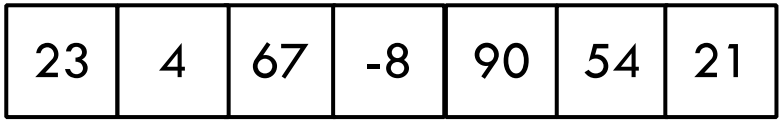
- **Pseudocódigo** (função auxiliar)

```
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

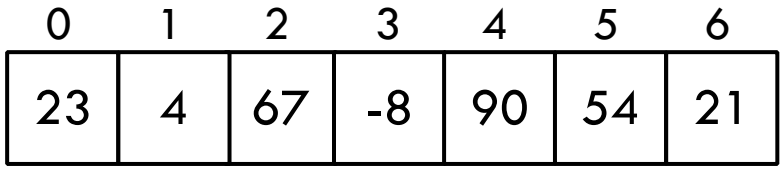
1. Introdução
2. Quick Sort
3. Exemplo
4. Exercícios
5. Referências

Exemplo



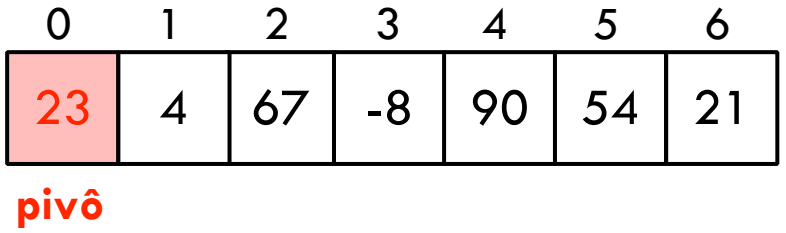
vetor não ordenado

Exemplo



QuickSort(V, 0, 6)

Exemplo

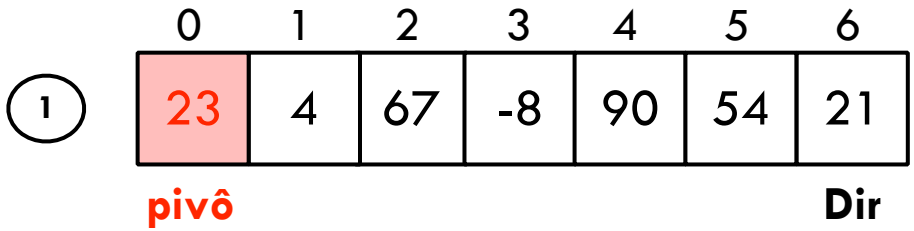


QuickSort(V, 0, 6)

Particiona(V, 0, 6)

Exemplo

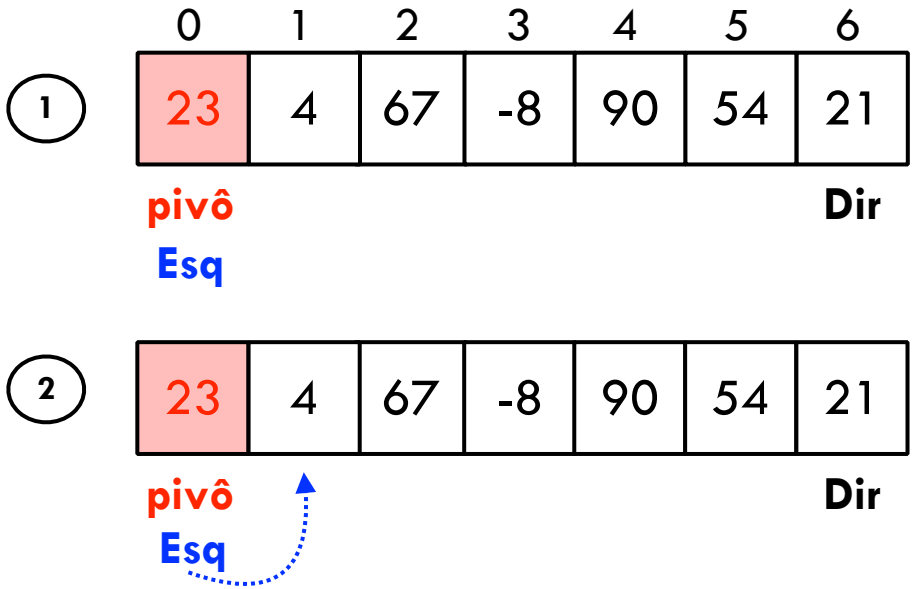
Particiona(V, 0, 6)



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

Exemplo

Particiona(V, 0, 6)

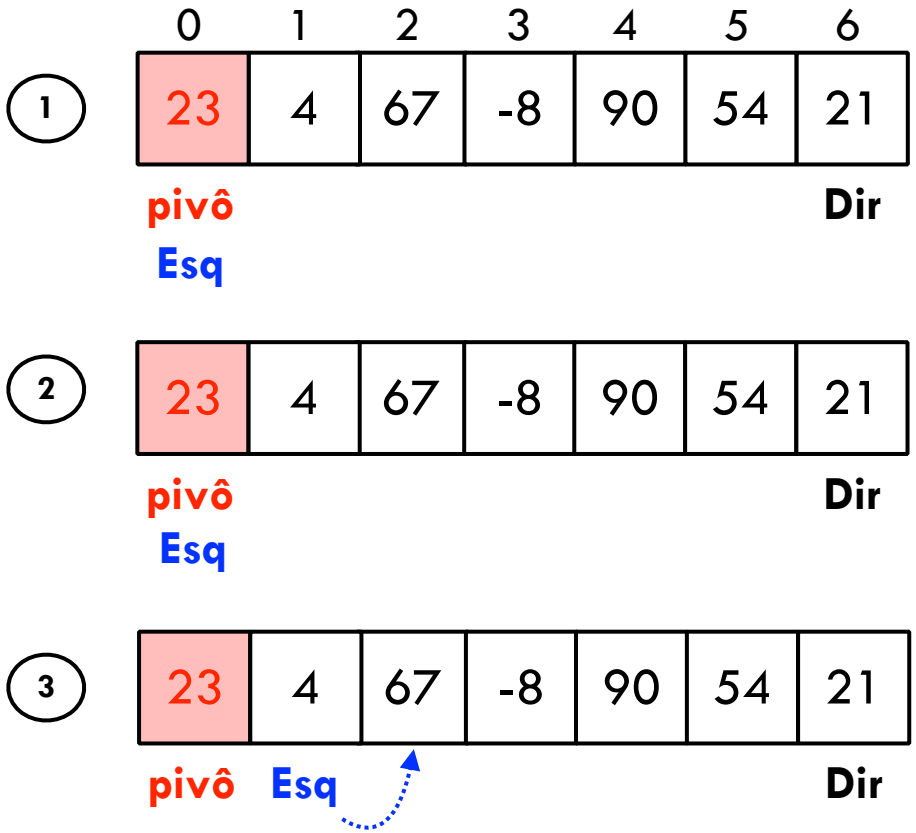


Esq = Inicio = 0
Dir = Fim = 6
pivô = v[Inicio] = 23

V[Esq] <= Pivô: Esq++

Exemplo

Particiona(V, 0, 6)



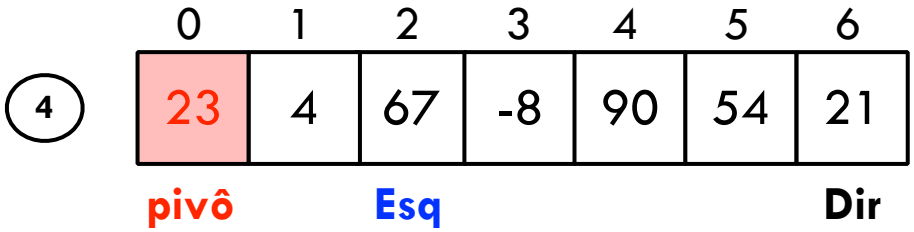
Esq = Inicio = 0
Dir = Fim = 6
pivô = v[Inicio] = 23

V[Esq] <= Pivô: Esq++

V[Esq] <= Pivô: Esq++

Exemplo

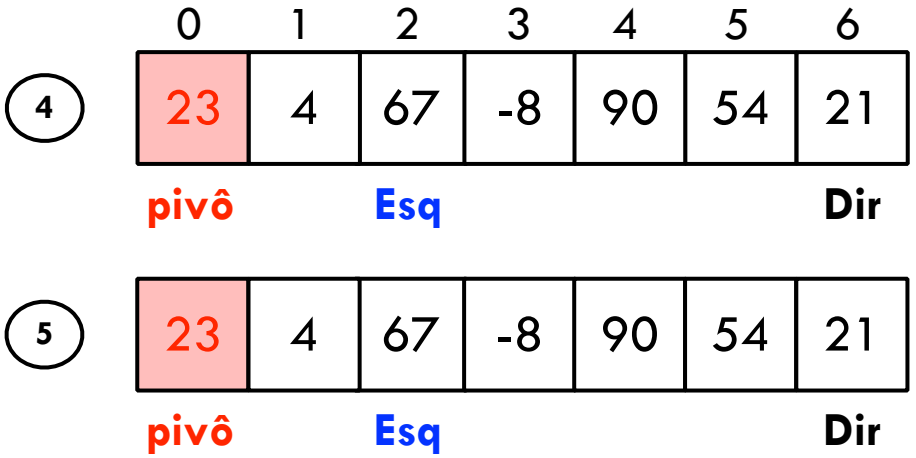
Particiona(V, 0, 6)



V[Esq] > Pivô: Comparar Dir

Exemplo

Particiona(V, 0, 6)

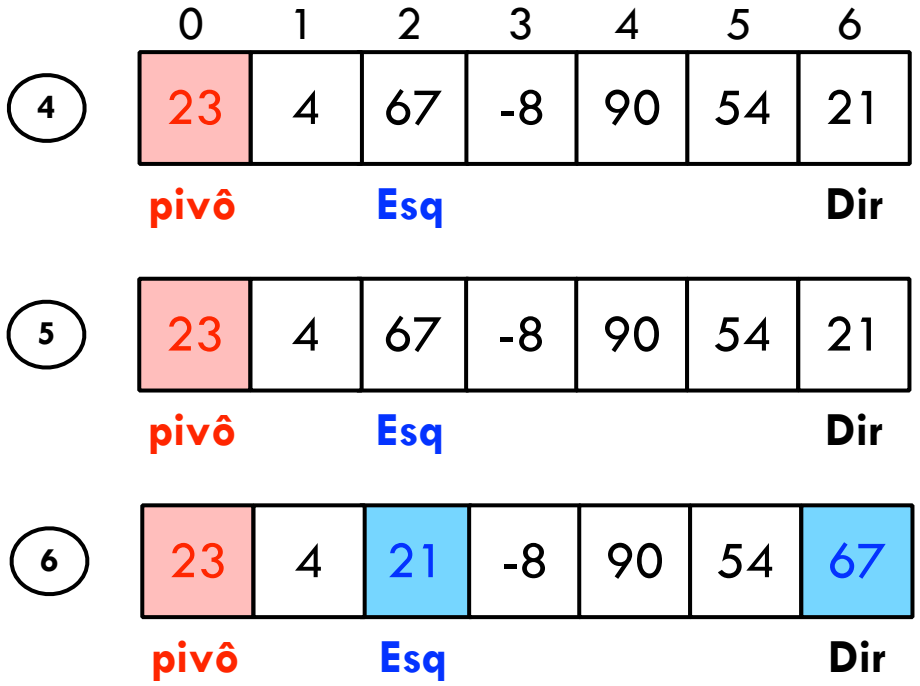


V[Esq] > Pivô: Comparar Dir

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

Exemplo

Particiona(V, 0, 6)



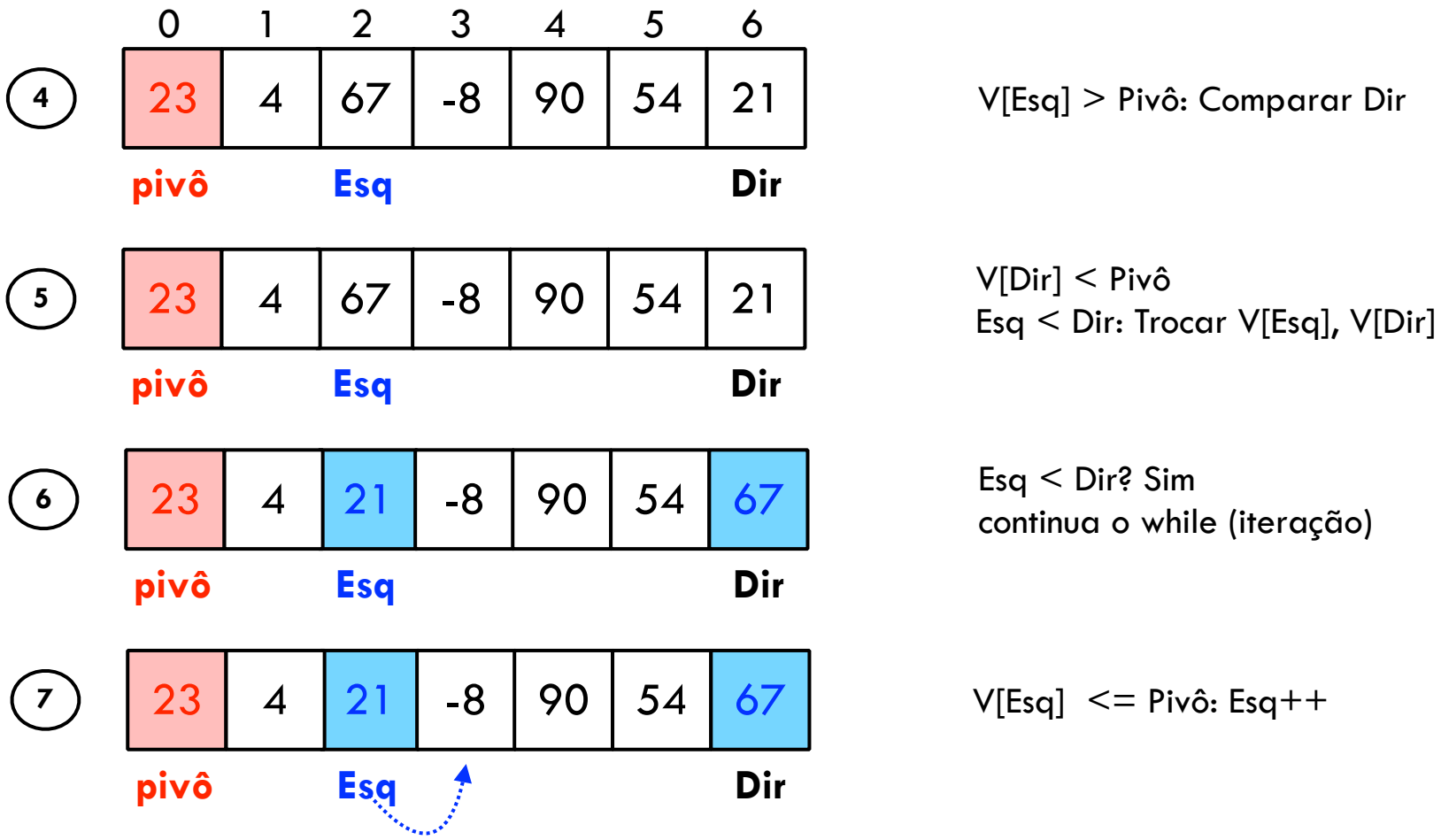
V[Esq] > Pivô: Comparar Dir

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

Esq < Dir? Sim
continua o while (iteração)

Exemplo

Particiona(V, 0, 6)



V[Esq] > Pivô: Comparar Dir

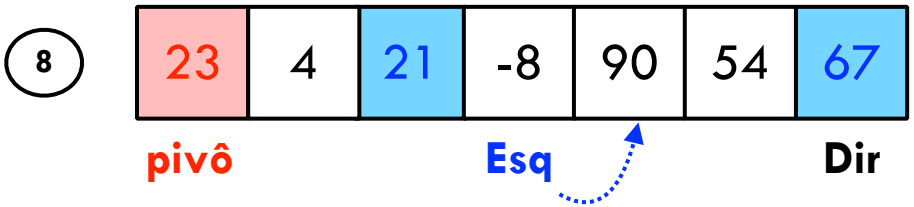
V[Dir] < Pivô
Esq < Dir: Trocar V[Esq], V[Dir]

Esq < Dir? Sim
continua o while (iteração)

V[Esq] <= Pivô: Esq++

Exemplo

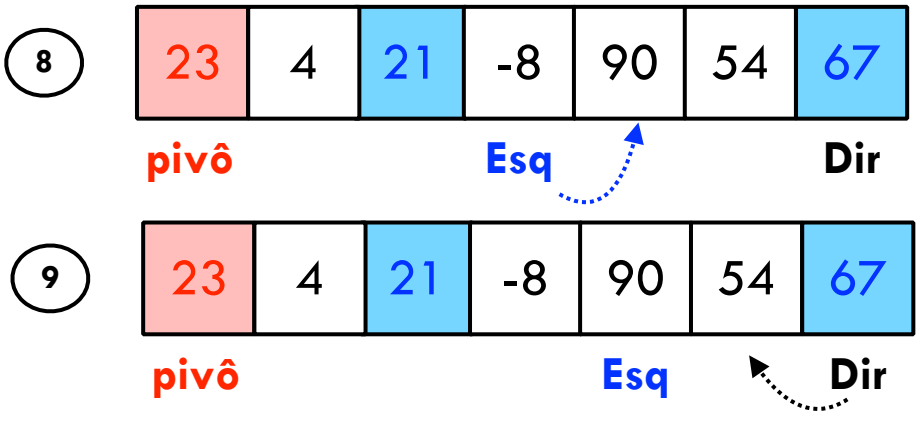
Particiona(V, 0, 6)



V[Esq] > Pivô: Comparar Dir

Exemplo

Particiona(V, 0, 6)

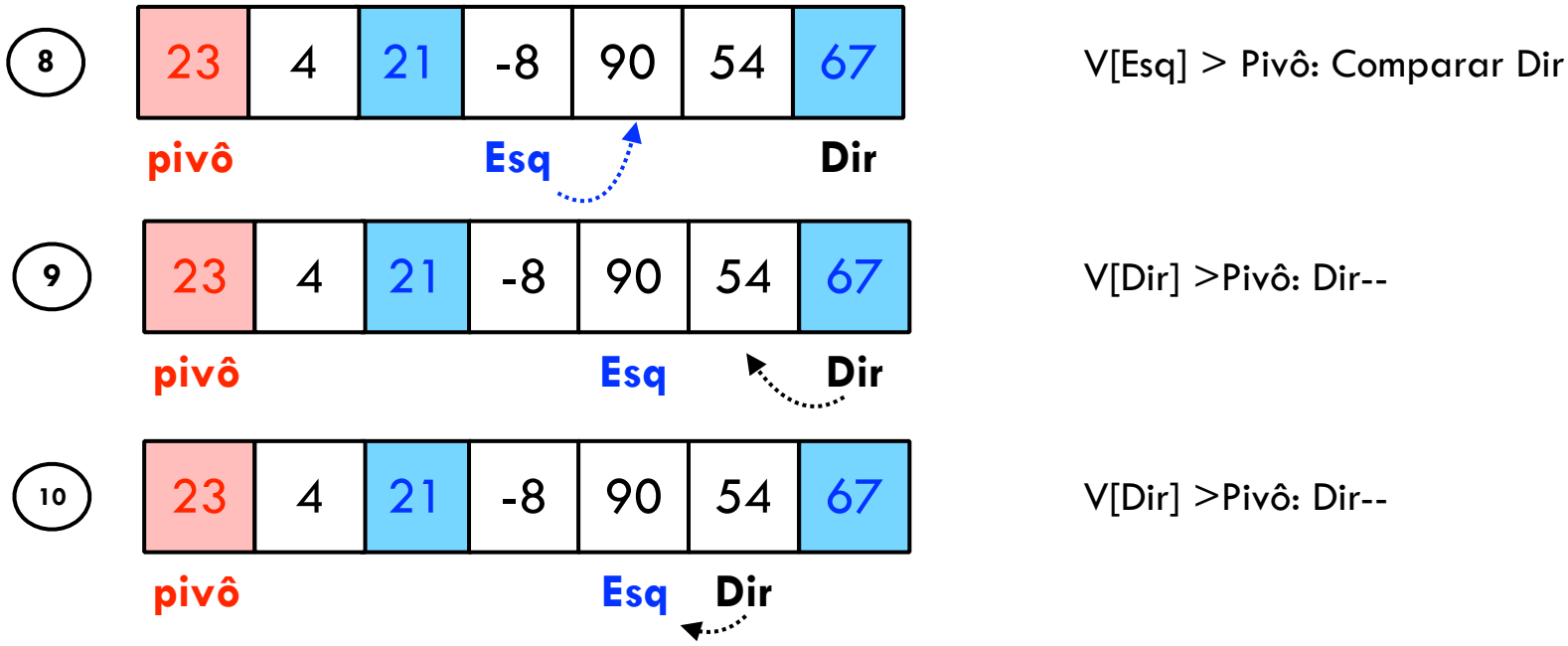


V[Esq] > Pivô: Comparar Dir

V[Dir] >Pivô: Dir--

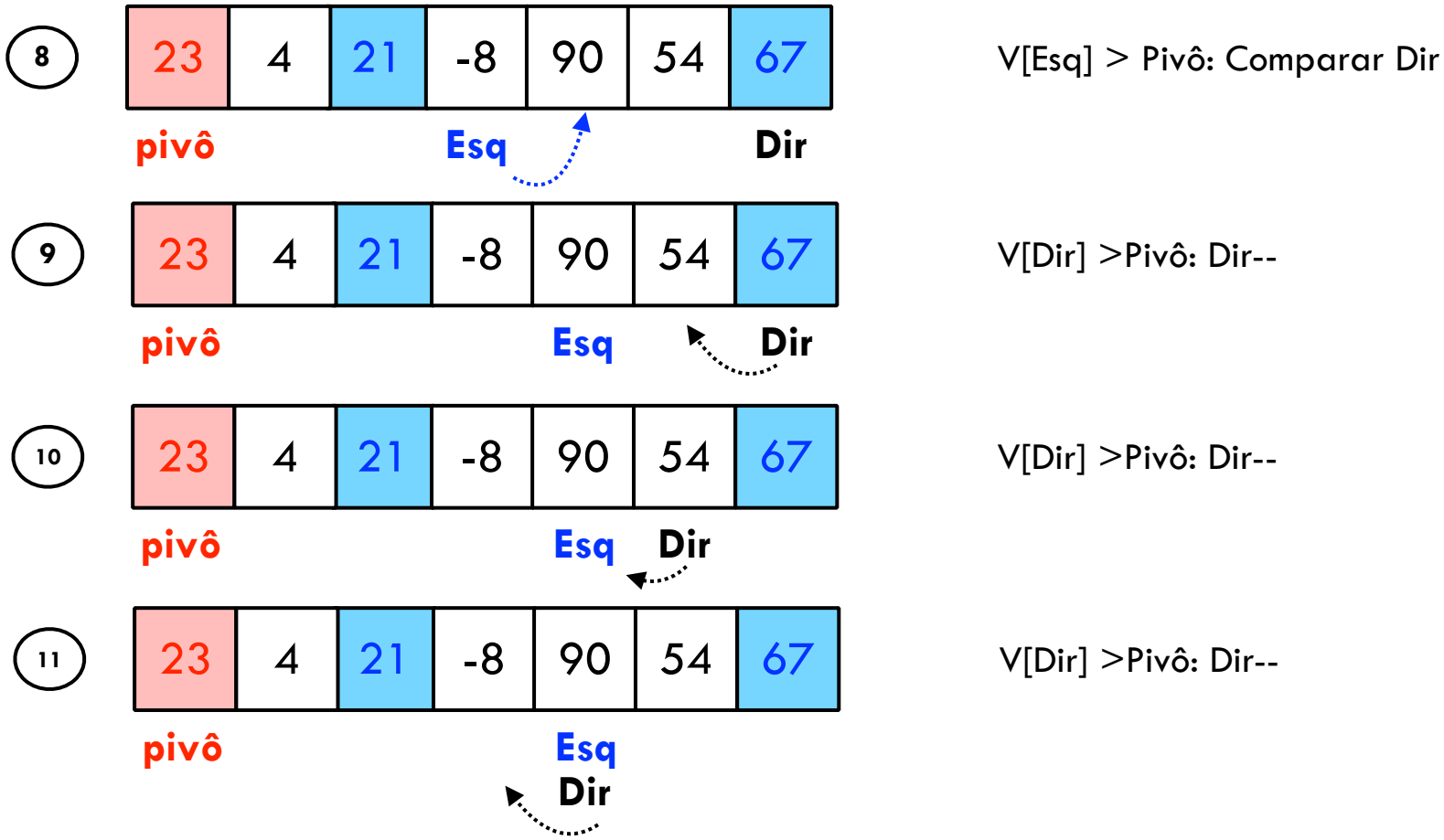
Exemplo

Particiona(V, 0, 6)



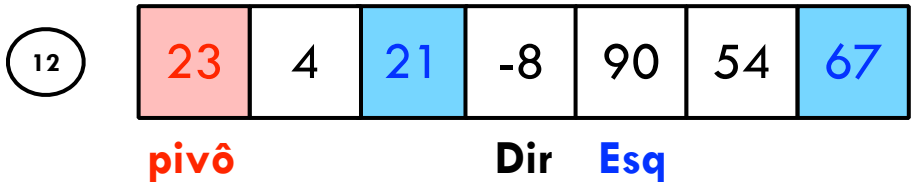
Exemplo

Particiona(V, 0, 6)



Exemplo

Particiona(V, 0, 6)

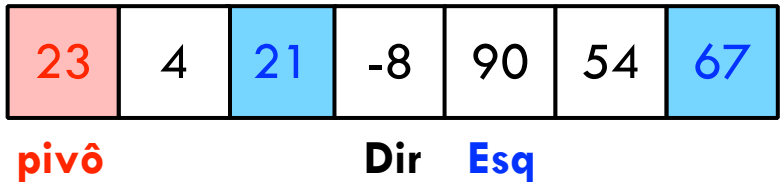


V[Dir] < Pivô
Dir < Esq: terminar while

Exemplo

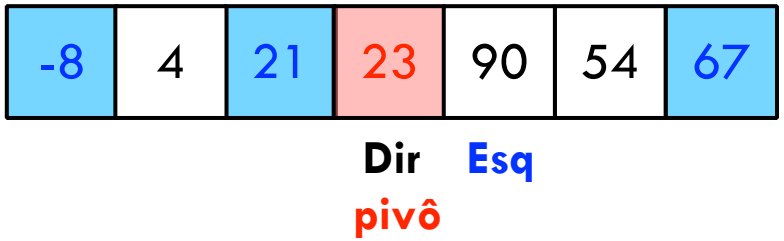
Particiona(V, 0, 6)

12



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

13

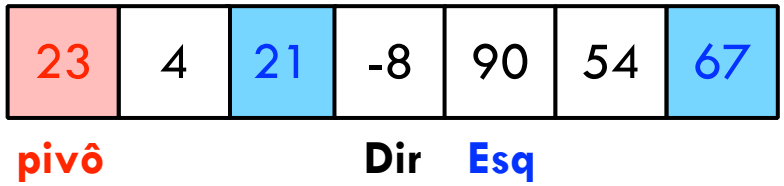


Trocar V[Dir] e V[Inicio]

Exemplo

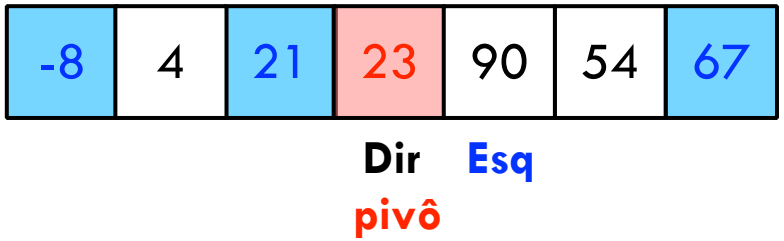
Particiona(V, 0, 6)

12



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

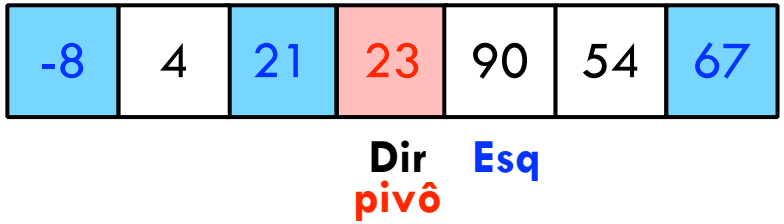
13



Trocar V[Dir] e V[Inicio]

V[dir] é o novo pivô

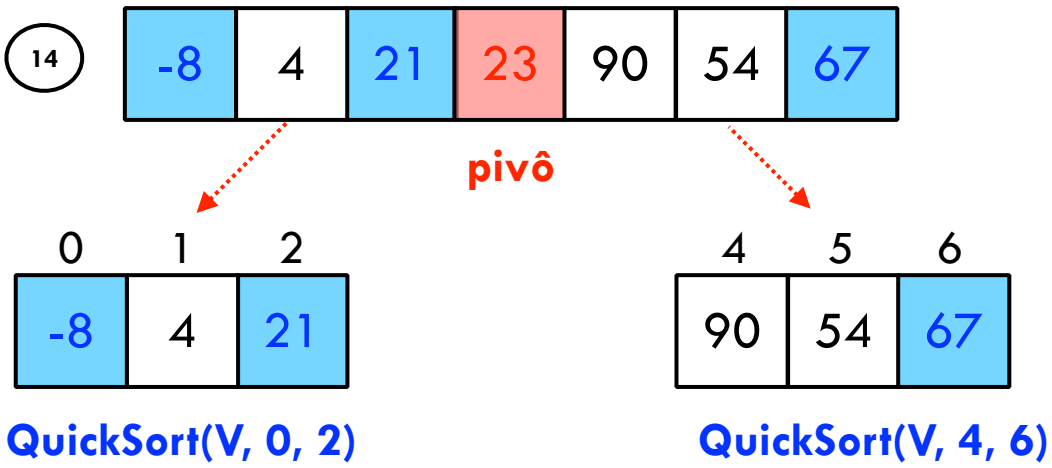
14



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

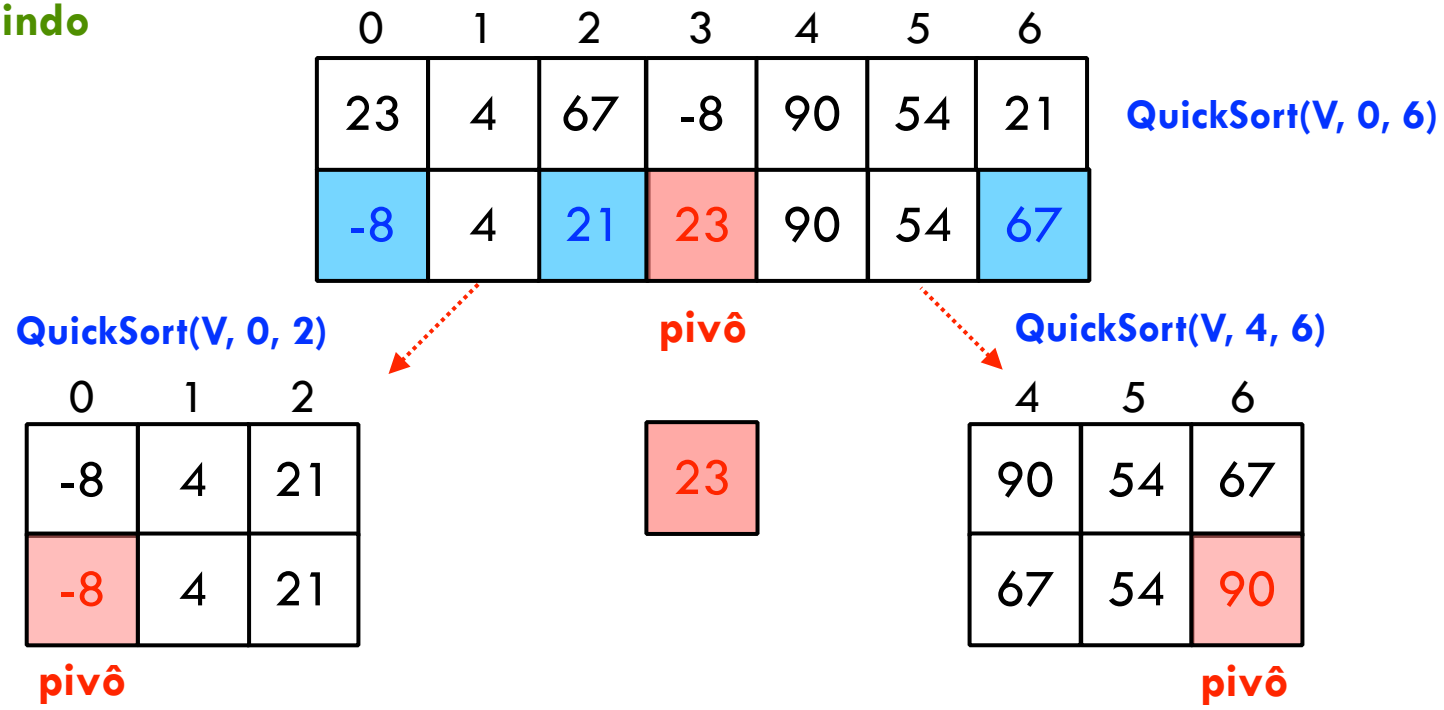
Exemplo

Particiona(V, 0, 6)

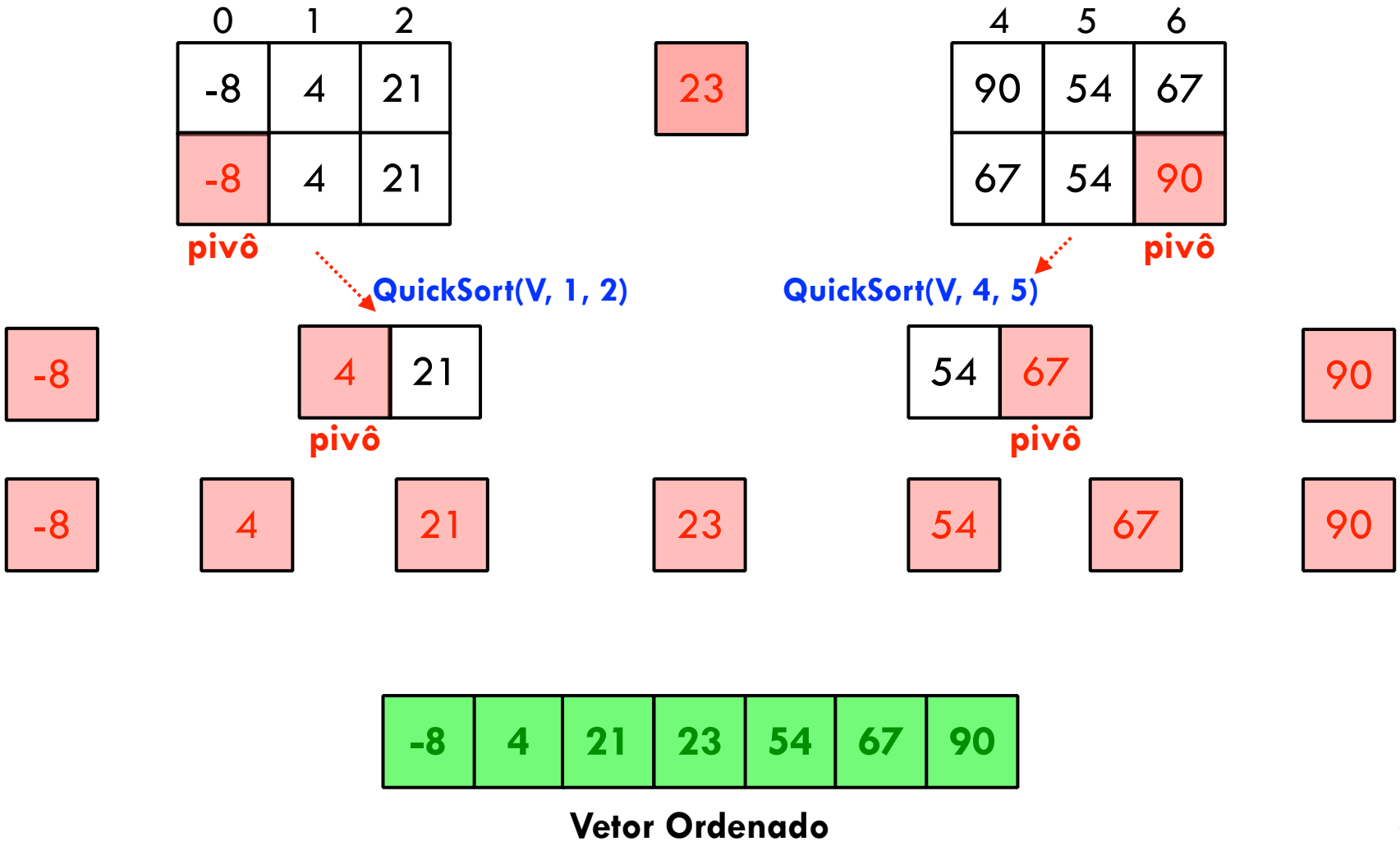


Exemplo

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) Reuna-se com seu grupo e execute o teste de mesa (simulação) do algoritmo para as sequências de números apresentadas

Link planilha grupos/sequências de teste:

https://docs.google.com/spreadsheets/d/1X9IGtcZeAt7j0IIR1W3JPupI2wCtPQgybH0KZm8j-iE/edit?usp=sharing

Exercícios

2) 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

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

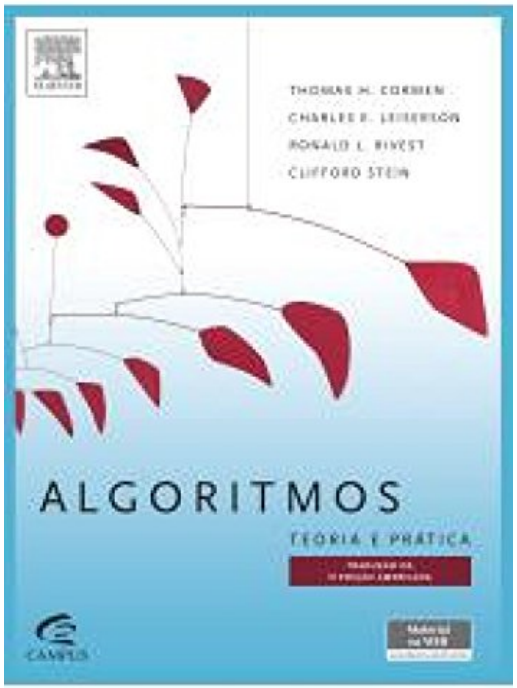
```
/* Parâmetros:
      v: vetor a ser particionado
      inicio: índice do primeiro elemento do vetor
      fim: índice do último elemento do vetor
   Retorno:
      índice do elemento pivô

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

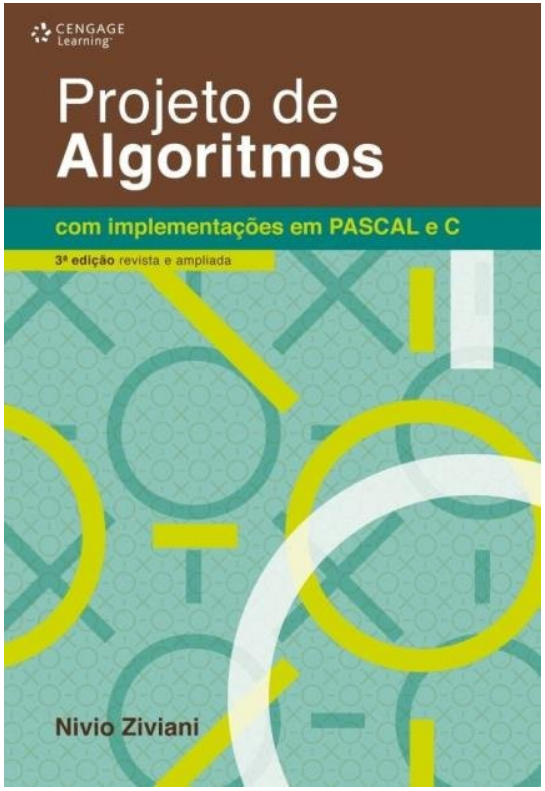
Referências sugeridas



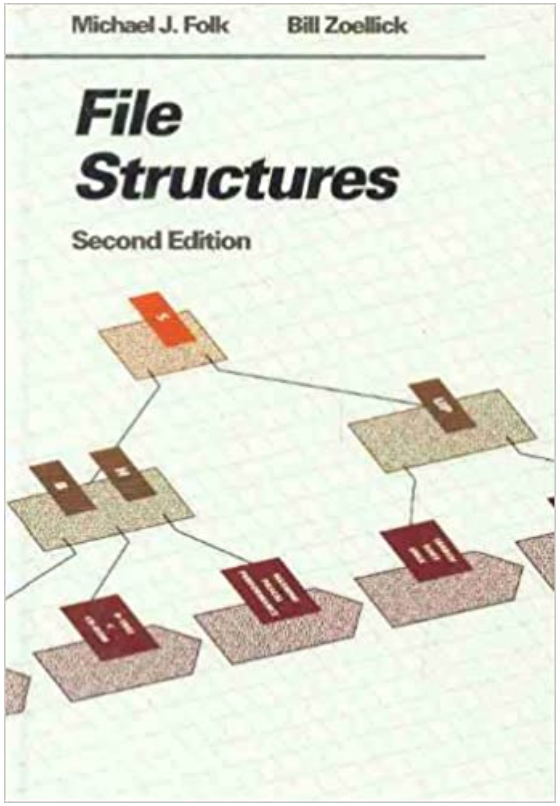
[Cormen et al, 2018]

[Drozdek, 2017]

Referências sugeridas



[Ziviani, 2010]



[Folk & Zoellick, 1992]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br