EDCO4B
ESTRUTURAS DE DADOS 2

Aula 06 - Quick Sort

Prof. Rafael G. Mantovani
Prof. Luiz Fernando Carvalho

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Apucarana - PR, Brasil

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia de Computação

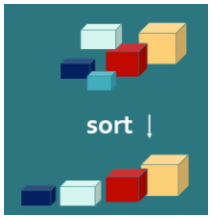
Roteiro

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

Roteiro

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

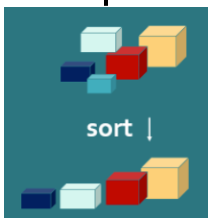
Introdução



Algoritmos de Ordenação

Introdução

Métodos Simples ⟷ Métodos Eficientes



Algoritmos de Ordenação

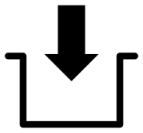
Introdução

Métodos Simples

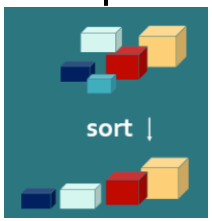
Métodos Eficientes

BubbleSort

InsertionSort



SelectionSort



Algoritmos de Ordenação

HeapSort

MergeSort

QuickSort

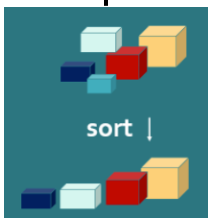
Introdução

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

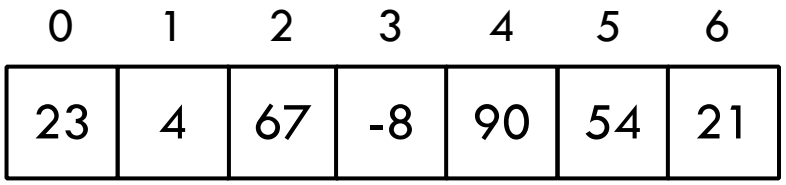
Quick Sort

- **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

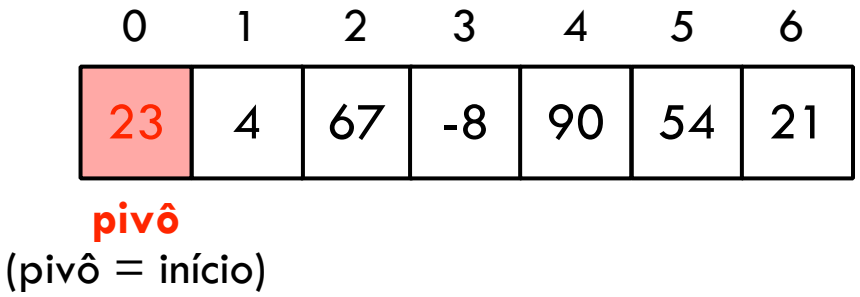
Quick Sort



QuickSort(V, 0, 6)

Quick Sort

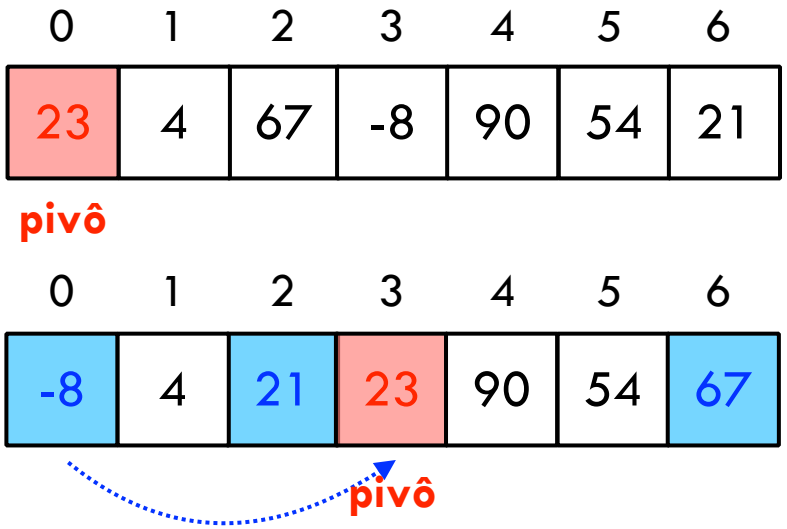
- Encontrar um pivô:



QuickSort(V, 0, 6)

Quick Sort

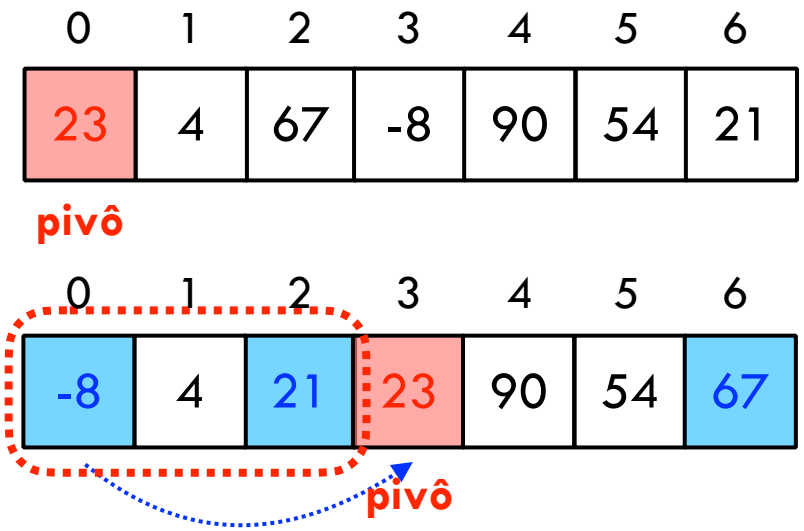
- Posicionar o pivô



QuickSort(V, 0, 6)

Quick Sort

- Posicionar o pivô

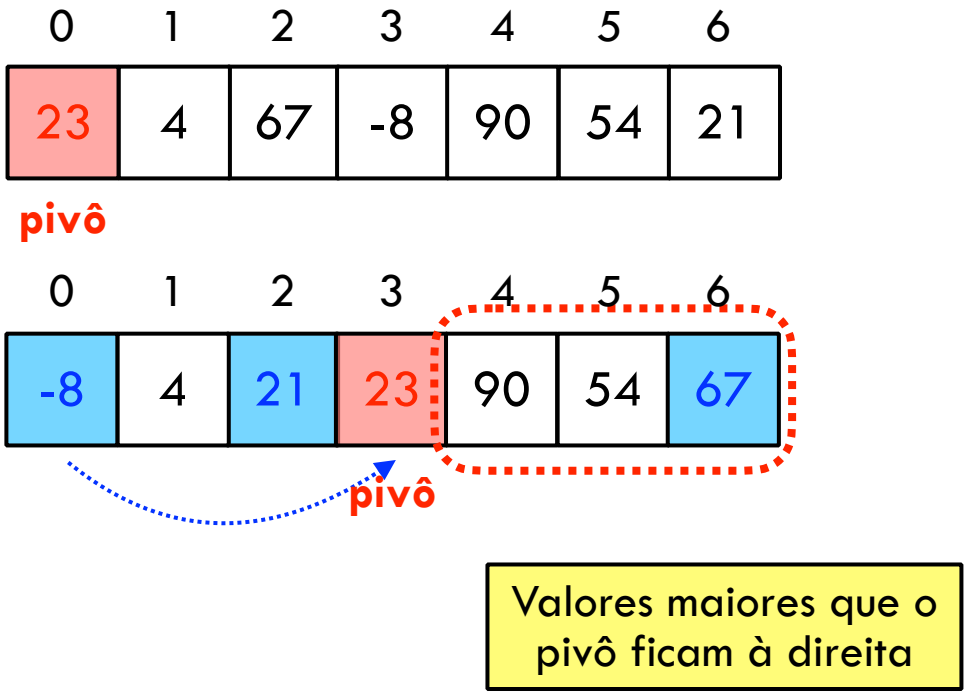


QuickSort(V, 0, 6)

Valores menores que o pivô ficam à esquerda

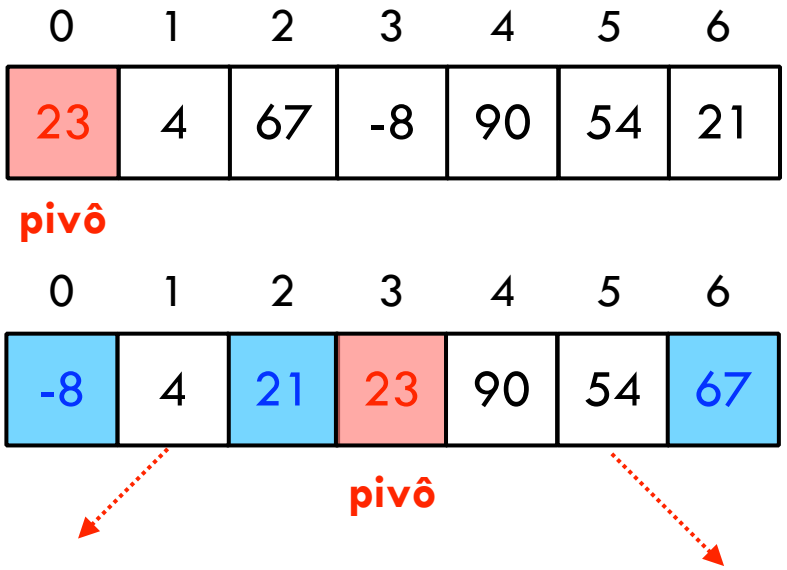
Quick Sort

- Posicionar o pivô



Quick Sort

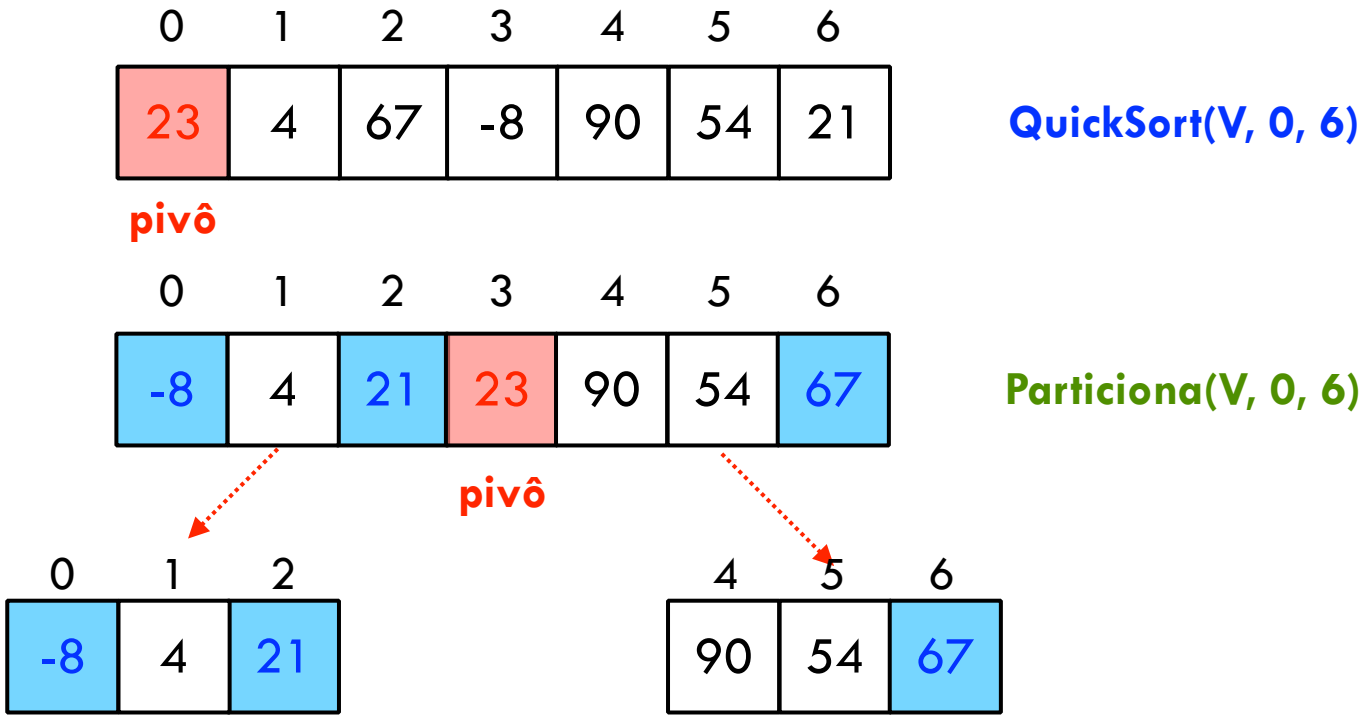
- Chamar recursivamente, desconsiderando o pivô:



QuickSort(V, 0, 6)

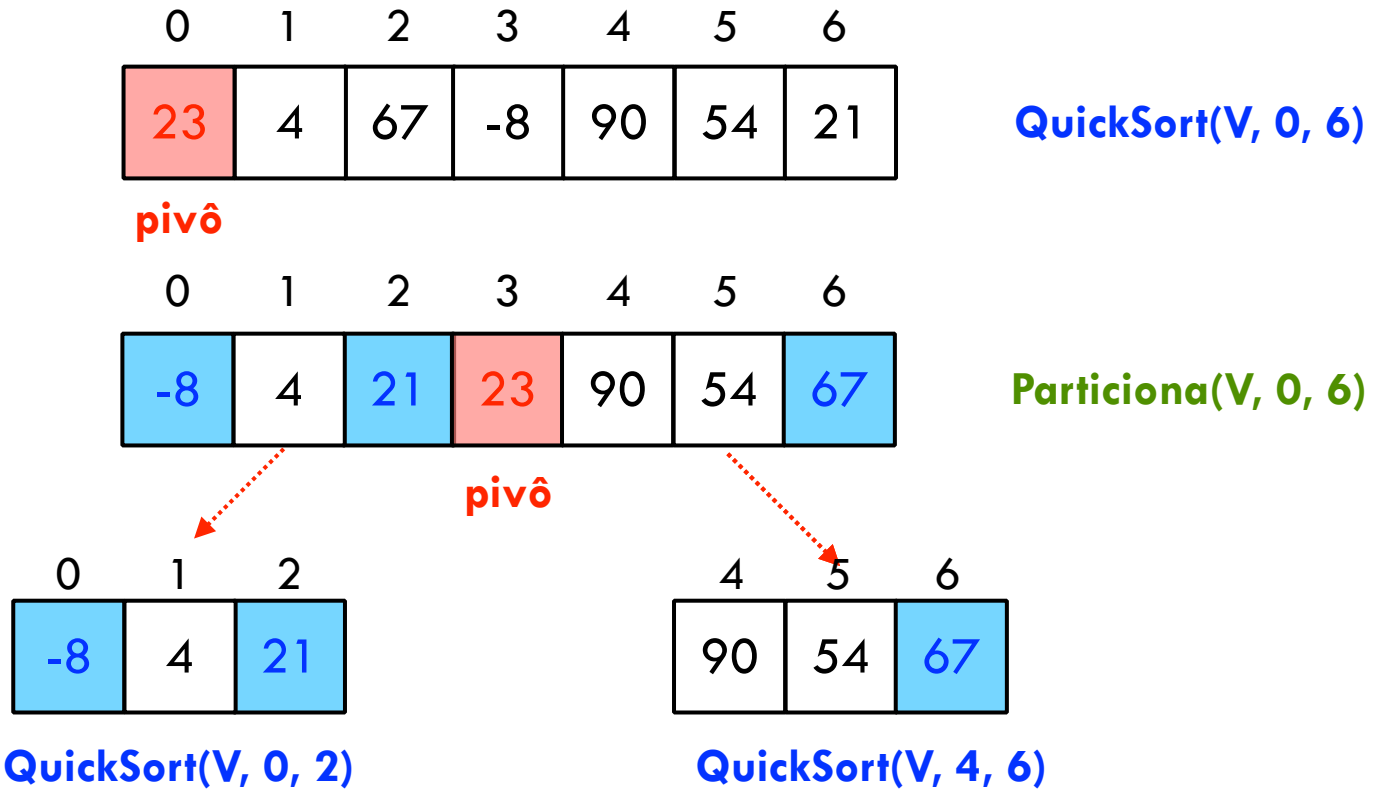
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



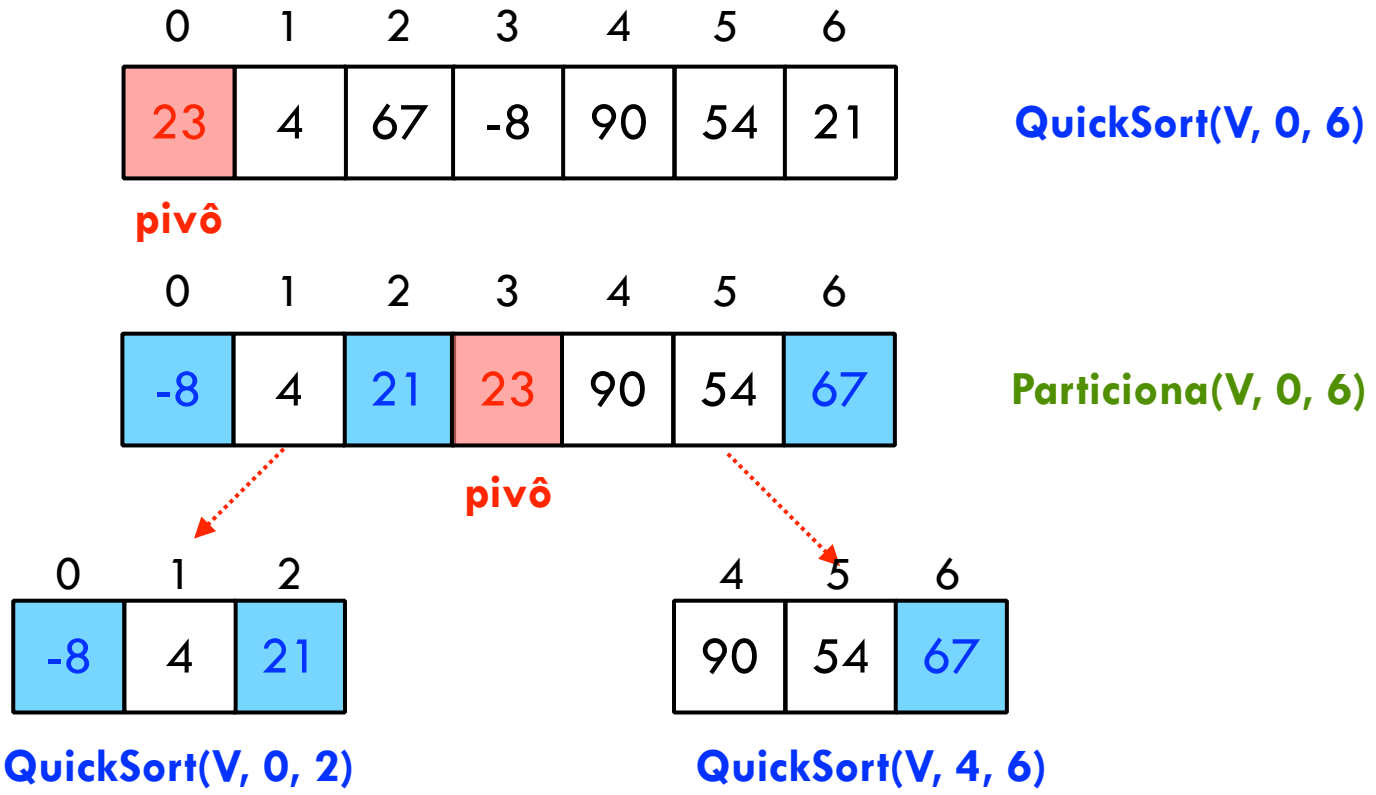
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



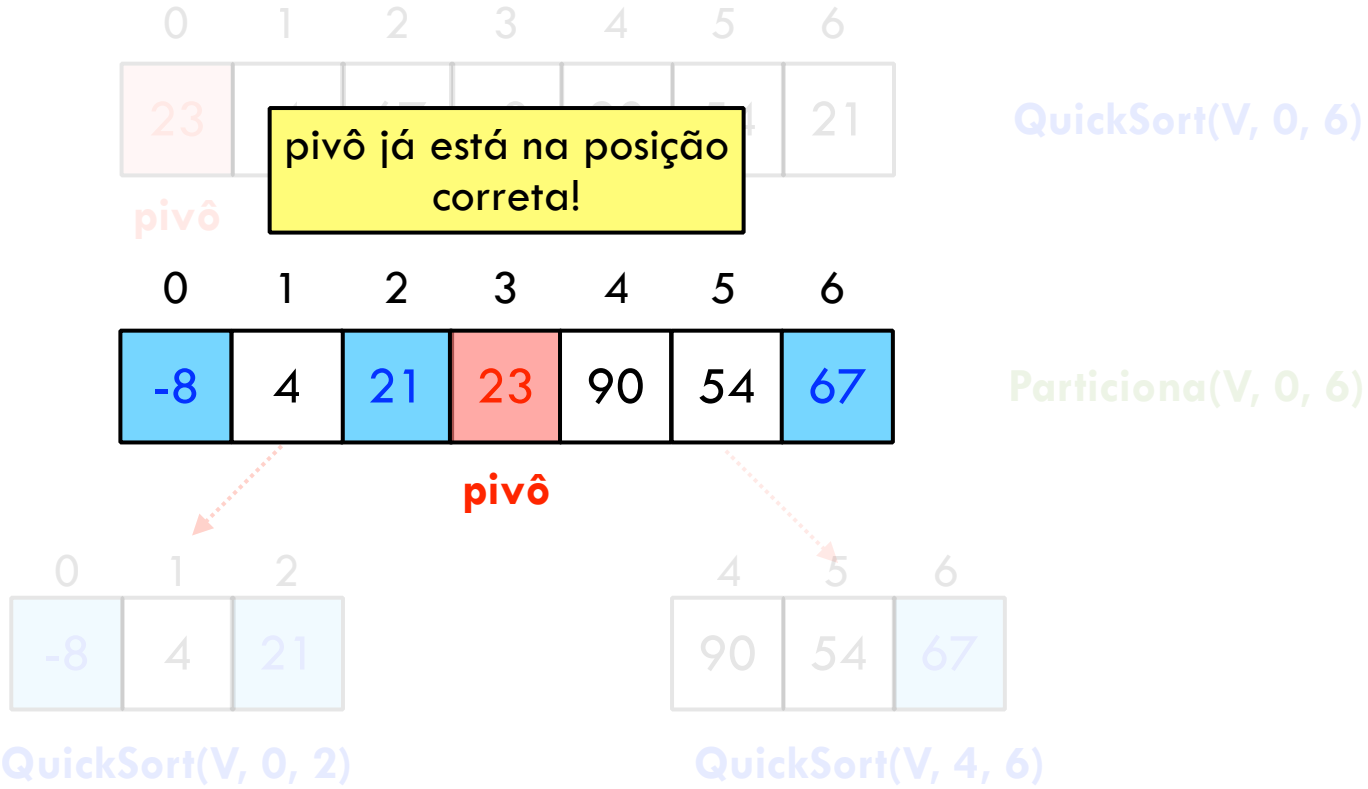
Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



Quick Sort

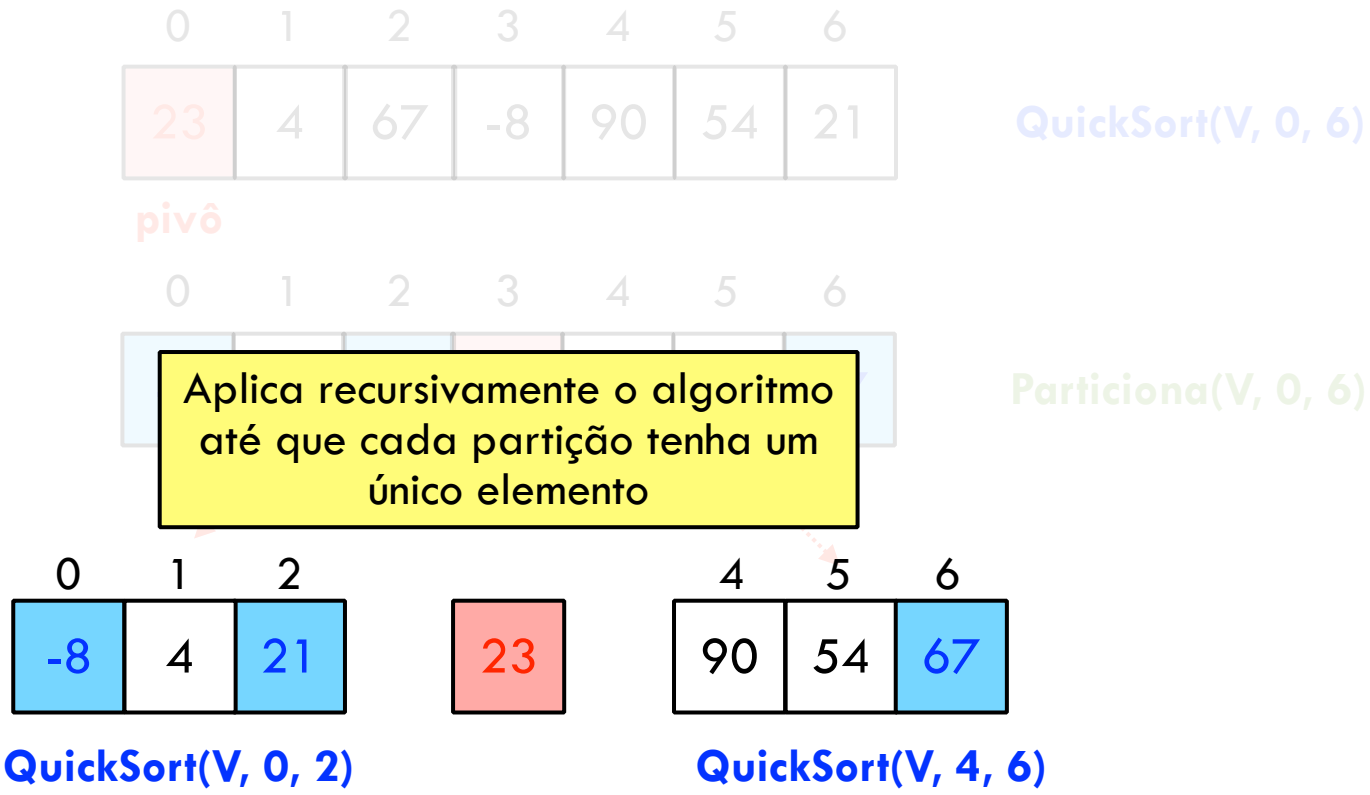
- Chamar recursivamente, desconsiderando o pivô:



...

Quick Sort

- Chamar recursivamente, desconsiderando o pivô:



Quick Sort

- **Desempenho**

* **melhor caso:** O(N log N), quando o pivô selecionado é a mediana dos elementos

* **pior caso:** $O(N^2)$, quando o array é dividido em duas partes com [N-1] e 1 elementos

* **caso médio:** O(N log 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

Pseudocódigo: Quick Sort

QuickSort (**vetor**, **Inicio**, **Fim**):

1. **Se** (**Inicio** < **Fim**) então : *//critério para controle de recursão*

Pseudocódigo: Quick Sort

QuickSort (**vetor**, **Inicio**, **Fim**):

```
1.  Se (Inicio < Fim) então :                    //critério para controle de recursão
        // define um elemento pivô, para separar os dados
2.      Pivo = Particiona(vetor, Inicio, Fim)
```

Pseudocódigo: Quick Sort

QuickSort (**vetor**, **Inicio**, **Fim**):

```
1.  Se (Inicio < Fim) então :                    //critério para controle de recursão

    // define um elemento pivô, para separar os dados
2.  Pivo = Particiona(vetor, Inicio, Fim)

    // chamada recursiva para os subproblemas
3.  QuickSort(vetor, Inicio, Pivo - 1)           //subproblema da esquerda
4.  QuickSort(vetor, Pivo + 1, Fim)              //subproblema da direita
```

Pseudocódigo: Quick Sort

```
QuickSort (vetor, Inicio, Fim):

1.  Se (Inicio < Fim) então :                    //critério para controle de recursão

        // define um elemento pivô, para separar os dados
2.      Pivo = Particiona(vetor, Inicio, Fim)

        // chamada recursiva para os subproblemas
3.      QuickSort(vetor, Inicio, Pivo - 1)       //subproblema da esquerda
4.      QuickSort(vetor, Pivo + 1, Fim)          //subproblema da direita

// Ao final da última chamada da função principal, o vetor "vetor"
está ordenado
```

Pseudocódigo: Quick Sort

QuickSort (**vetor**, **Inicio**, **Fim**):

```
1.  Se (Inicio < Fim) então :
2.      Pivo = Particiona(vetor, Inicio, Fim)
3.      QuickSort(vetor, Inicio, Pivo - 1)      //subproblema da esquerda
4.      QuickSort(vetor, Pivo + 1, Fim)         //subproblema da direita
```

Pseudocódigo: Quick Sort

```
QuickSort (vetor, Inicio, Fim):

1.  Se (Inicio < Fim) então :
2.      Pivo = Particiona(vetor, Inicio, Fim)
3.      QuickSort(vetor, Inicio, Pivo - 1)      //subproblema da esquerda
4.      QuickSort(vetor, Pivo + 1, Fim)         //subproblema da direita
```

Função Auxiliar: Particiona

Particiona (**vetor**, **Inicio**, **Fim**):

```
// iniciar variáveis locais
1. Esquerda = Inicio
2. Direita = Fim
3. Pivo = vetor [Inicio]
```

Função Auxiliar: Particiona

Particiona (**vetor**, **Inicio**, **Fim**):

// iniciar variáveis locais

1. **Esquerda** = **Inicio**
2. **Direita** = **Fim**
3. **Pivo** = **vetor** [**Inicio**]

// procurar elementos para trocar de posição

4. **Enquanto** (**Esquerda** < **Direita**), **faça**:

Função Auxiliar: Particiona

```
Particiona (vetor, Inicio, Fim):
    // iniciar variáveis locais
1.  Esquerda = Inicio
2.  Direita = Fim
3.  Pivo = vetor [Inicio]
    // procurar elementos para trocar de posição
4.  Enquanto (Esquerda < Direita), faça:
        // procurar do início ao fim do vetor elementos que sejam maiores que o pivô
5.      Enquanto ((vetor[ Esquerda ] <= Pivo) E (Esquerda <= Fim)), faça:
```

Função Auxiliar: Particiona

```
Particiona (vetor, Inicio, Fim):
    // iniciar variáveis locais
1.  Esquerda = Inicio
2.  Direita = Fim
3.  Pivo = vetor [Inicio]
    // procurar elementos para trocar de posição
4.  Enquanto (Esquerda < Direita), faça:
        // procurar do início ao fim do vetor elementos que sejam maiores que o pivô
5.      Enquanto ((vetor[ Esquerda ] <= Pivo) E (Esquerda <= Fim)), faça:
6.          Esquerda = Esquerda + 1
```

Função Auxiliar: Particiona

Particiona (vetor, Inicio, Fim):

Função Auxiliar: Particiona

```
Particiona (vetor, Inicio, Fim):

    // procurar do fim ao início do vetor elementos que sejam menores que o pivô
7.  Enquanto ((vetor[ Direita ] > Pivo) E (Direita > Inicio)), faça:
8.      Direita = Direita - 1
```

Função Auxiliar: Particiona

```
Particiona (vetor, Inicio, Fim):
    // procurar do fim ao início do vetor elementos que sejam menores que o pivô
7.  Enquanto ((vetor[ Direita ] > Pivo) E (Direita > Inicio)), faça:
8.      Direita = Direita - 1
    // se foram encontrados dois elementos que podem ser trocados, efetuar a troca
9.  Se (Esquerda <  Direita), faça:
```

Função Auxiliar: Particiona

Particiona (**vetor**, **Inicio**, **Fim**):

```
    // procurar do fim ao início do vetor elementos que sejam menores que o pivô
7.  Enquanto ((vetor[ Direita ] > Pivo) E (Direita > Inicio)), faça:
8.      Direita = Direita - 1
    // se foram encontrados dois elementos que podem ser trocados, efetuar a troca
9.  Se (Esquerda <  Direita), faça:
10.     Trocar de posição os elementos vetor [Esquerda] e vetor [Direita]
```

Função Auxiliar: Particiona

```
Particiona (vetor, Inicio, Fim):
    // procurar do fim ao início do vetor elementos que sejam menores que o pivô
7.  Enquanto ((vetor[ Direita ] > Pivo) E (Direita > Inicio)), faça:
8.      Direita = Direita - 1
    // se foram encontrados dois elementos que podem ser trocados, efetuar a troca
9.  Se (Esquerda <  Direita), faça:
10.     Trocar de posição os elementos vetor [Esquerda] e vetor [Direita]
// Direita armazena a posição correta para o pivô (vetor [inicio])
```

Função Auxiliar: Particiona

Particiona (**vetor**, **Inicio**, **Fim**):

```
    // procurar do fim ao início do vetor elementos que sejam menores que o pivô
7.  Enquanto ((vetor[ Direita ] > Pivo) E (Direita > Inicio)), faça:
8.      Direita = Direita - 1
    // se foram encontrados dois elementos que podem ser trocados, efetuar a
    troca
9.  Se (Esquerda <  Direita), faça:
10.     Trocar de posição os elementos vetor [Esquerda] e vetor [Direita]
    // Direita armazena a posição correta para o pivô (vetor [inicio])
11. Trocar de posição os elementos vetor [Direita] e vetor [Inicio]
```

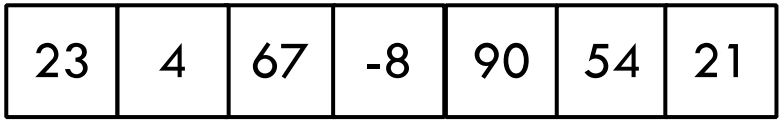
Função Auxiliar: Particiona

```
Particiona (vetor, Inicio, Fim):
    // procurar do fim ao início do vetor elementos que sejam menores que o pivô
7.  Enquanto ((vetor[ Direita ] > Pivo) E (Direita > Inicio)), faça:
8.      Direita = Direita - 1
    // se foram encontrados dois elementos que podem ser trocados, efetuar a troca
9.  Se (Esquerda < Direita), faça:
10.     Trocar de posição os elementos vetor [Esquerda] e vetor [Direita]
    // Direita armazena a posição correta para o pivô (vetor [inicio])
11. Trocar de posição os elementos vetor [Direita] e vetor [Inicio]
12. Retorna (Direita)                    // retorna o índice da posição correta do pivô
```

Roteiro

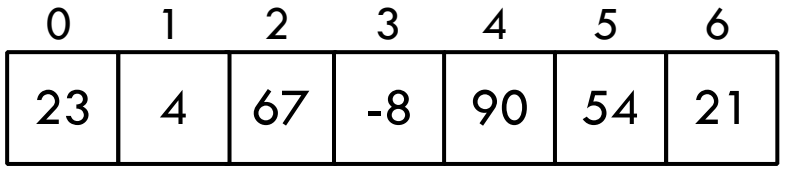
1. Introdução
2. Quick Sort
3. Exemplo
4. Exercícios
5. Referências

Exemplo



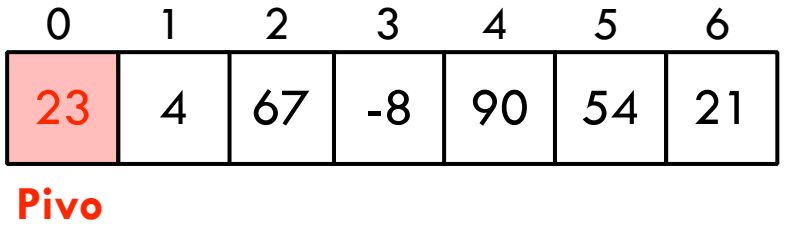
vetor não ordenado

Exemplo



QuickSort(V, 0, 6)

Exemplo

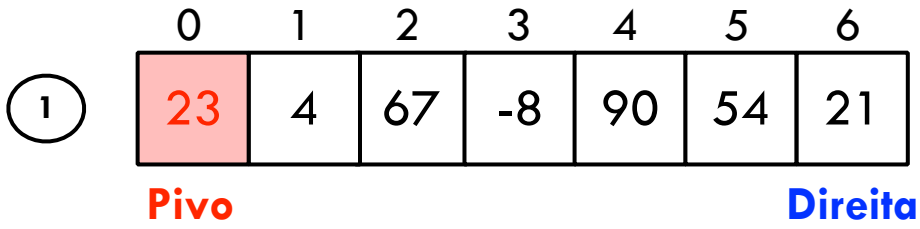


QuickSort(V, 0, 6)

Particiona(V, 0, 6)

Exemplo

Particiona(V, 0, 6)



Esquerda = **Inicio** = 0
Direita = **Fim** = 6
Pivo = **vetor** [**Inicio**] = 23

Exemplo

Particiona(V, 0, 6)

Esquerda = Inicio = 0
Direita = Fim = 6
Pivo = vetor [Inicio] = 23

vetor [Esquerda] <= Pivo: TRUE
Esquerda++

Exemplo

Particiona(V, 0, 6)

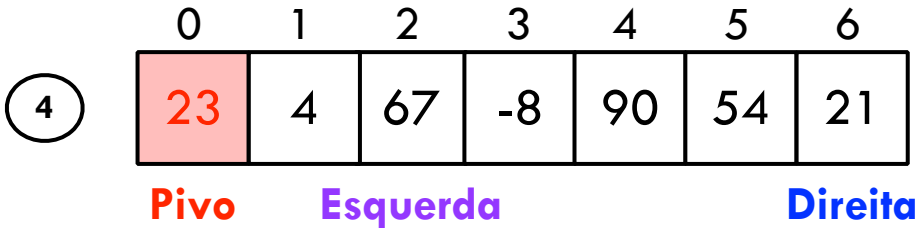
Esquerda = Inicio = 0
Direita = Fim = 6
Pivo = vetor [Inicio] = 23

vetor [Esquerda] <= Pivo: TRUE
Esquerda++

vetor [Esquerda] <= Pivo: TRUE
Esquerda++

Exemplo

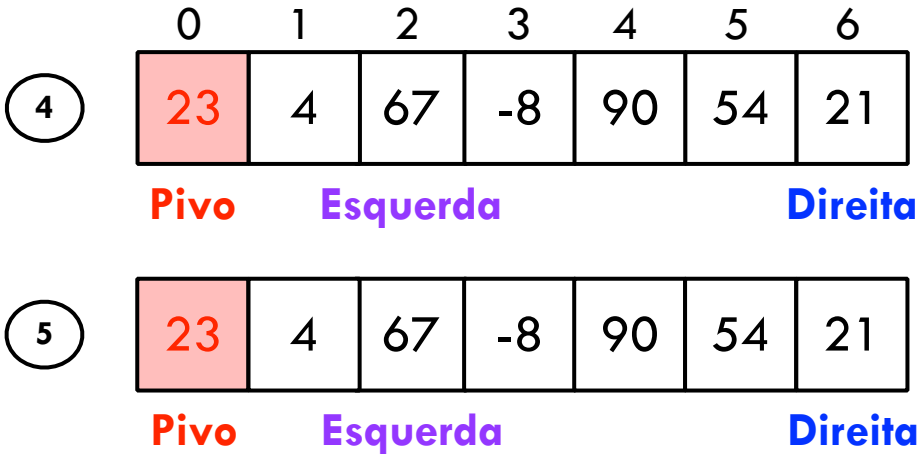
Particiona(V, 0, 6)



vetor [Esquerda] <= **Pivo**: **FALSE**
// comparar Direita

Exemplo

Particiona(V, 0, 6)

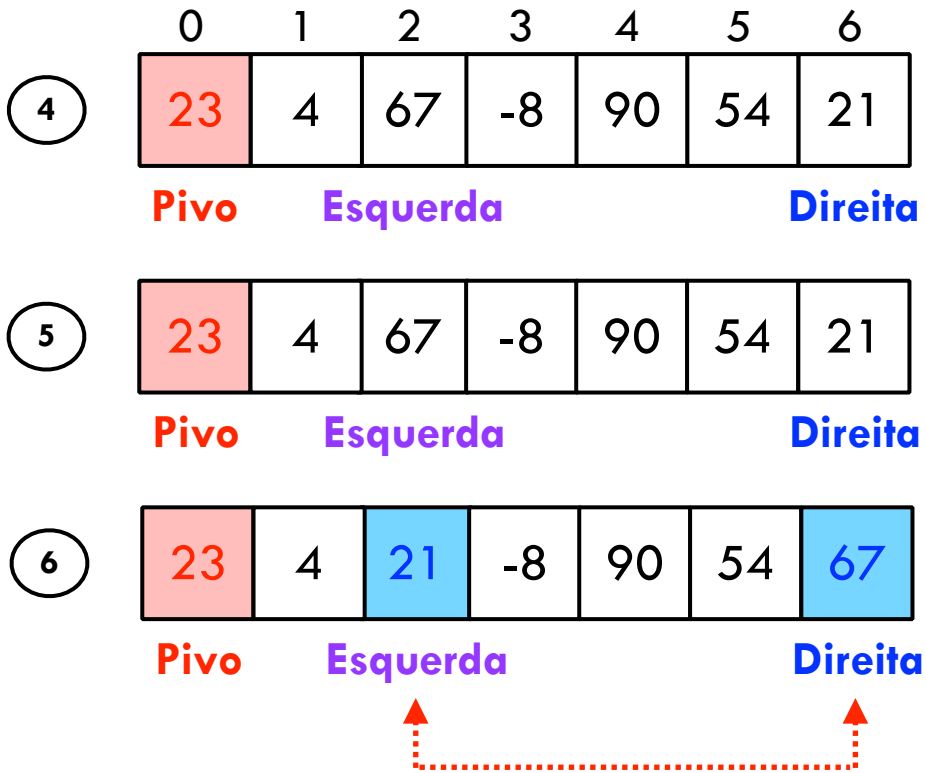


vetor [Esquerda] <= Pivo: FALSE
// comparar Direita

vetor [Direita] > Pivo: FALSE

Exemplo

Particiona(V, 0, 6)



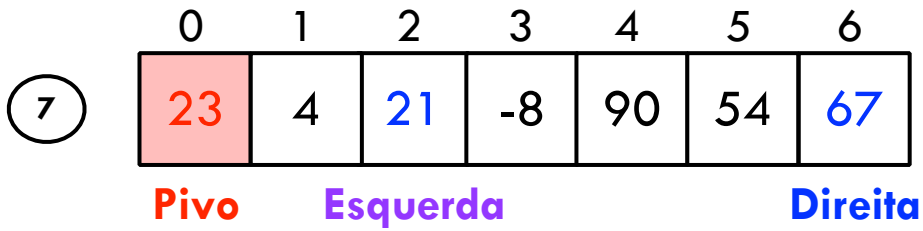
vetor [Esquerda] <= Pivo: FALSE
// comparar Direita

vetor [Direita] > Pivo: FALSE

Esquerda < Direita? TRUE
Trocar:
vetor[Direita] e **vetor[Esquerda]**

Exemplo

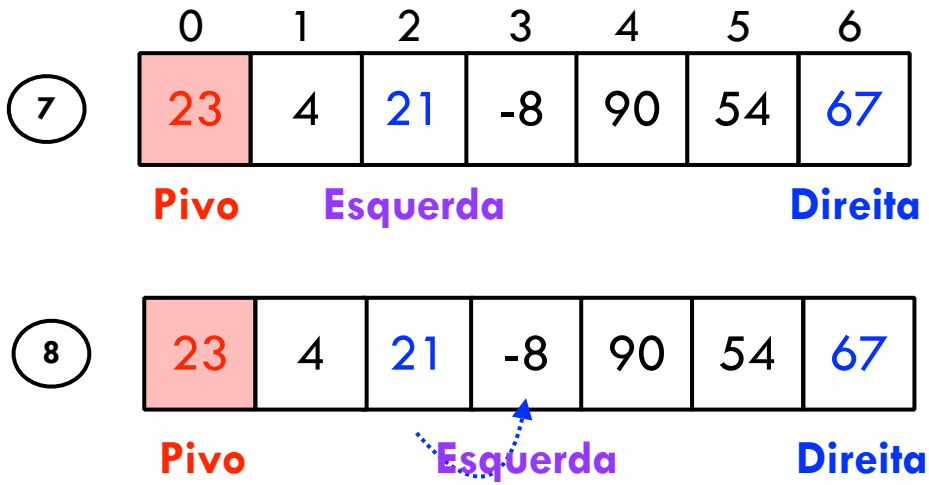
Particiona(V, 0, 6)



vetor [Esquerda] <= **Pivo**: **TRUE**
Esquerda++

Exemplo

Particiona(V, 0, 6)



vetor [Esquerda] <= Pivo: TRUE
Esquerda++

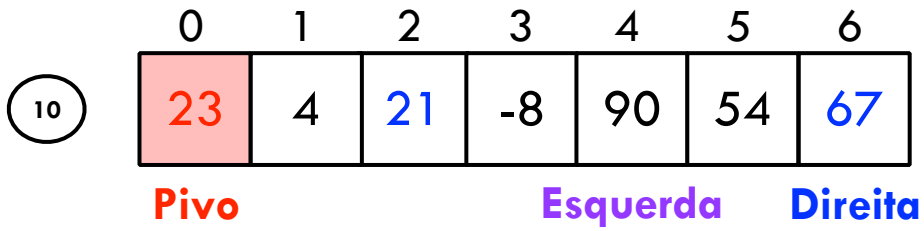
vetor [Esquerda] <= Pivo: TRUE
Esquerda++

Exemplo

Particiona(V, 0, 6)

Exemplo

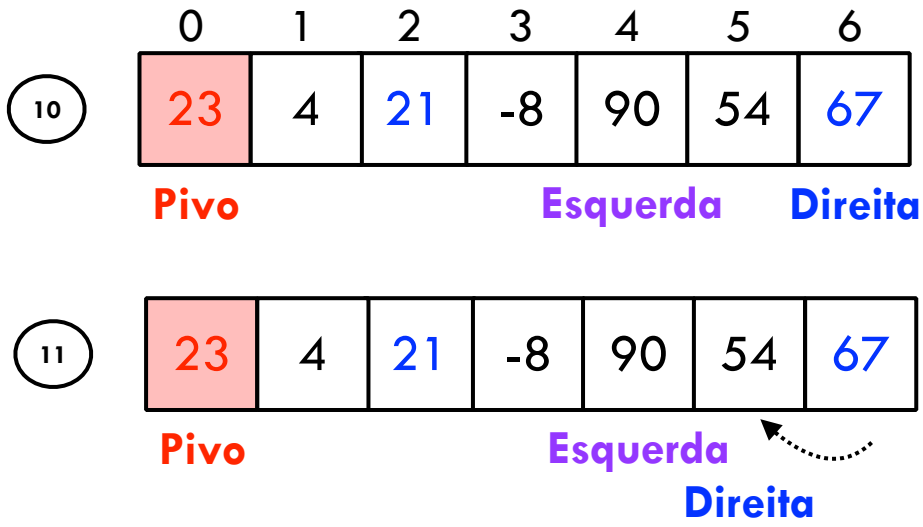
Particiona(V, 0, 6)



vetor [Esquerda] <= **Pivo**: **FALSE**
// comparar Direita

Exemplo

Particiona(V, 0, 6)

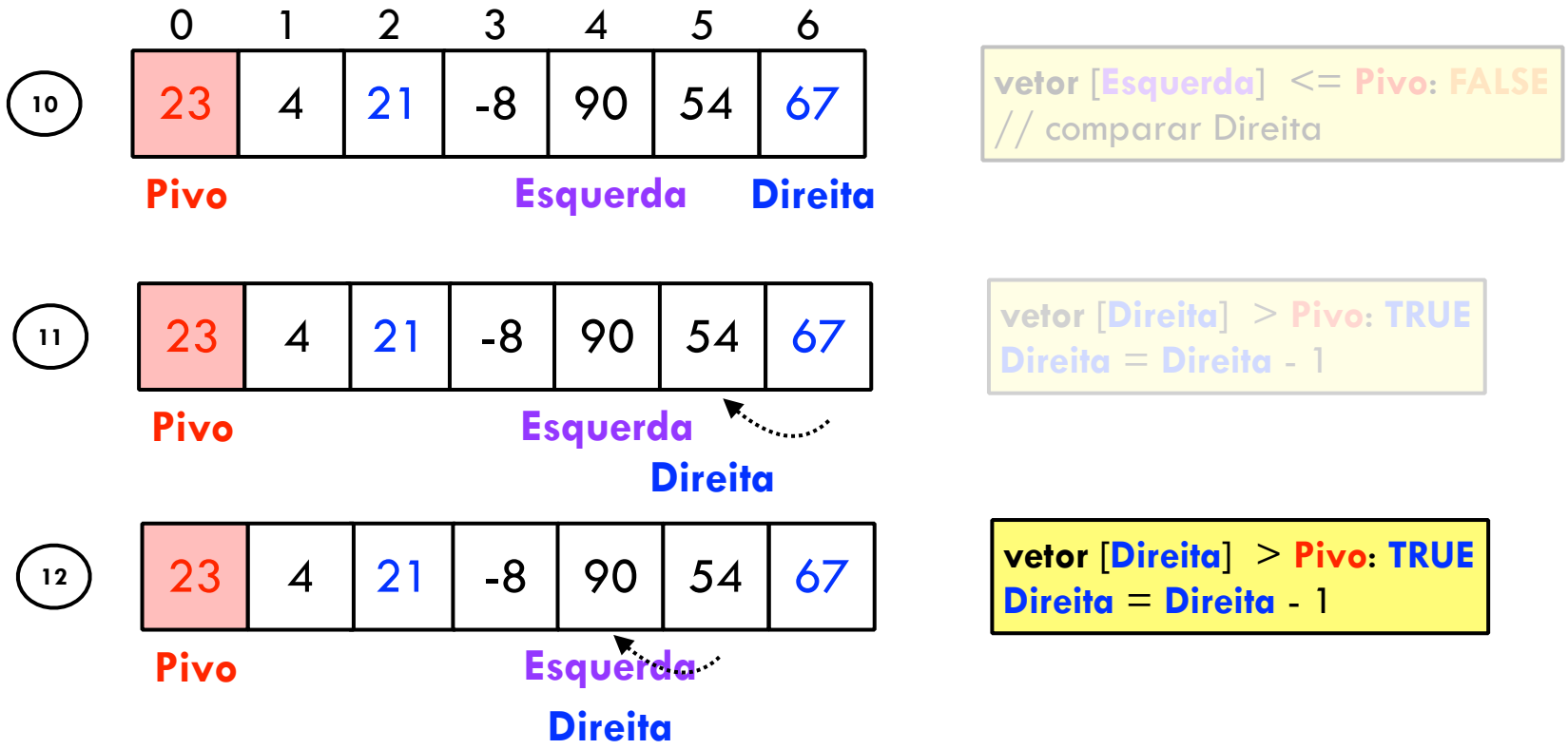


vetor [Esquerda] <= Pivo: FALSE
// comparar Direita

vetor [Direita] > Pivo: TRUE
Direita = Direita - 1

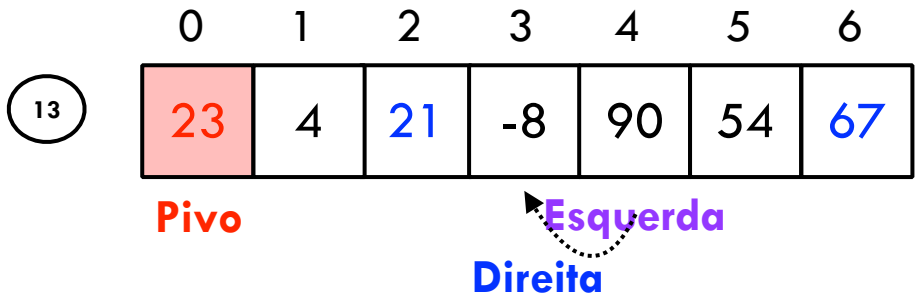
Exemplo

Particiona(V, 0, 6)



Exemplo

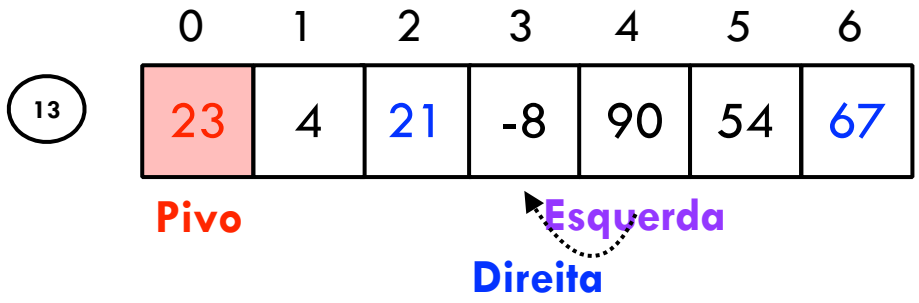
Particiona(V, 0, 6)



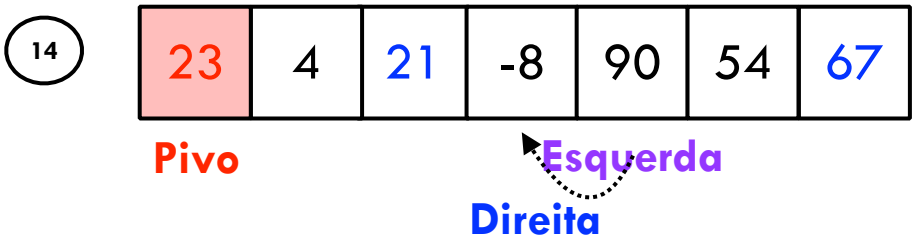
vetor [Direita] > Pivo: TRUE
Direita = Direita - 1

Exemplo

Particiona(V, 0, 6)



vetor [Direita] > Pivo: TRUE
Direita = Direita - 1



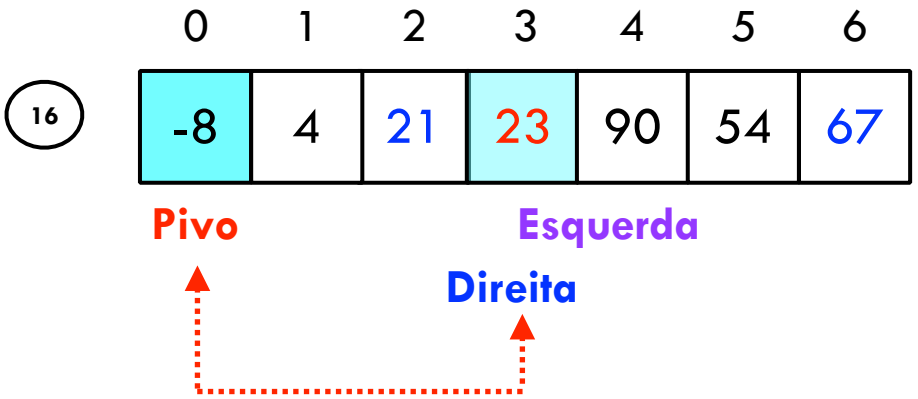
Esquerda < Direita? FALSE
Não trocar

Exemplo

Particiona(V, 0, 6)

Exemplo

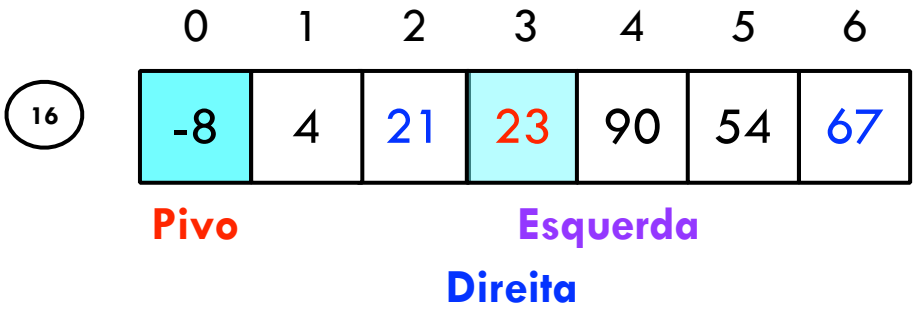
Particiona(V, 0, 6)



Trocar:
vetor[Direita] e **vetor[Inicio]**

Exemplo

Particiona(V, 0, 6)

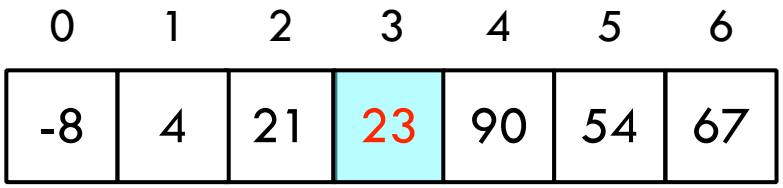


Trocar:
vetor[Direita] e **vetor[Inicio]**

Retornar: **Direita (3)**

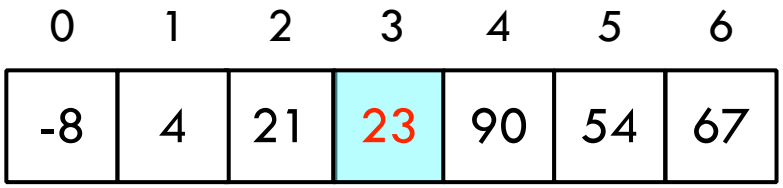
Exemplo

Particiona(V, 0, 6)



Exemplo

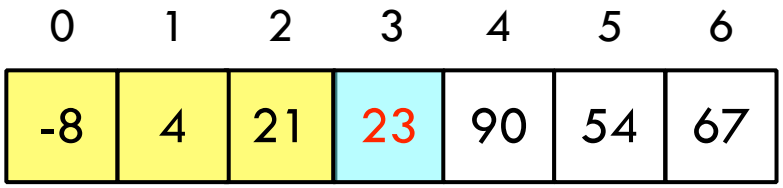
Particiona(V, 0, 6)



23 já está na posição correta!

Exemplo

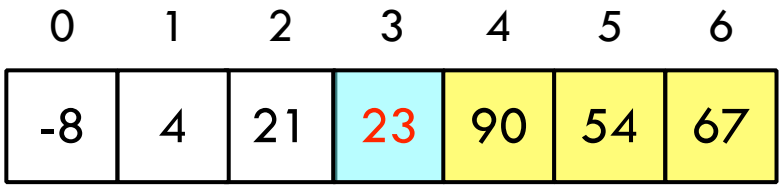
Particiona(V, 0, 6)



Elementos menores que 23

Exemplo

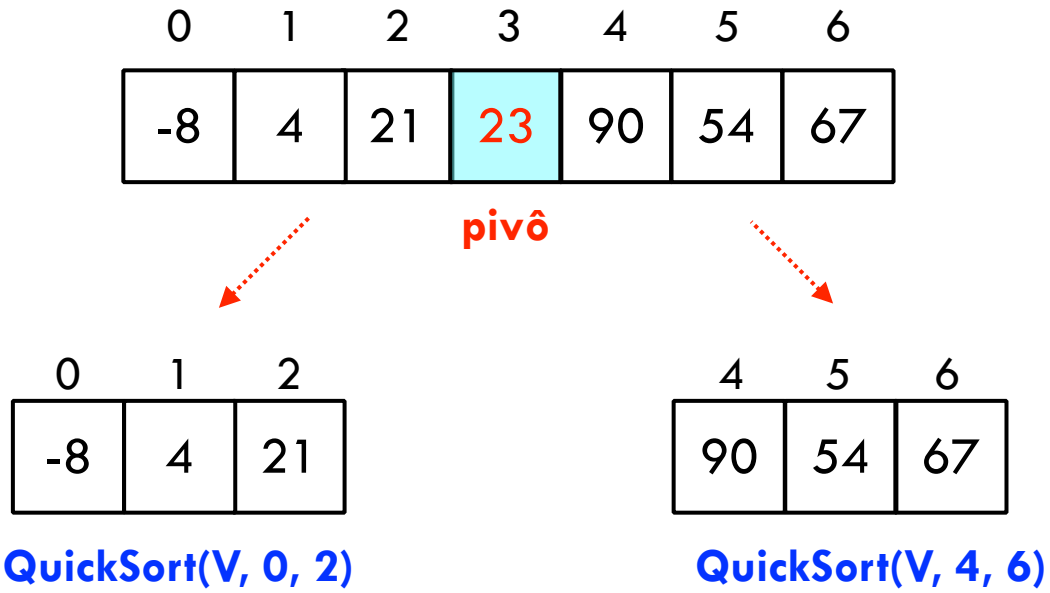
Particiona(V, 0, 6)



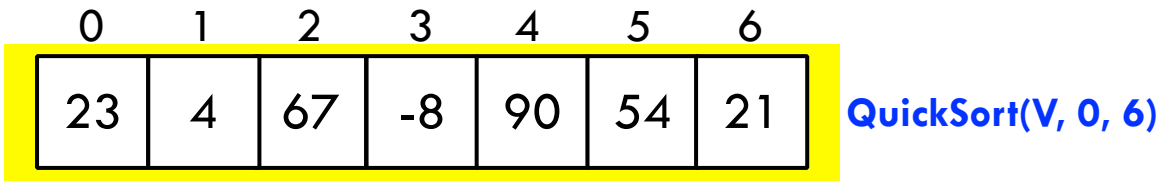
Elementos maiores que 23

Exemplo

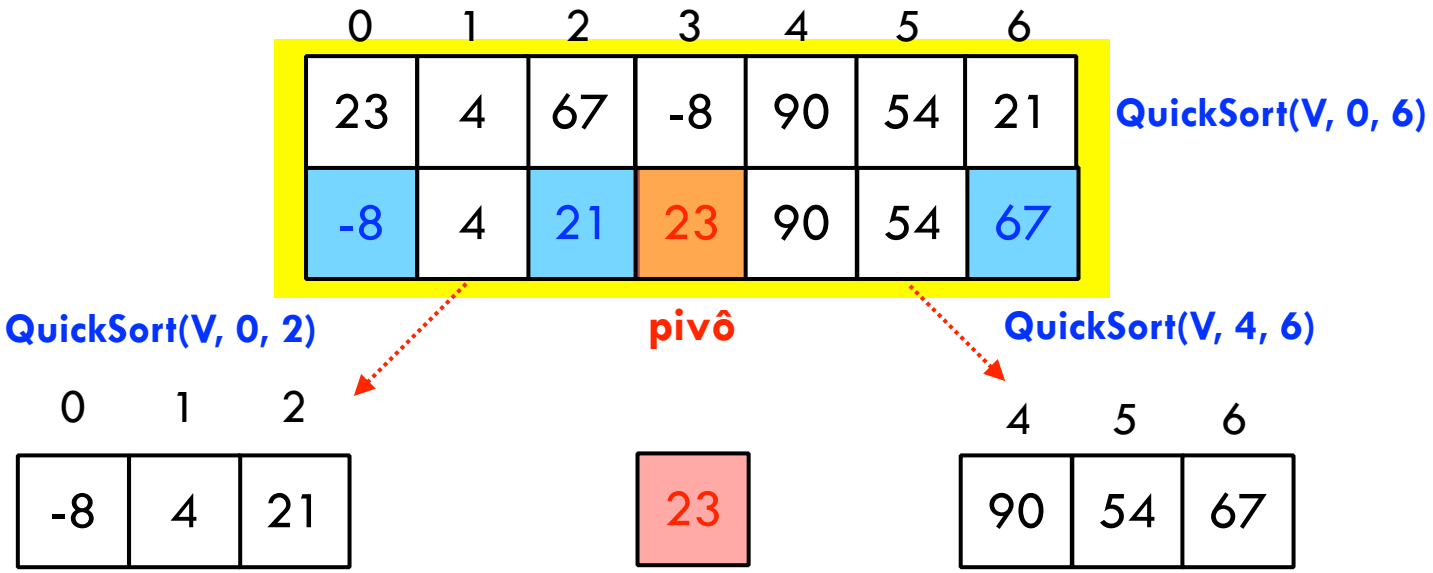
Particiona(V, 0, 6)



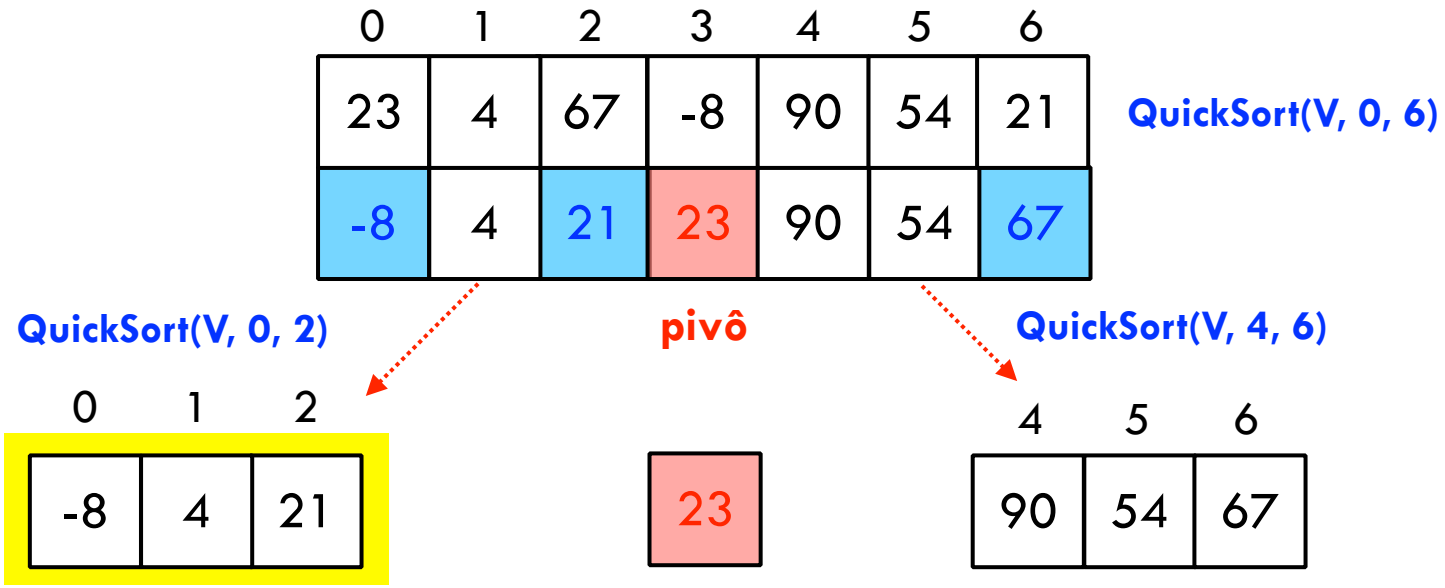
Resumindo



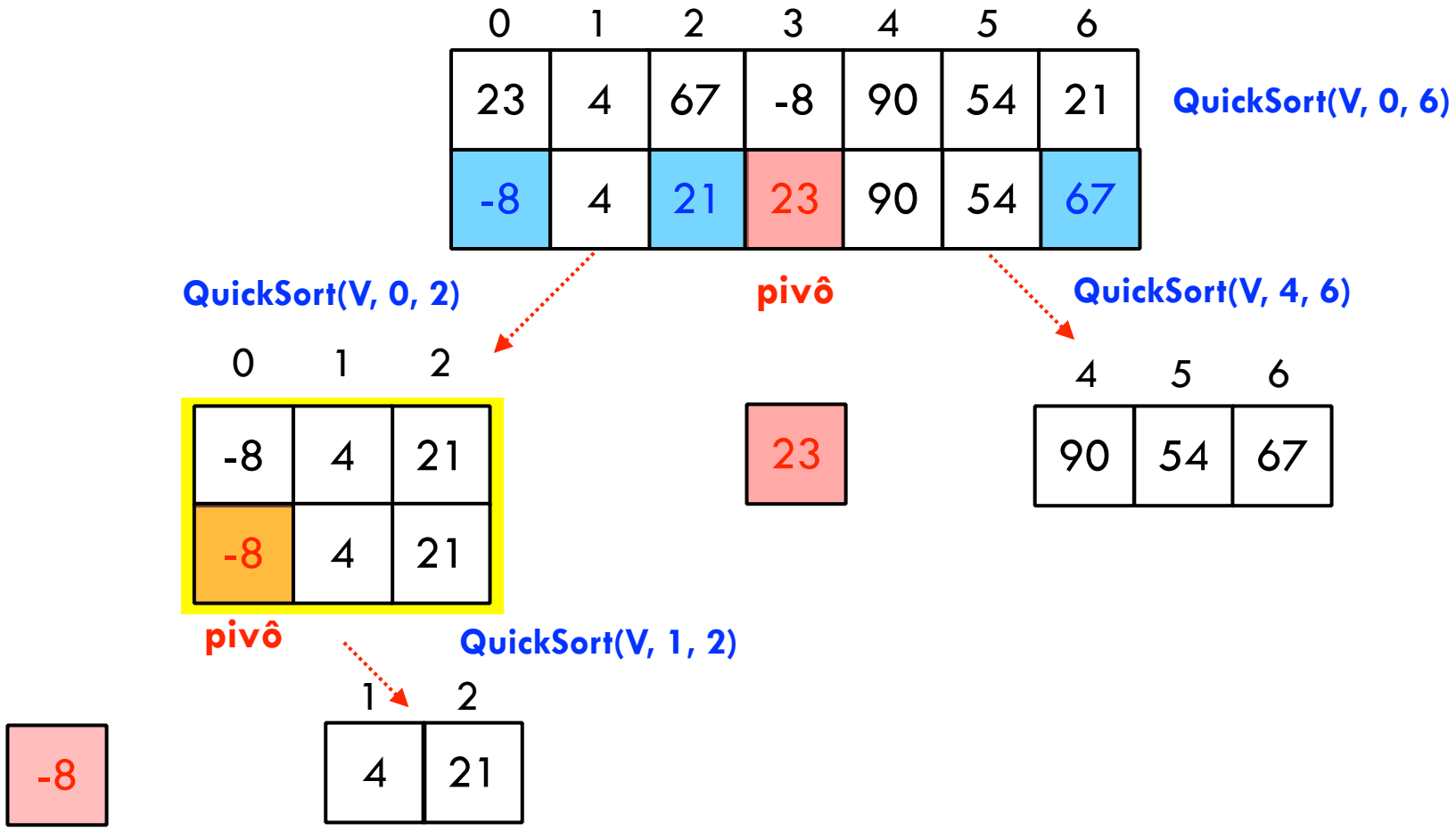
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
23
4 5 6
90 54 67

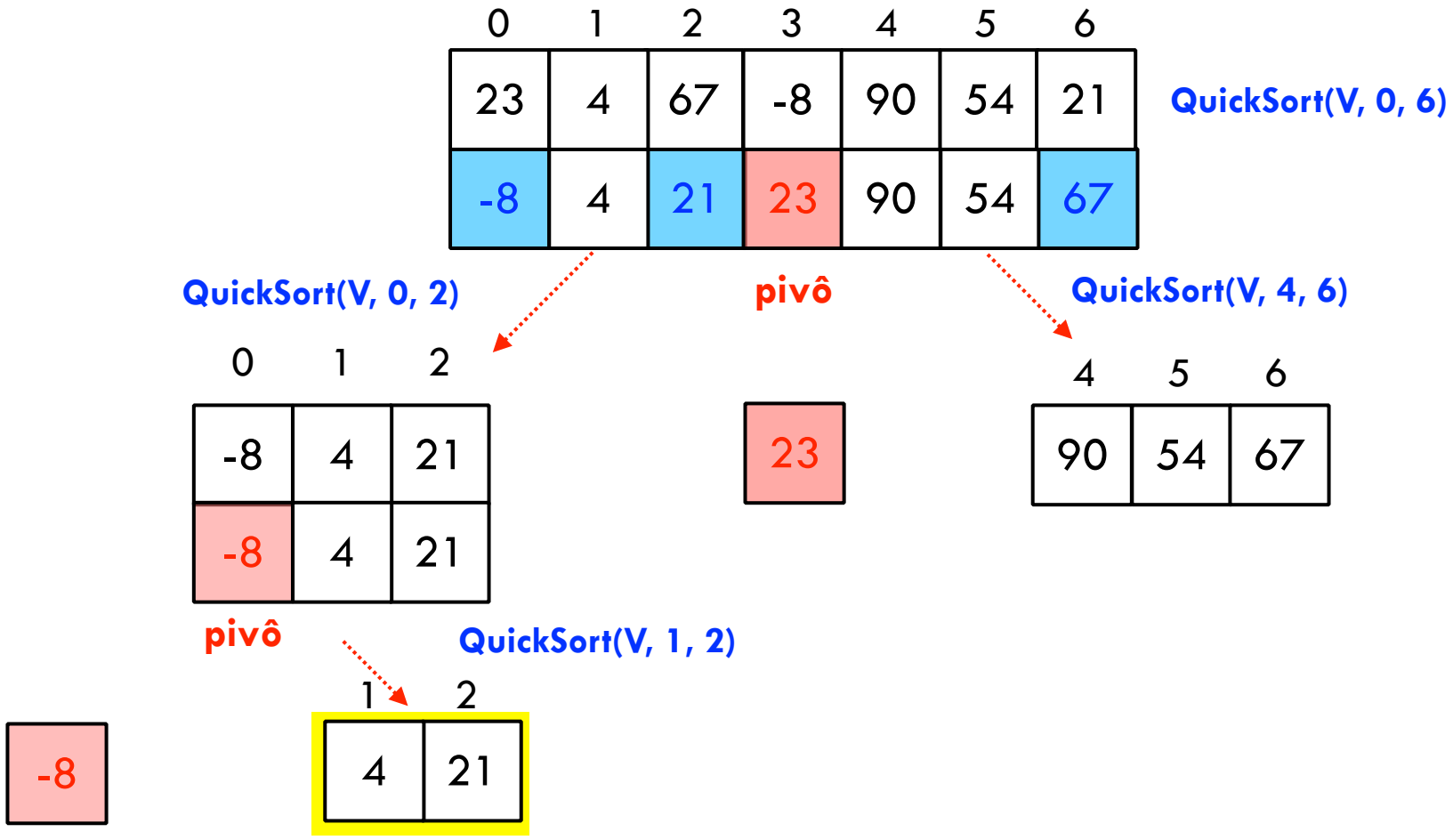
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
QuickSort(V, 0, 6)
-8 4 21 23 90 54 67
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
23
4 5 6
90 54 67

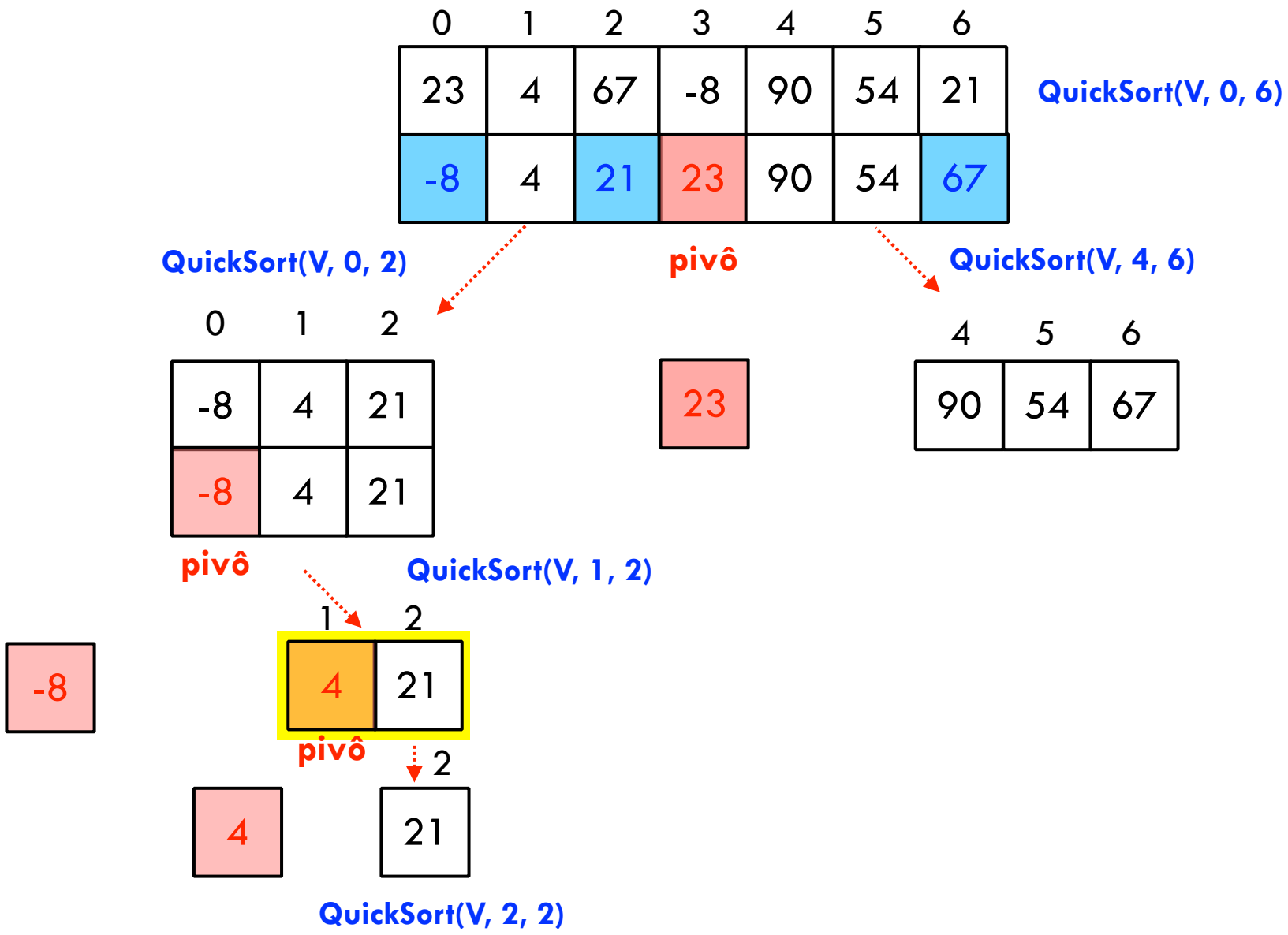
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
pivô
QuickSort(V, 1, 2)
1 2
4 21
-8

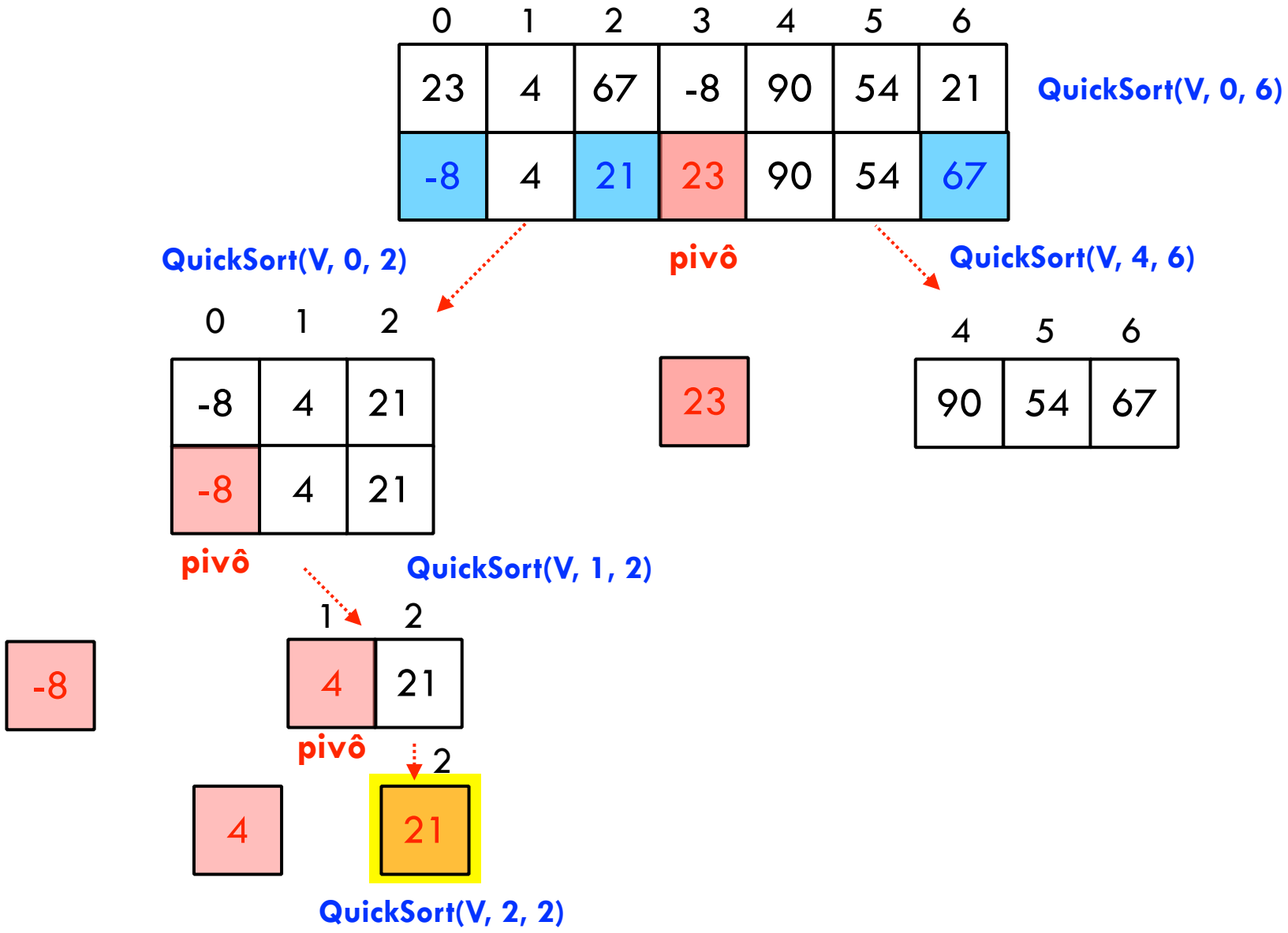
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
pivô
QuickSort(V, 1, 2)
1 2
4 21
-8

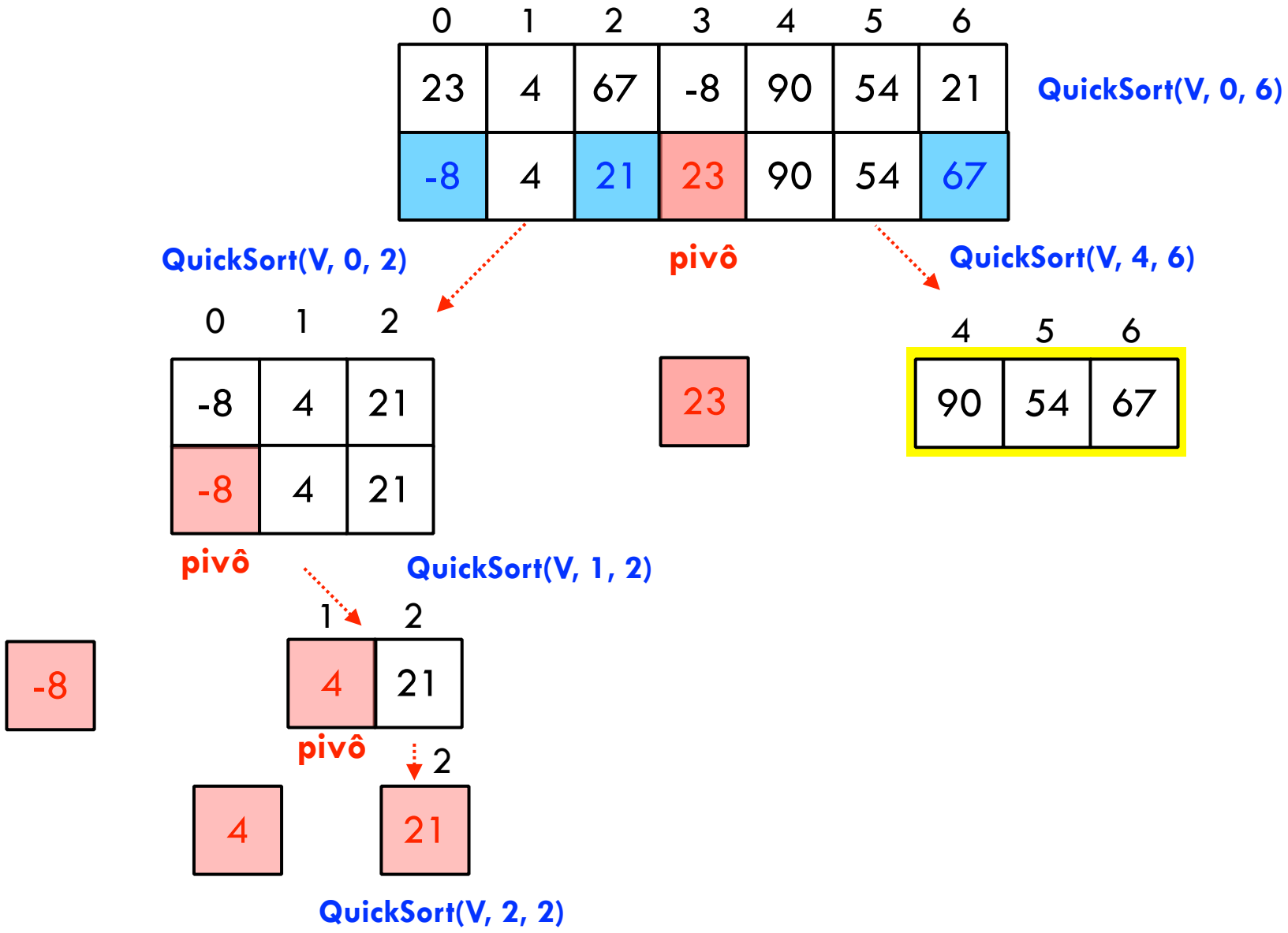
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
QuickSort(V, 0, 6)
-8 4 21 23 90 54 67
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
pivô
QuickSort(V, 1, 2)
-8
1 2
4 21
pivô
2
4
21
QuickSort(V, 2, 2)

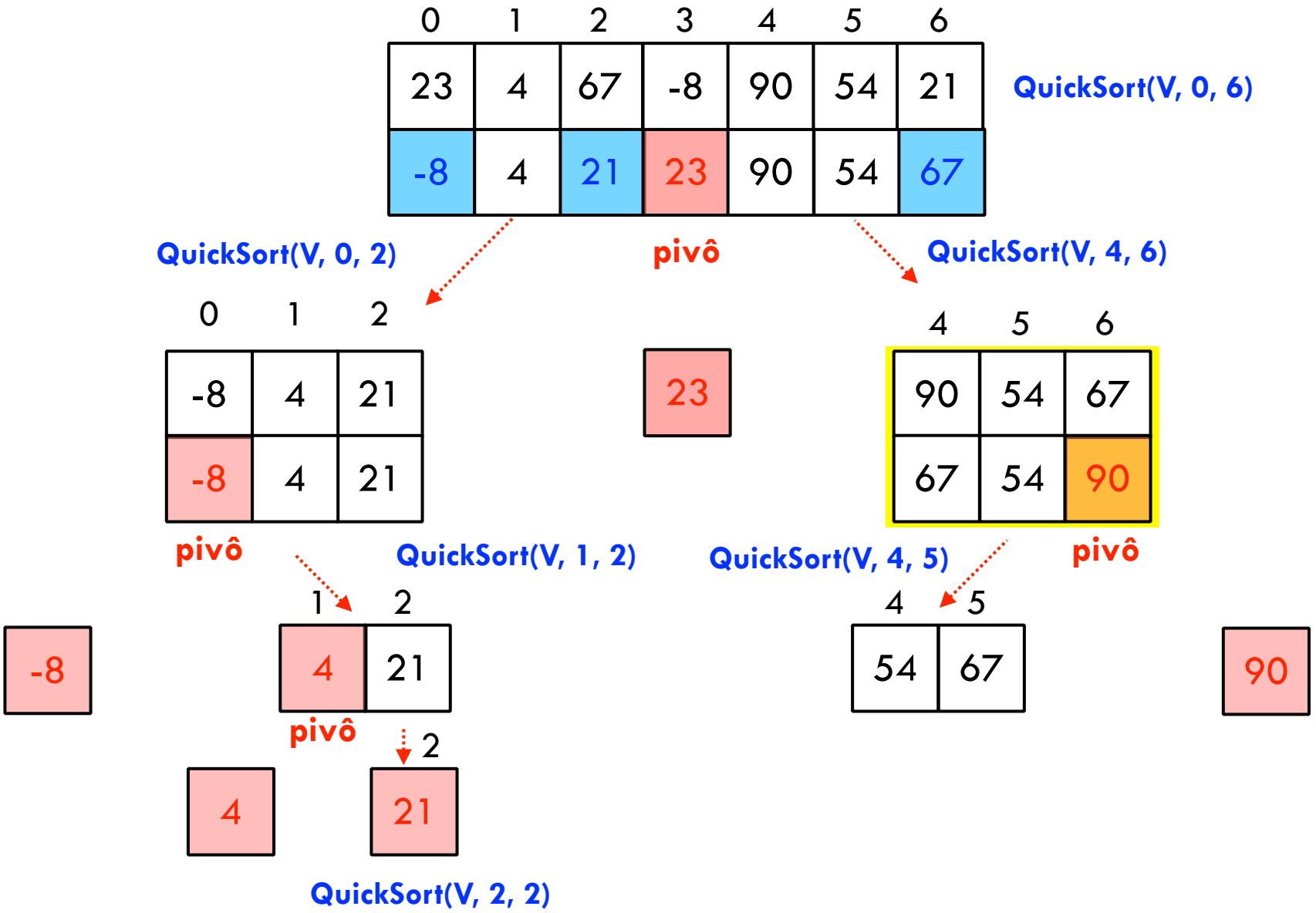
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
pivô
QuickSort(V, 1, 2)
-8
1 2
4 21
pivô
2
4
21
QuickSort(V, 2, 2)

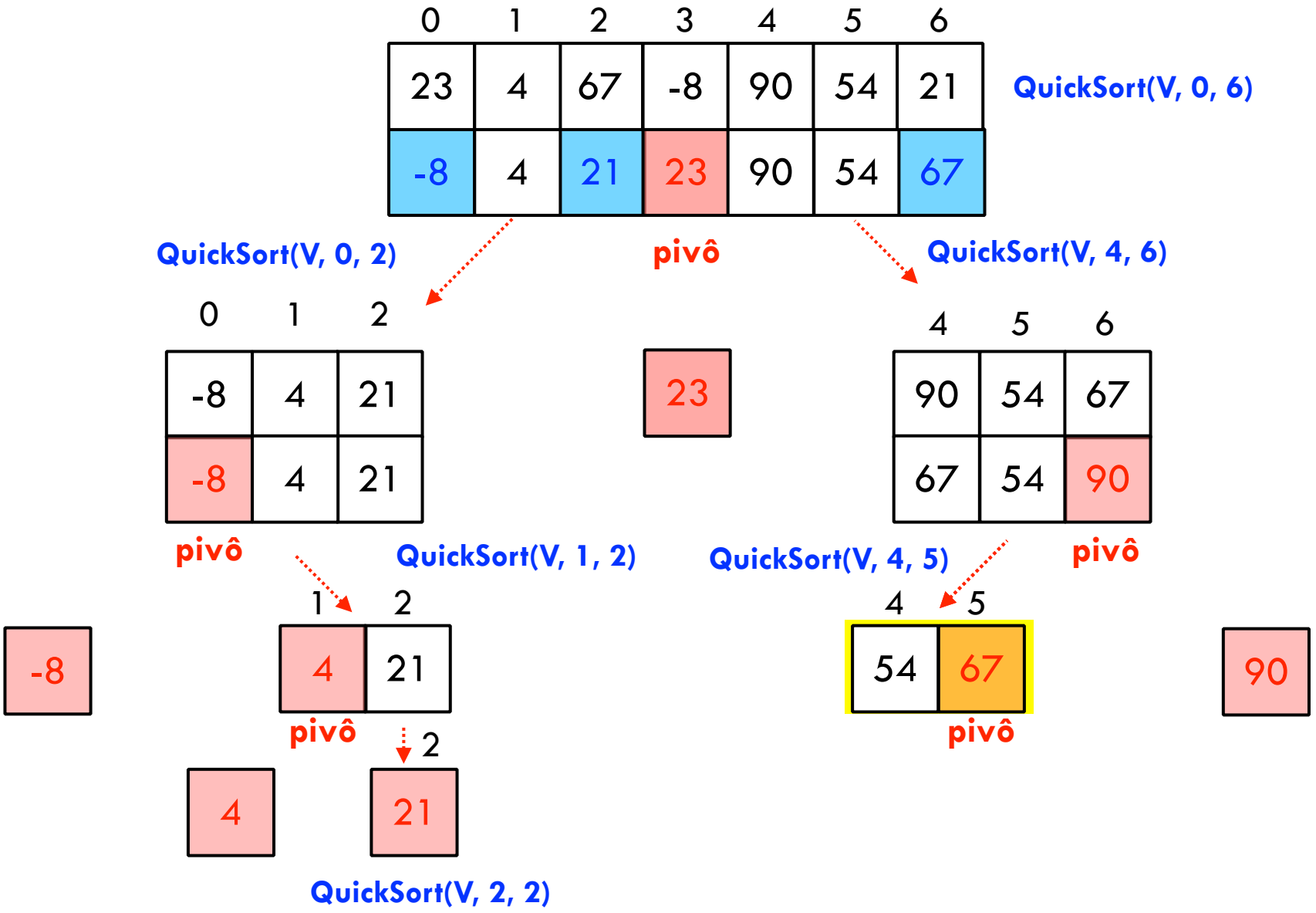
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
pivô
QuickSort(V, 1, 2)
-8
1 2
4 21
pivô
2
4
21
QuickSort(V, 2, 2)

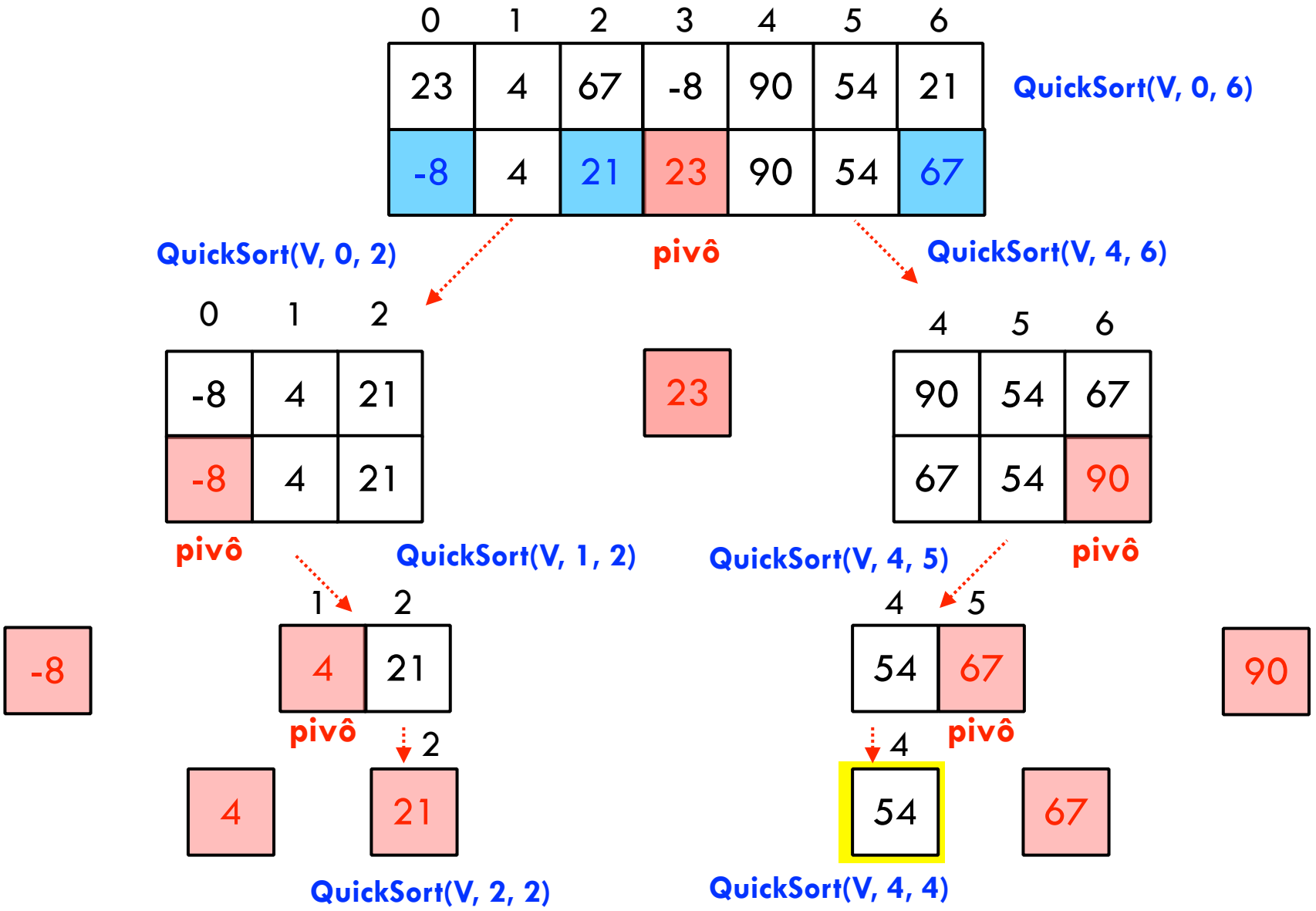
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
67 54 90
pivô
QuickSort(V, 1, 2)
QuickSort(V, 4, 5)
pivô
-8
1 2
4 21
pivô
4 5
54 67
90
2
4
21
QuickSort(V, 2, 2)

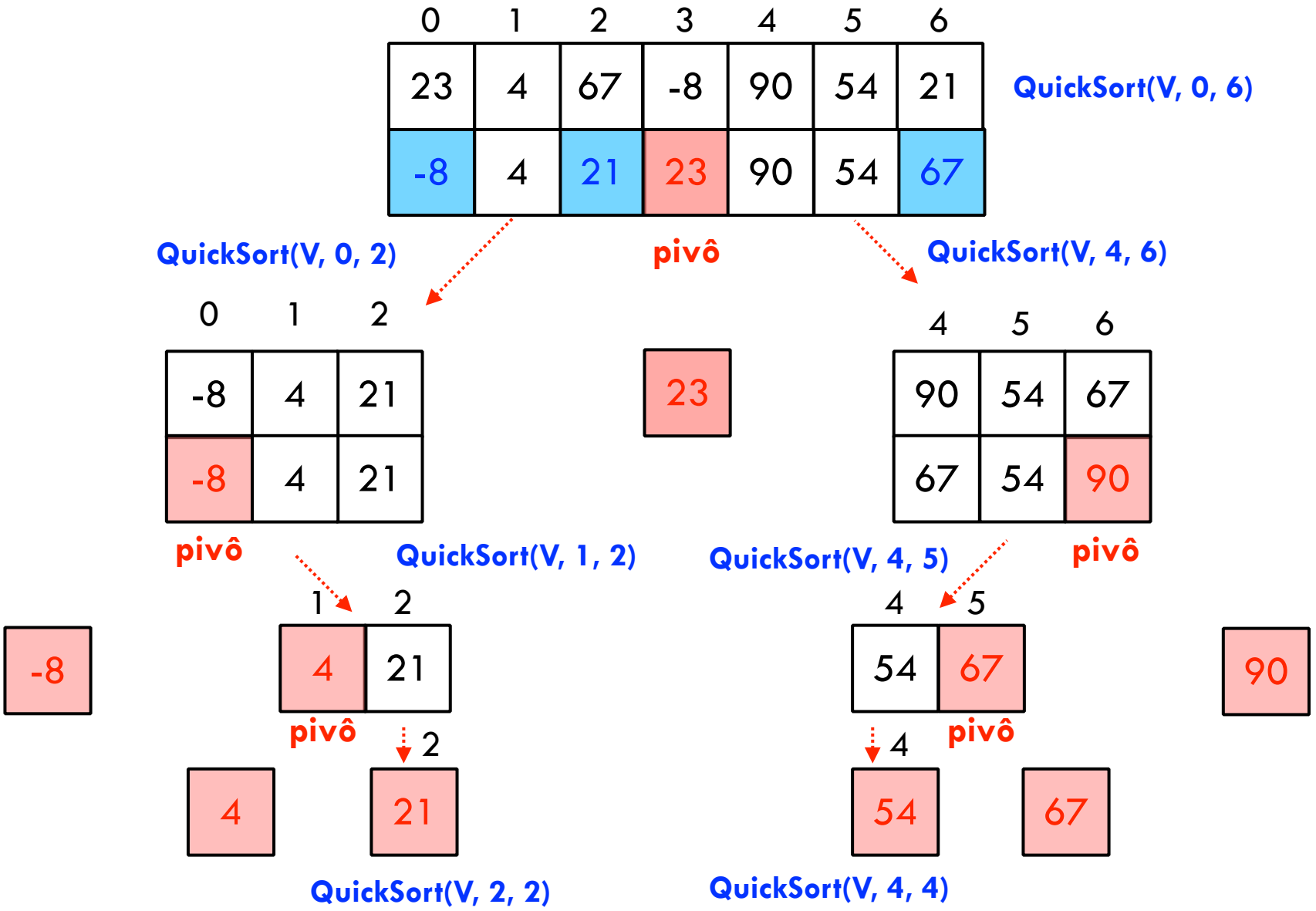
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
67 54 90
pivô
QuickSort(V, 1, 2)
QuickSort(V, 4, 5)
pivô
-8
1 2
4 21
pivô
4 5
54 67
pivô
90
2
4
21
QuickSort(V, 2, 2)

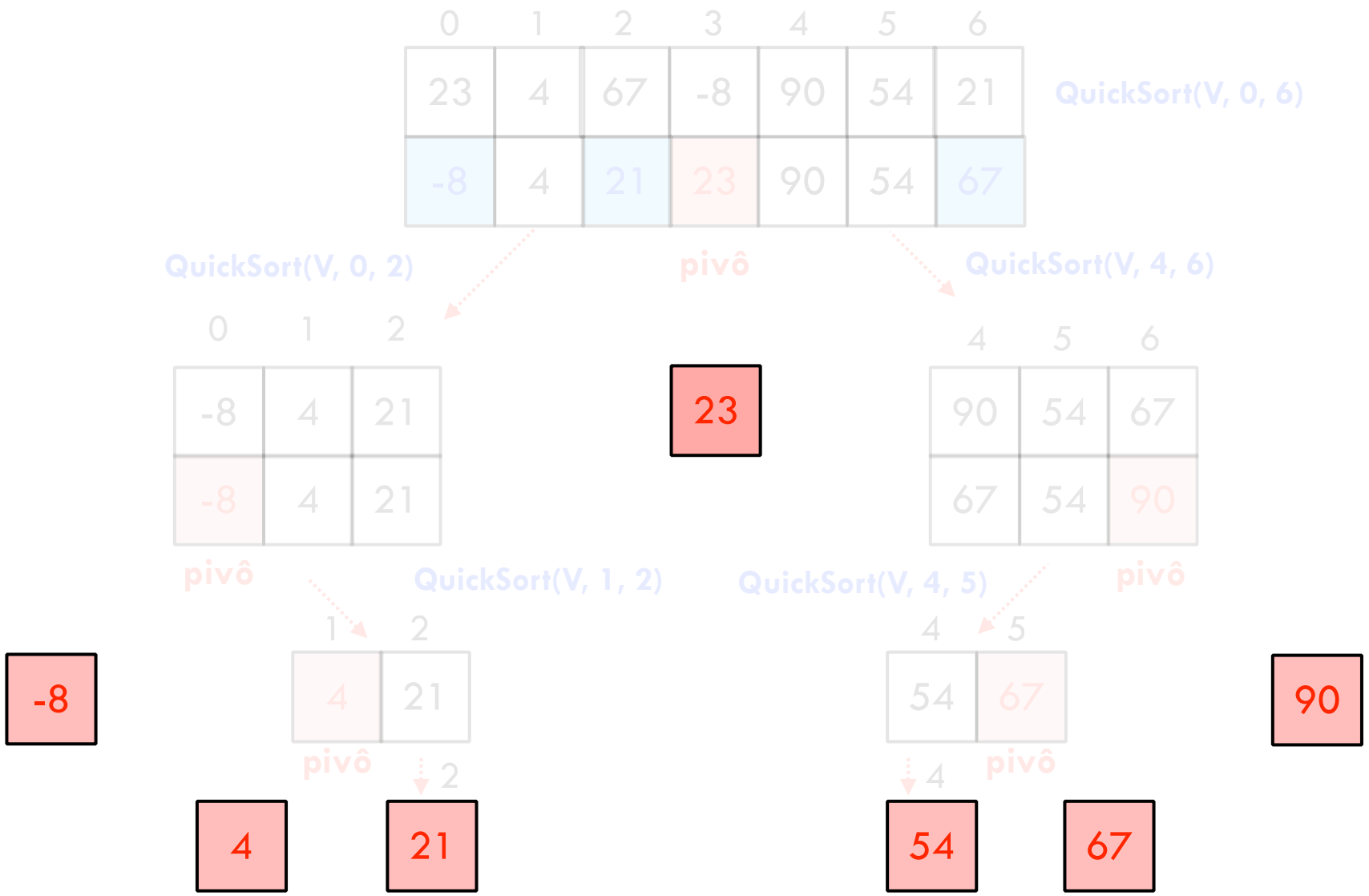
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
67 54 90
pivô
QuickSort(V, 1, 2)
QuickSort(V, 4, 5)
pivô
-8
1 2
4 21
pivô
2
4 5
54 67
90
4
pivô
4
21
4
54
67
QuickSort(V, 2, 2)
QuickSort(V, 4, 4)

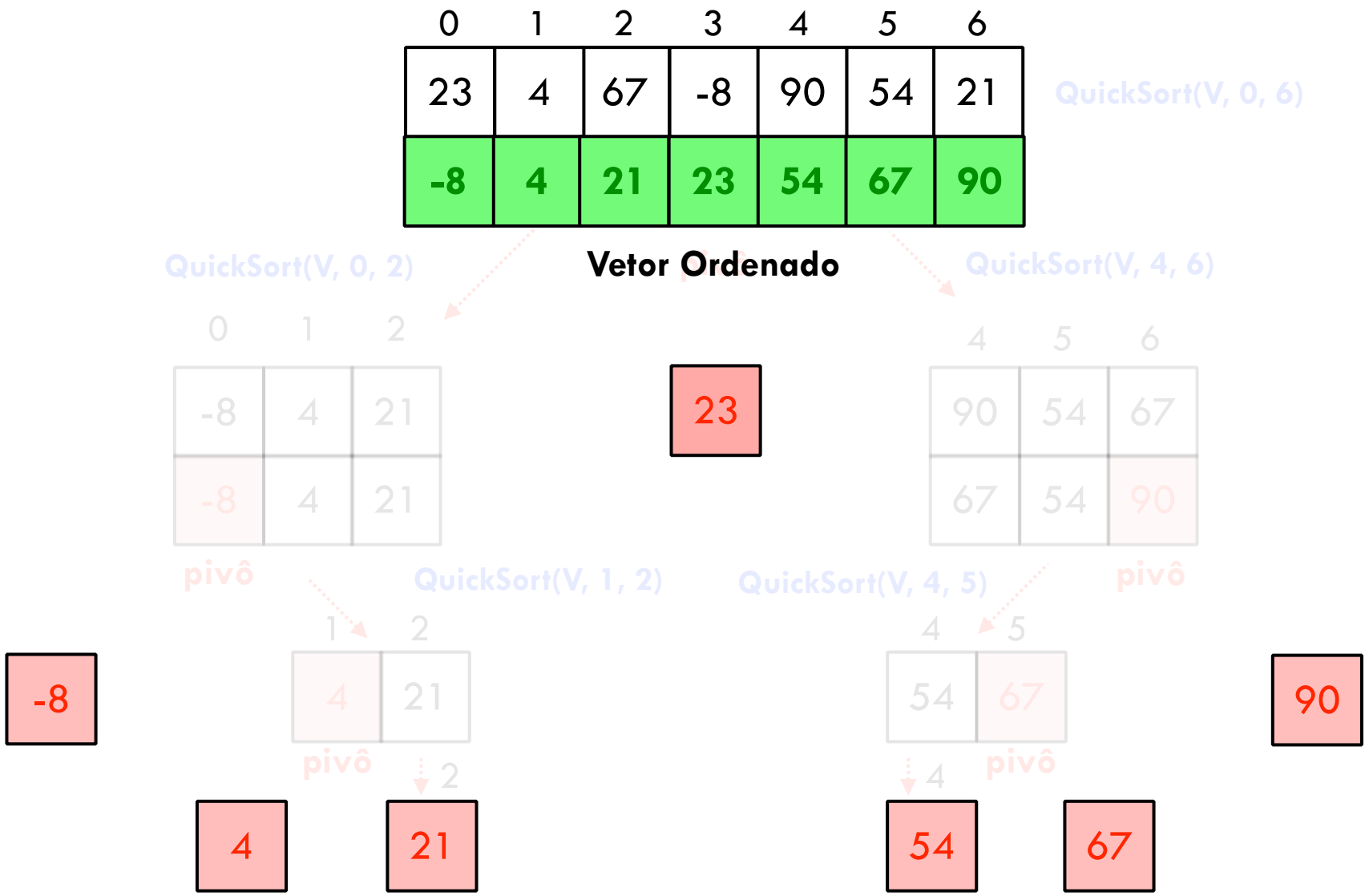
Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
23
4 5 6
90 54 67
67 54 90
pivô
QuickSort(V, 1, 2)
QuickSort(V, 4, 5)
pivô
-8
1 2
4 21
pivô
2
4 5
54 67
90
4
pivô
4
21
4
54
67
QuickSort(V, 2, 2)
QuickSort(V, 4, 4)

Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
QuickSort(V, 0, 6)
QuickSort(V, 0, 2)
pivô
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
4 5 6
90 54 67
67 54 90
23
pivô
QuickSort(V, 1, 2)
QuickSort(V, 4, 5)
pivô
1 2
4 21
54 67
-8
90
pivô
2
4
pivô
4
4
21
54
67

Resumindo


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
-8 4 21 23 54 67 90
QuickSort(V, 0, 6)
Vetor Ordenado
QuickSort(V, 0, 2)
QuickSort(V, 4, 6)
0 1 2
-8 4 21
-8 4 21
pivô
23
4 5 6
90 54 67
67 54 90
pivô
QuickSort(V, 1, 2)
QuickSort(V, 4, 5)
1 2
4 21
pivô
2
4 5
54 67
pivô
4
-8
4
21
54
67
90

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) Execute o teste de mesa (simulação) do algoritmo **Quick Sort** para a sua sequência de números aleatórios, definida na planilha da disciplina.

Exercícios

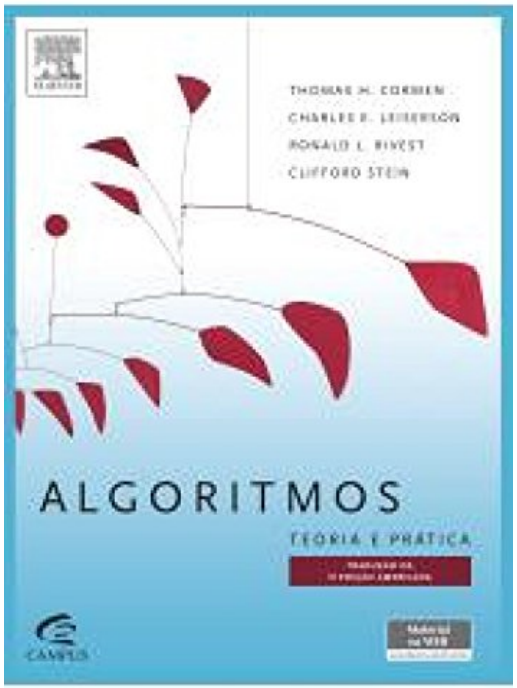
2) Implemente o **mergeSort** em **Python** considerando a seguinte assinatura de função:

```
/* Ordena o vetor usando Quick Sort
Parâmetros:
   array: vetor a ser ordenado
   option: 1 - ordenação crescente, 2 - ordenação decrescente

Esse algoritmo tem um comportamento assintótico O(N log N) */

def quickSort(array, option):
```

Roteiro

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

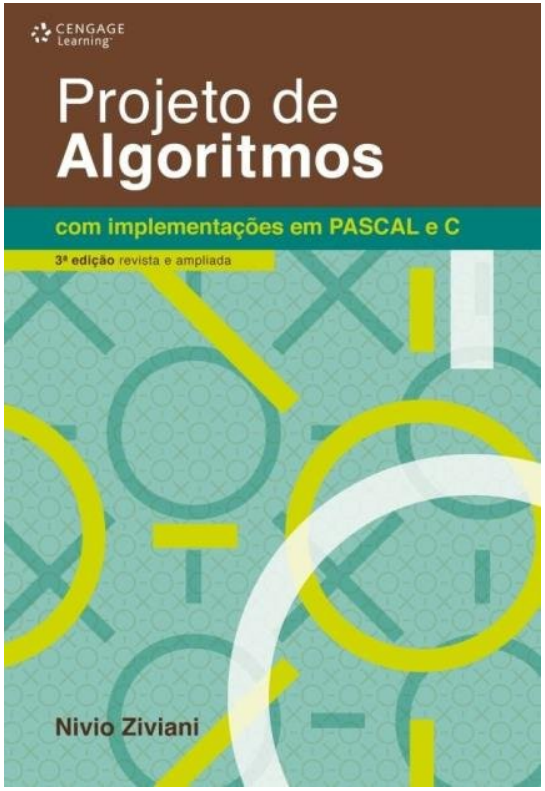
Referências sugeridas



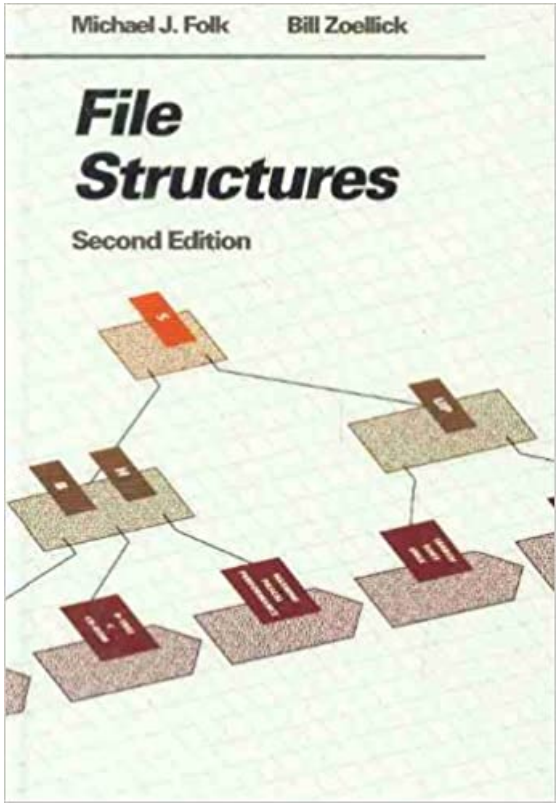
[Cormen et al, 2018]

[Drozdek, 2017]

Referências sugeridas



[Ziviani, 2010]



[Folk & Zoellick, 1992]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br