EDCO4B
ESTRUTURAS DE DADOS 2

Aula 05 - Merge Sort

Prof. Rafael G. Mantovani

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Apucarana - PR, Brasil

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia de Computação

Licença

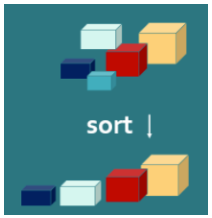
Roteiro

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

Roteiro

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

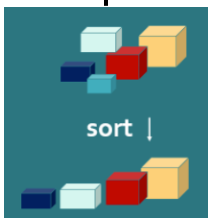
Introdução



Algoritmos de Ordenação

Introdução

Métodos Simples ⟷ Métodos Eficientes



Algoritmos de Ordenação

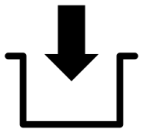
Introdução

Métodos Simples

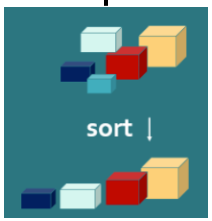
Métodos Eficientes

BubbleSort

InsertionSort



SelectionSort



Algoritmos de Ordenação

HeapSort

MergeSort

QuickSort

Introdução

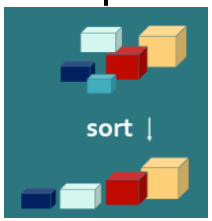
Métodos Simples

Métodos Eficientes

OK BubbleSort

OK InsertionSort

OK SelectionSort



Algoritmos de Ordenação

HeapSort

MergeSort

QuickSort

Roteiro

1. Introdução
2. **Merge Sort**
3. Exemplo
4. Exercício
5. Referências

Merge Sort

- Ordenação por Mistura

Merge Sort

- Ordenação por Mistura

Merge Sort

- Ordenação por Mistura

Merge Sort

- Ordenação por Mistura

League: levar os 6 mais fortes

Merge Sort

Merge Sort

Lv 65 Lv 16 Lv 54 Lv 15 Lv 60 Lv 5 Lv 40 Lv 39 Lv 50

Merge Sort

Lv 65 

Lv 16 

Lv 54 

Lv 15 

Lv 60 

Meio

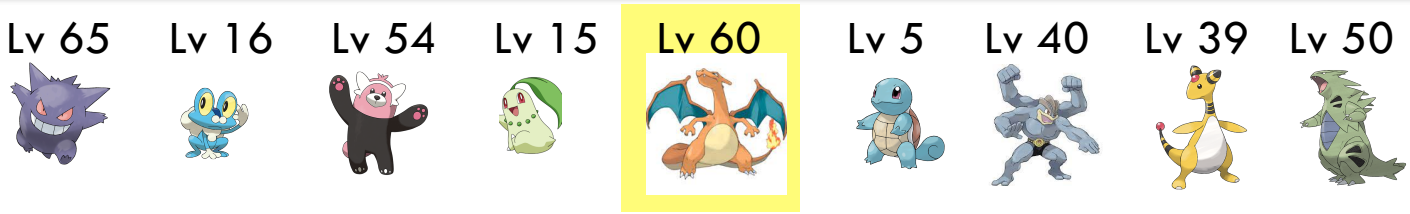
Lv 5 

Lv 40 

Lv 39 

Lv 50 

Merge Sort



Sub-problema: esquerda

Sub-problema: direita

Merge Sort

Lv 65 

Lv 16 

Lv 54 

Meio

Lv 15 

Lv 60 

Lv 5 

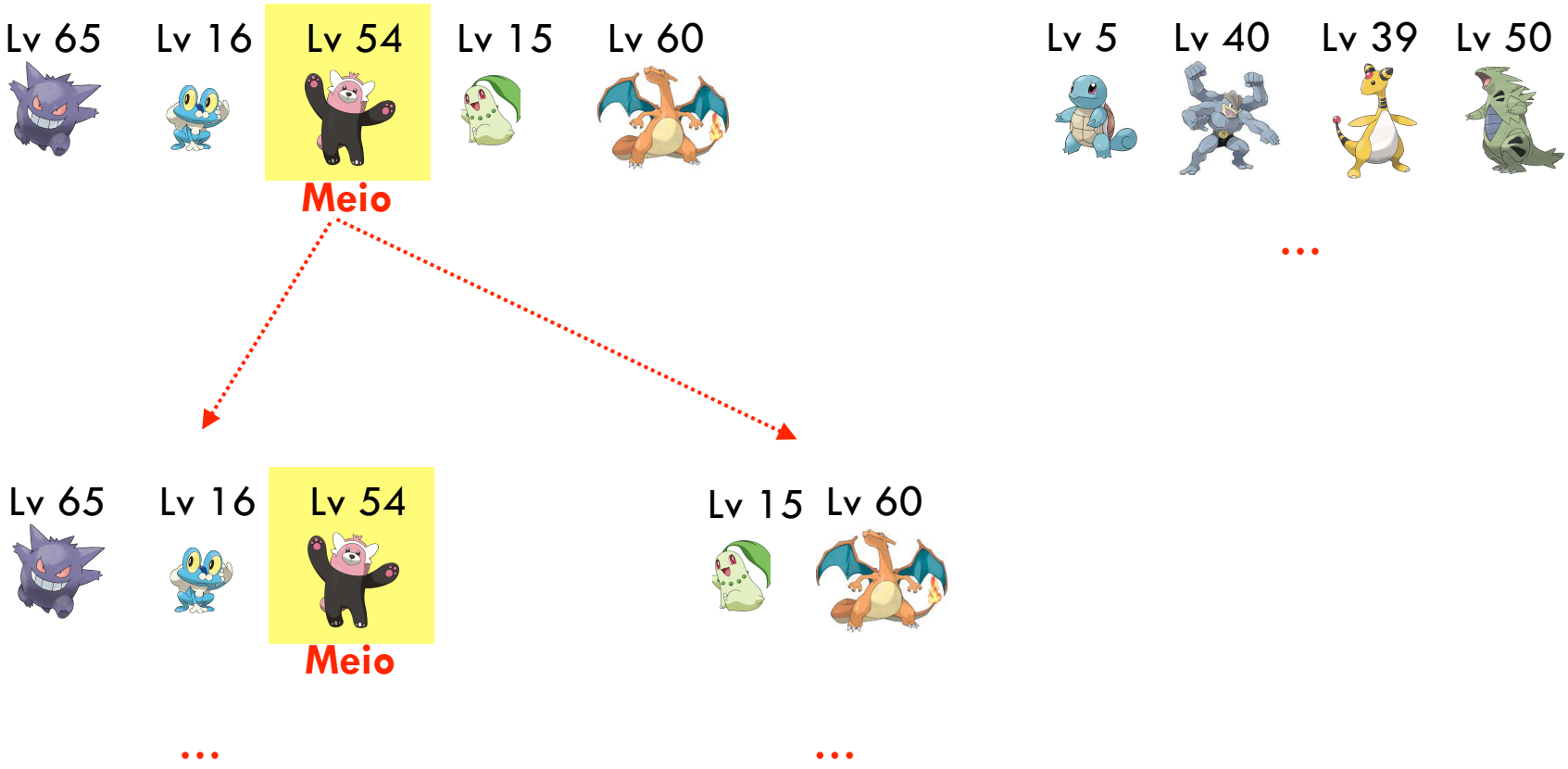
Lv 40 

Lv 39 

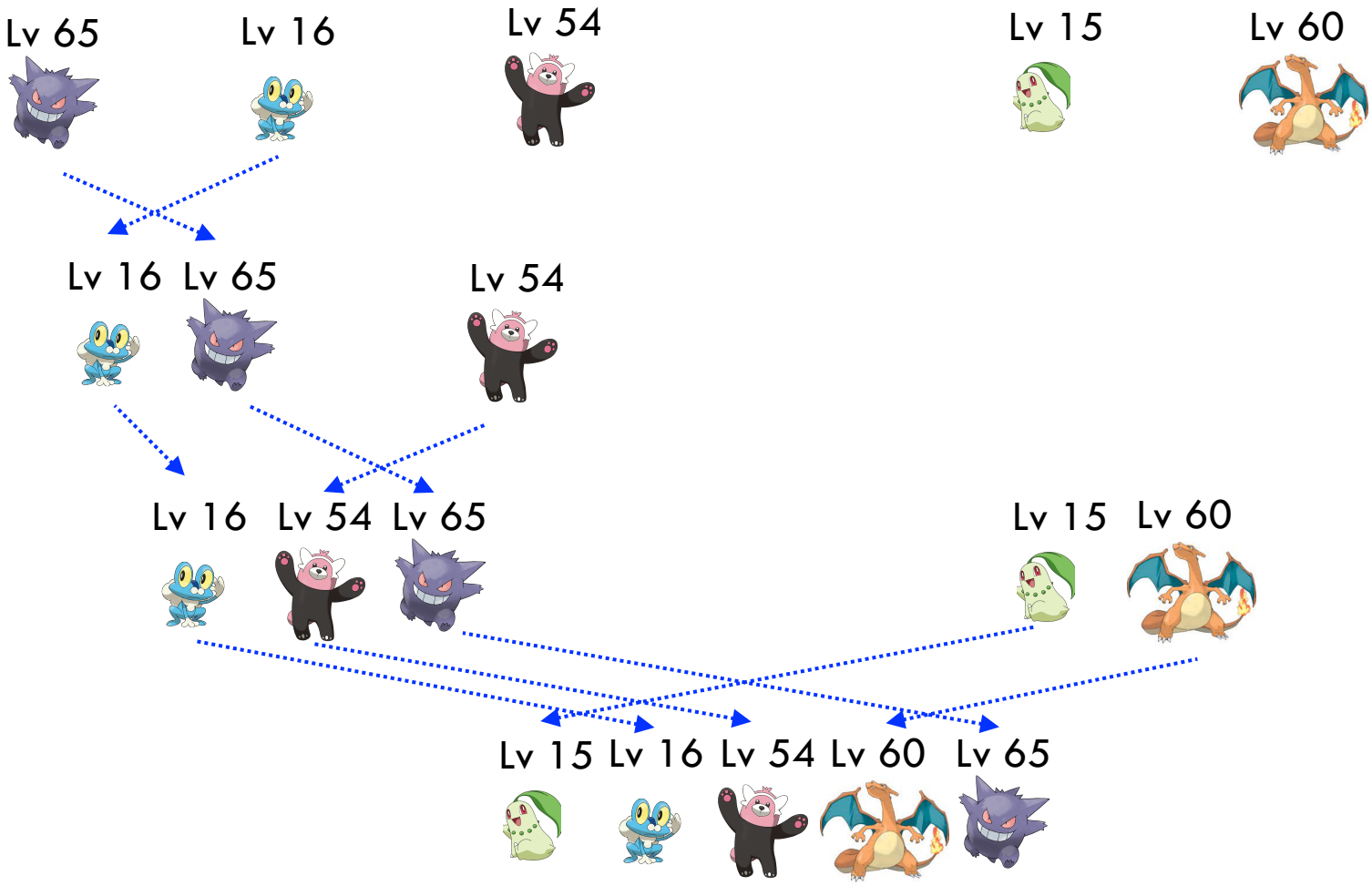
Lv 50 

...

Merge Sort


Lv 65
Lv 16
Lv 54
Lv 15
Lv 60
Meio
Lv 5
Lv 40
Lv 39
Lv 50
...
Lv 65
Lv 16
Lv 54
Meio
Lv 15
Lv 60
...
...

Merge Sort


Lv 65
Lv 16
Lv 54
Lv 15
Lv 60
Lv 16 Lv 65
Lv 54
Lv 16 Lv 54 Lv 65
Lv 15 Lv 60
Lv 15 Lv 16 Lv 54 Lv 60 Lv 65

Merge Sort

Lv 5 Lv 15 Lv 16 Lv 39 Lv 40 Lv 50 Lv 54 Lv 60 Lv 65

Merge Sort

Lv 5 Lv 15 Lv 16



Lv 39 Lv 40 Lv 50 Lv 54 Lv 60 Lv 65






Good Team :)

Merge Sort

- Ordenação por Mistura

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

Merge Sort

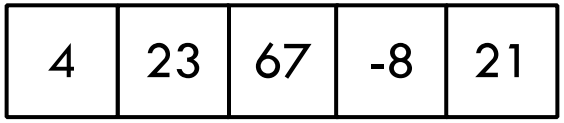
- **Funcionamento**

* Dividir e conquistar:

1. Divide recursivamente o array até obter subconjuntos com elementos únicos
2. Volta da recursão combinando 2 conjuntos de forma a obter um conjunto maior e mais ordenado
3. Processo se repete até que existe apenas um conjunto único e ordenado

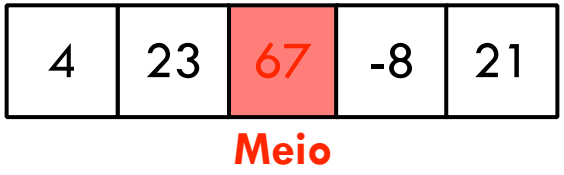
Merge Sort

- Dividir



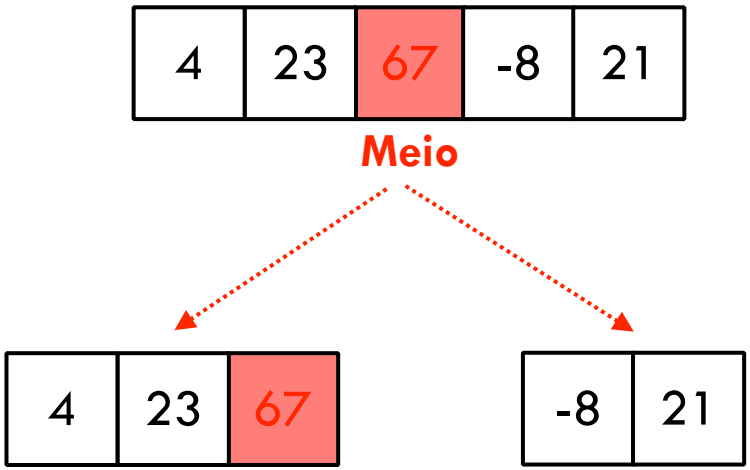
Merge Sort

- Dividir



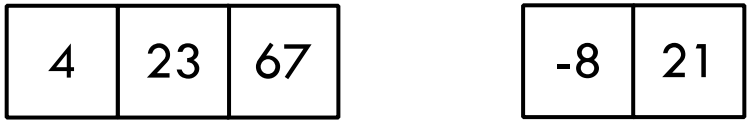
Merge Sort

- Dividir



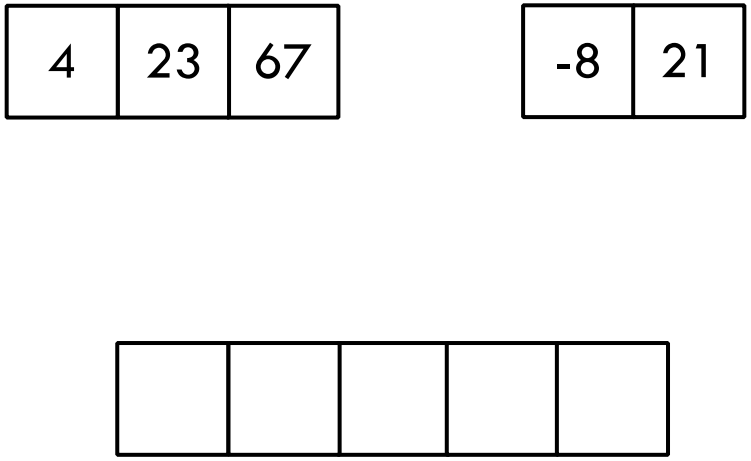
Merge Sort

- Recombinar



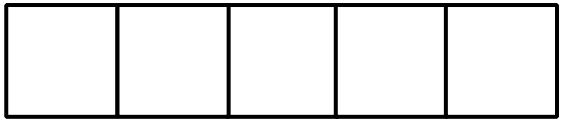
Merge Sort

- Recombinar



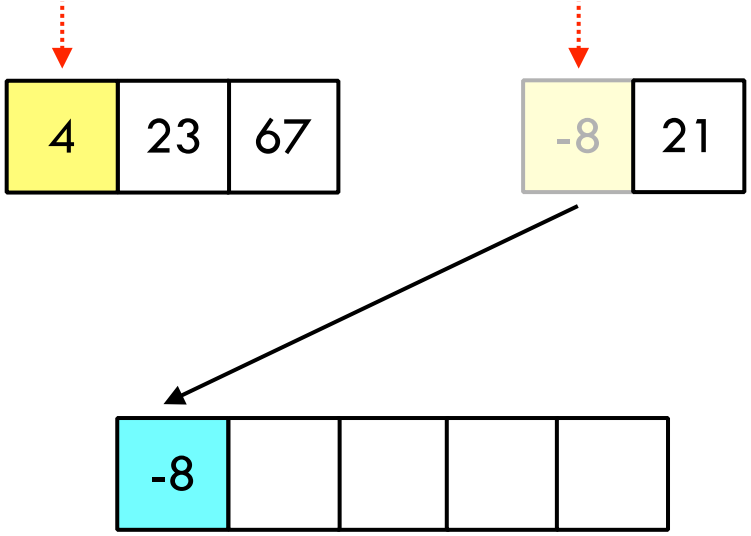
Merge Sort

- Recombinar



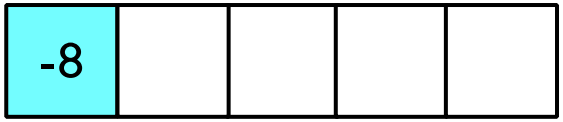
Merge Sort

- Recombinar



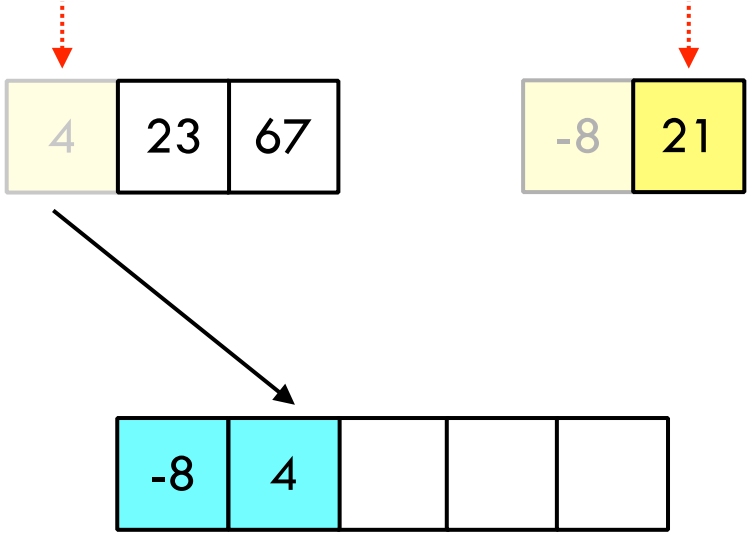
Merge Sort

- Recombinar



Merge Sort

- Recombinar

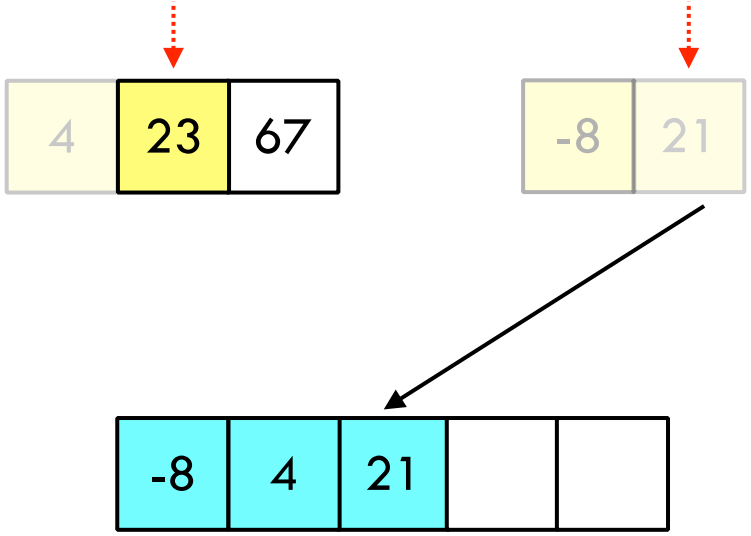


Merge Sort

- Recombinar

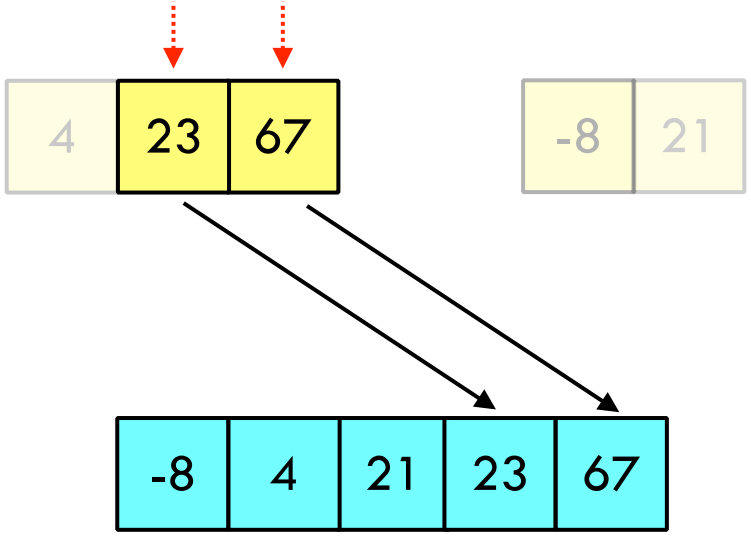
Merge Sort

- Recombinar



Merge Sort

- Recombinar



Merge Sort

- **Desempenho**

* **melhor caso:** O(N log N), elementos já ordenados

* **pior caso:** O(N log N), elementos na ordem decrescente

* **caso médio:** O(N log N)

Merge Sort

- **Desempenho**

* **melhor caso:** O(N log N), elementos já ordenados

* **pior caso:** O(N log N), elementos na ordem decrescente

* **caso médio:** O(N log N)

Obs: Eficiente para grandes conjuntos de dados. Estável.

Merge Sort

- **Pseudocódigo** (função principal)

```
1.  MergeSort (V, Inicio, Fim)
2.      Se (Inicio < Fim), então:
3.          Meio = ((Inicio + Fim)/2)
4.          MergeSort(V, Inicio, Meio)
5.          MergeSort(V, Meio+1, Fim)
6.          Merge(V, Inicio, Meio, Fim)
```

Merge Sort

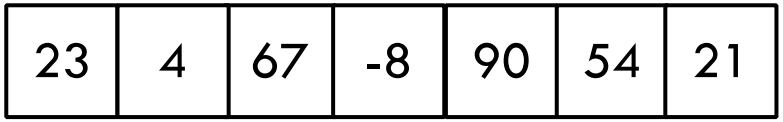
- **Pseudocódigo** (função auxiliar)

```
1.  Merge (V, Inicio, Meio, Fim)
2.      Alocar dinamicamente um vetor auxiliar
3.      P1 = Inicio
4.      P2 = Meio+1
5.      Enquanto (P1 < Meio E P2 < Fim) faça:
6.          Copia para o vetor auxiliar o menor valore entre V[P1] e V[P2]
7.          Incrementa o contador correspondente
8.      Se P1 == Meio então:
9.          Copia o que sobrou a partir de P2
10.     Senão:
11.         Copia o que sobrou a partir de P1
12.     Copia o vetor auxiliar no original
```

Roteiro

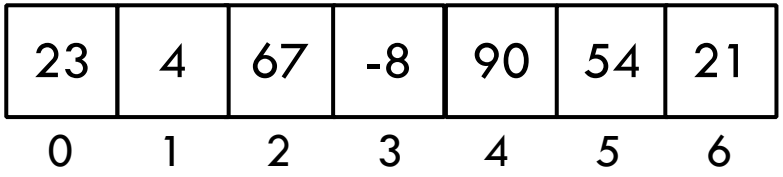
1. Introdução
2. Merge Sort
3. Exemplo
4. Exercícios
5. Referências

Exemplo

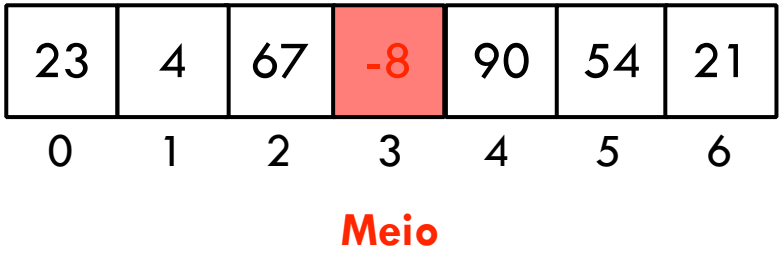


vetor não ordenado

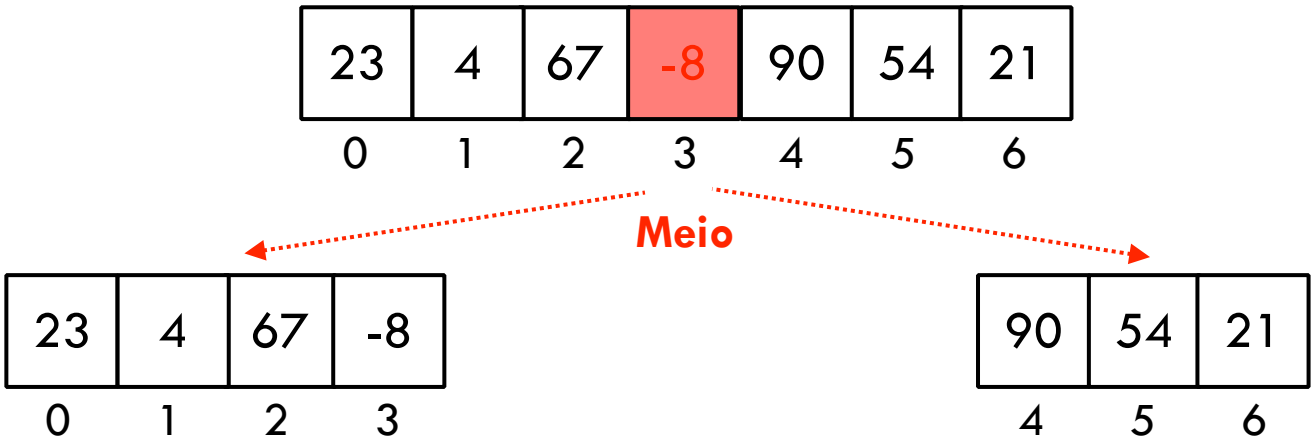
Exemplo



Exemplo

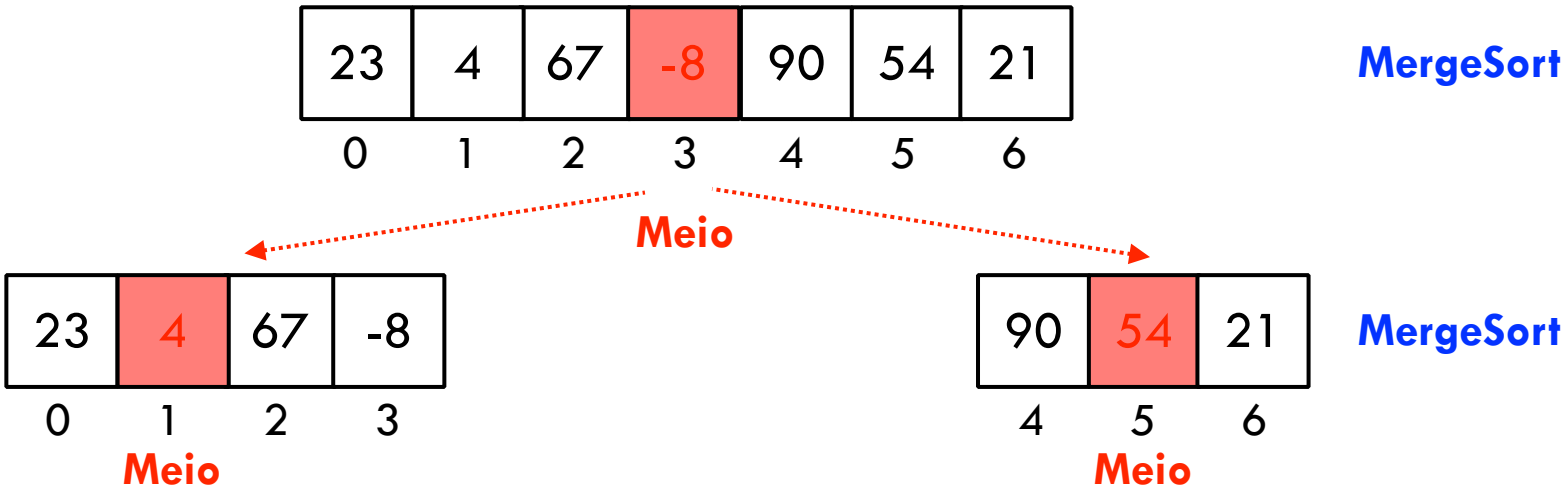


Exemplo

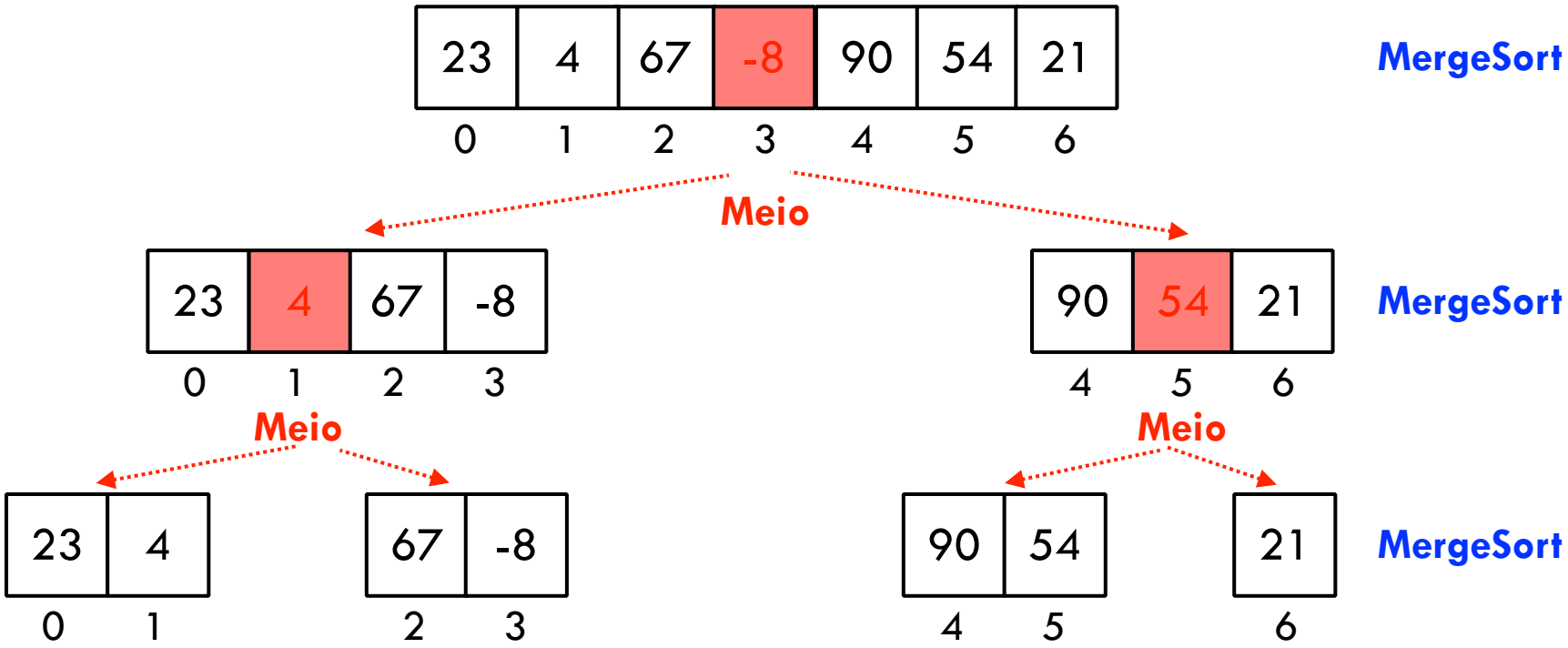


MergeSort

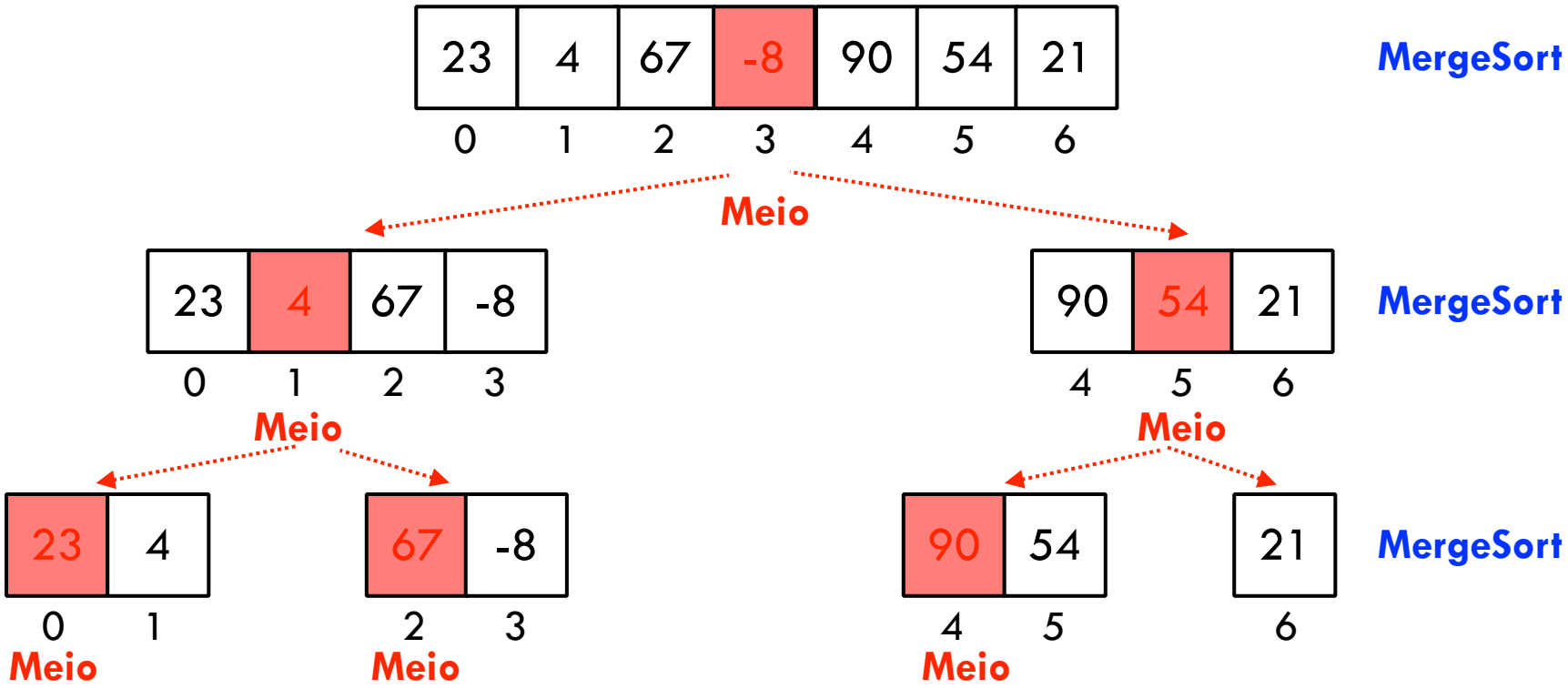
Exemplo


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

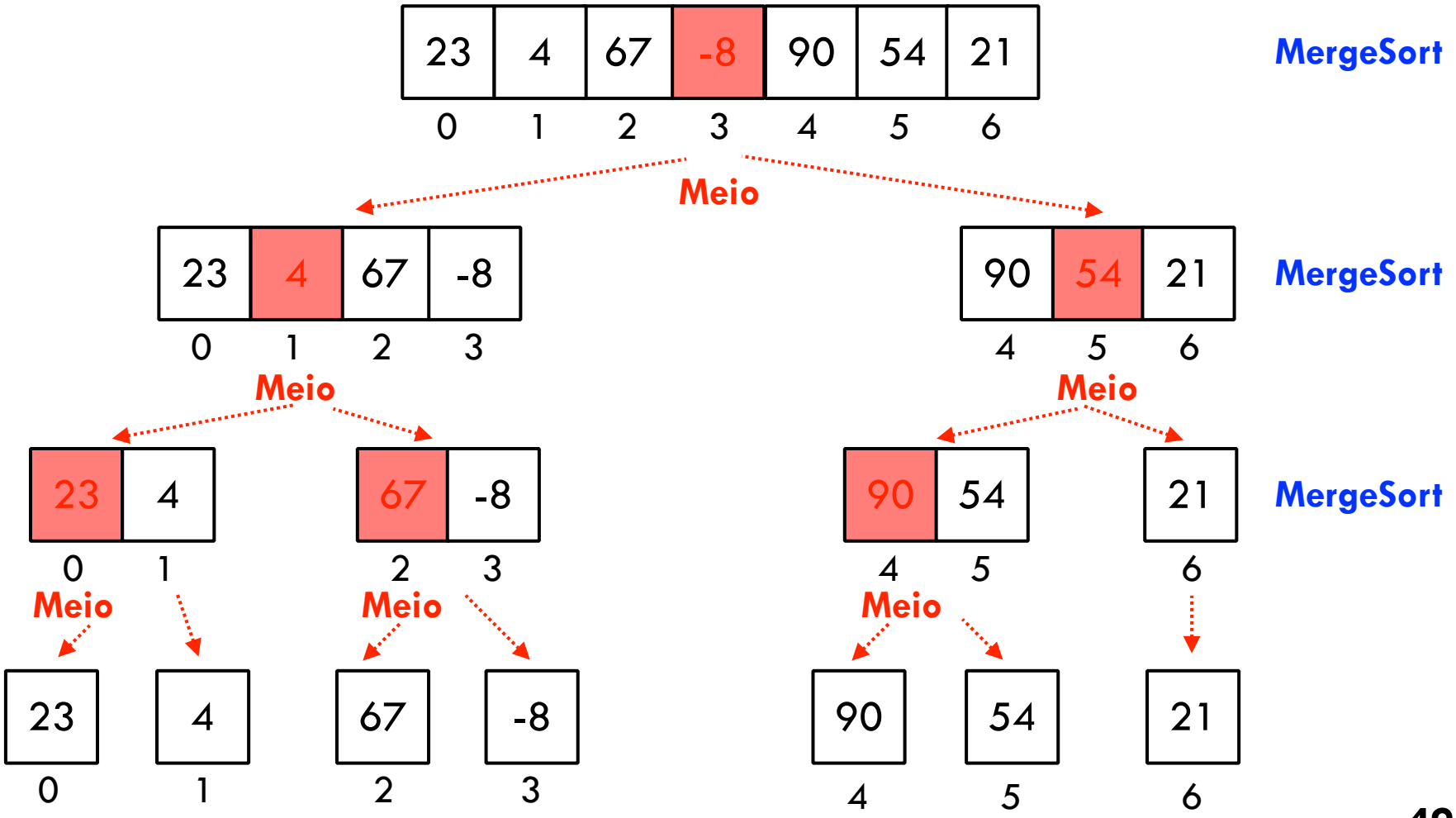
Exemplo


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

Exemplo

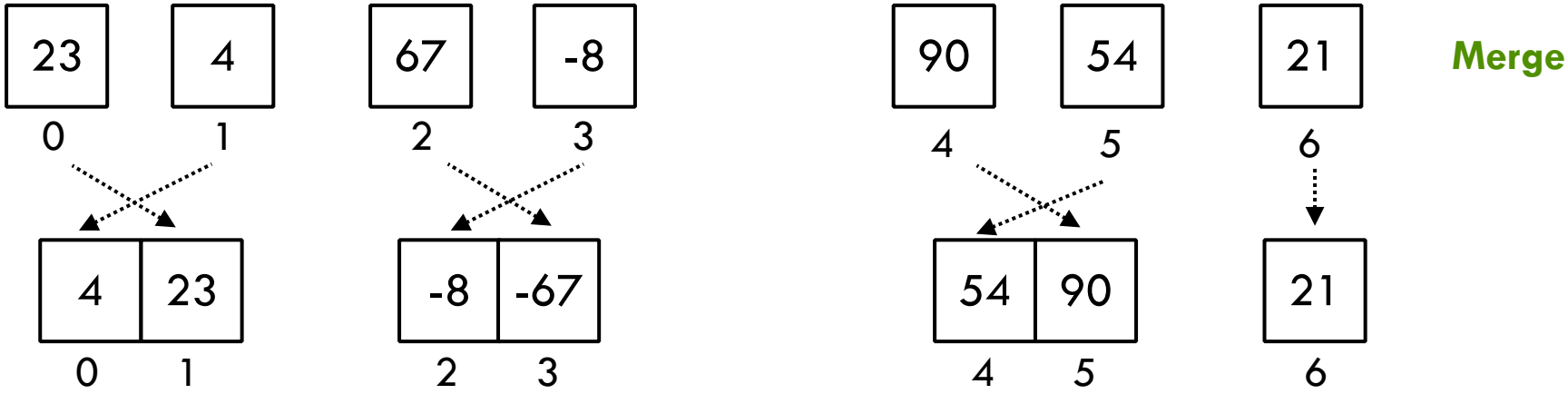

23 4 67 -8 90 54 21
0 1 2 3 4 5 6
MergeSort
Meio
23 4 67 -8
0 1 2 3
Meio
90 54 21
4 5 6
Meio
MergeSort
23 4
0 1
Meio
67 -8
2 3
Meio
90 54
4 5
Meio
21
6
MergeSort

Exemplo

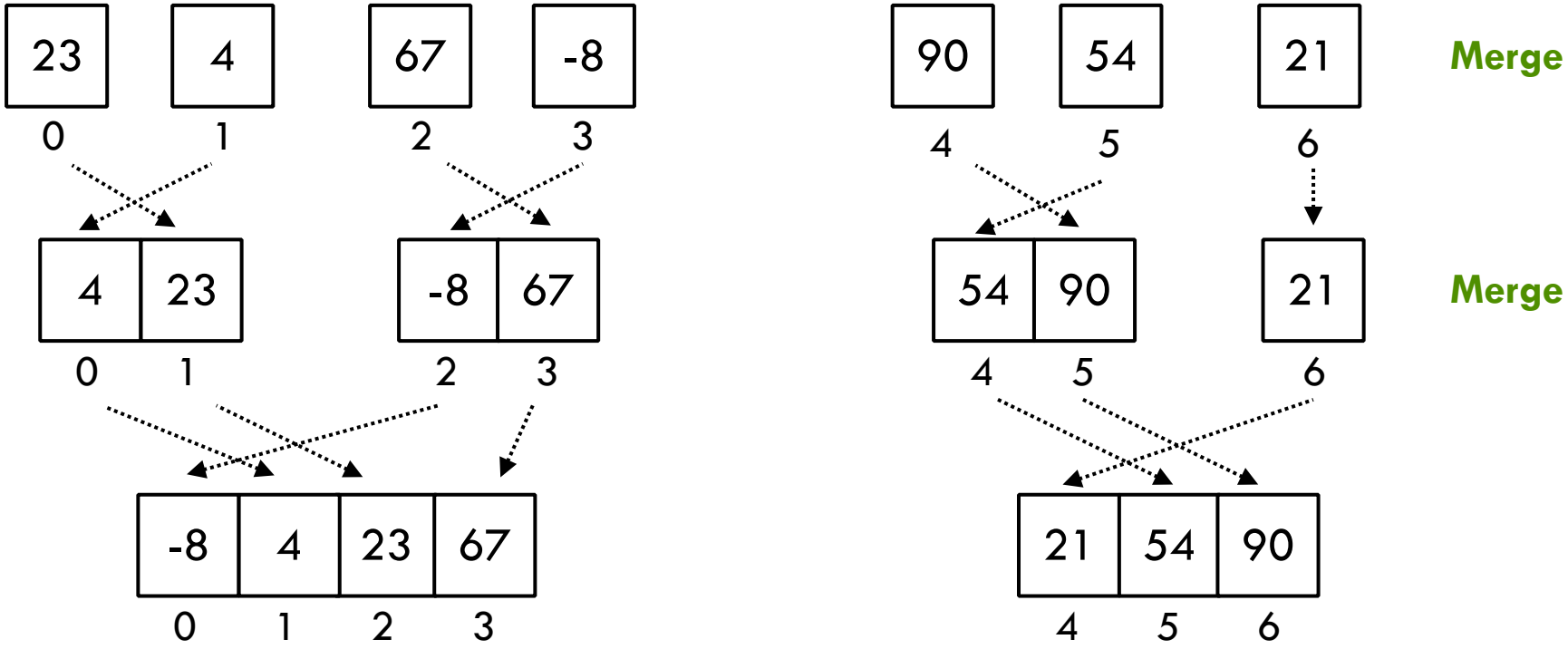

23 4 67 -8 90 54 21
0 1 2 3 4 5 6
MergeSort
Meio
23 4 67 -8
0 1 2 3
90 54 21
4 5 6
MergeSort
Meio
Meio
23 4
0 1
67 -8
2 3
90 54
4 5
21
6
MergeSort
Meio
Meio
Meio
23
0
4
1
67
2
-8
3
90
4
54
5
21
6

Exemplo

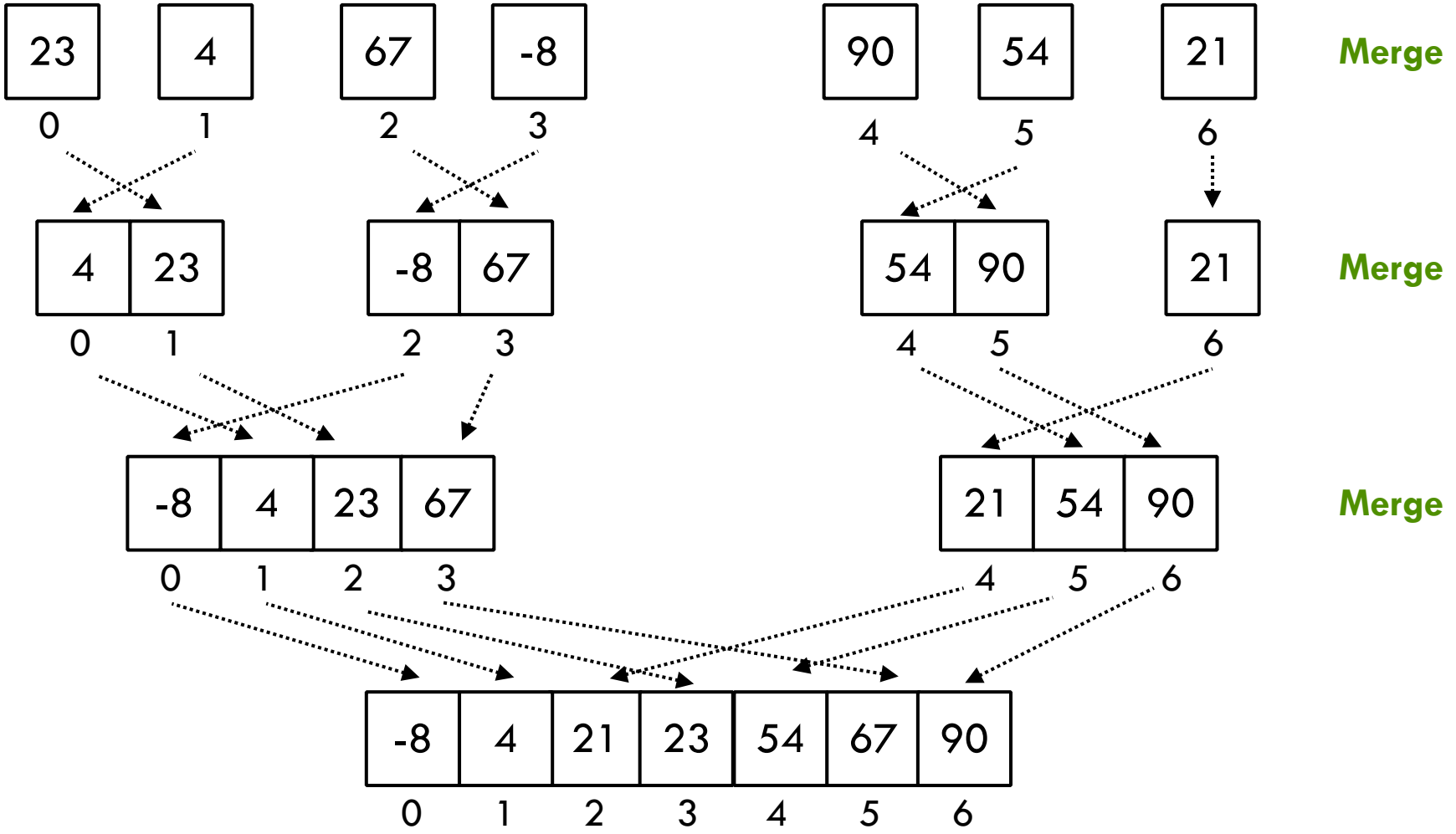
Exemplo



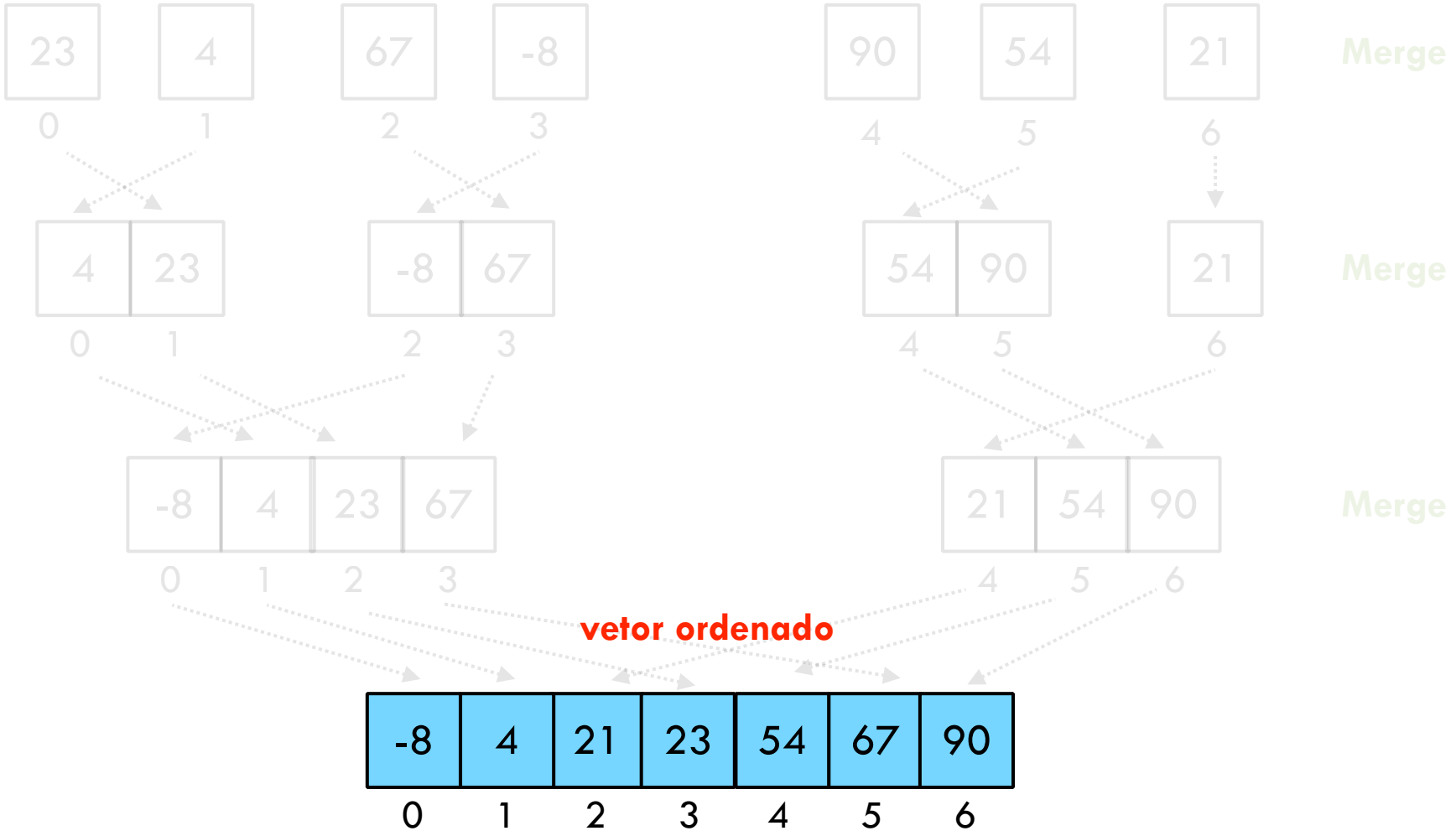
Exemplo


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

Exemplo


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

Exemplo


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

Merge Sort

Vantagens

* elegante e eficiente
* não altera a ordem dos dados (estável)
...

Desvantagens

* Recursivo
* Uso de memória - usa um vetor auxiliar durante a ordenação
* Pode ser custoso dependendo do tamanho do array

Roteiro

1. Introdução
2. Merge 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 **Merge Sort** em C considerando a seguinte assinatura de função:

```
/* Ordena o vetor usando Merge Sort
Parâmetros:
   v: vetor a ser ordenado
   n: número de elemento do vetor, tamanho do vetor

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

 void mergeSort(int *v, int n);
```

Exercícios

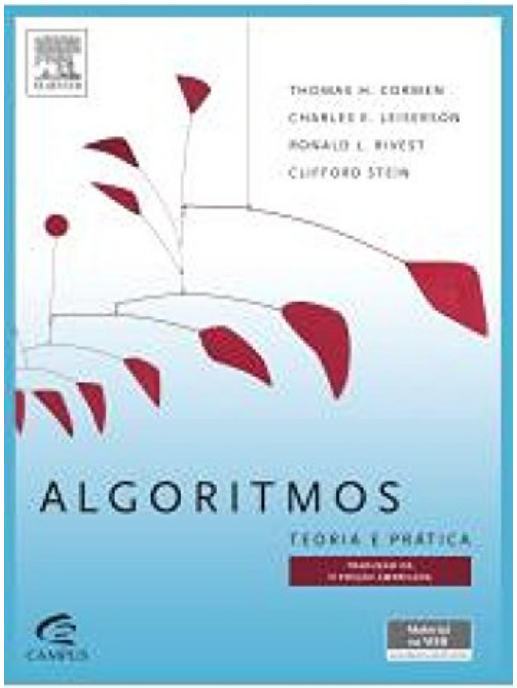
3) Adapte sua implementação do **mergeSort** para realizar tanto a ordenação crescente, como decrescente.

```
/* Ordena o vetor usando Merge Sort
Parâmetros:
   v: vetor a ser ordenado
   n: número de elemento do vetor, tamanho do vetor
   op: 1 para realizar ordenação crescente, 2 para ordenação
       decrescente

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

 void mergeSort(int *v, int n, int op);
```

Roteiro

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

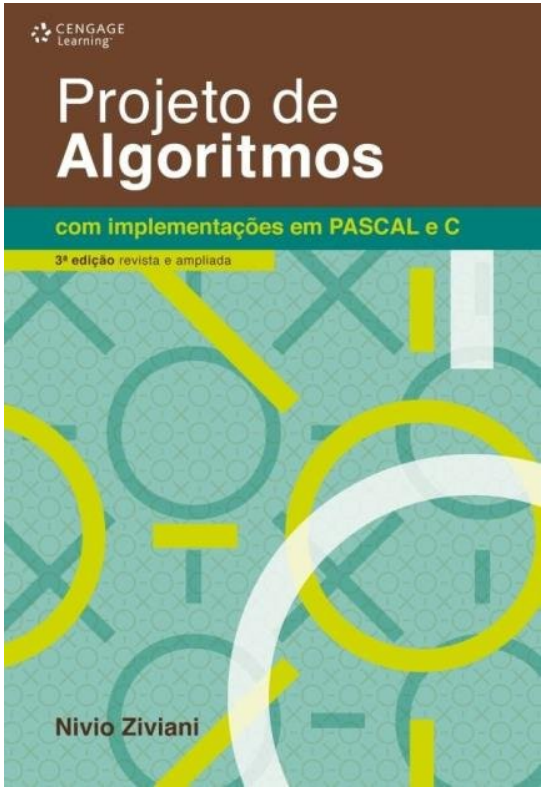
Referências sugeridas



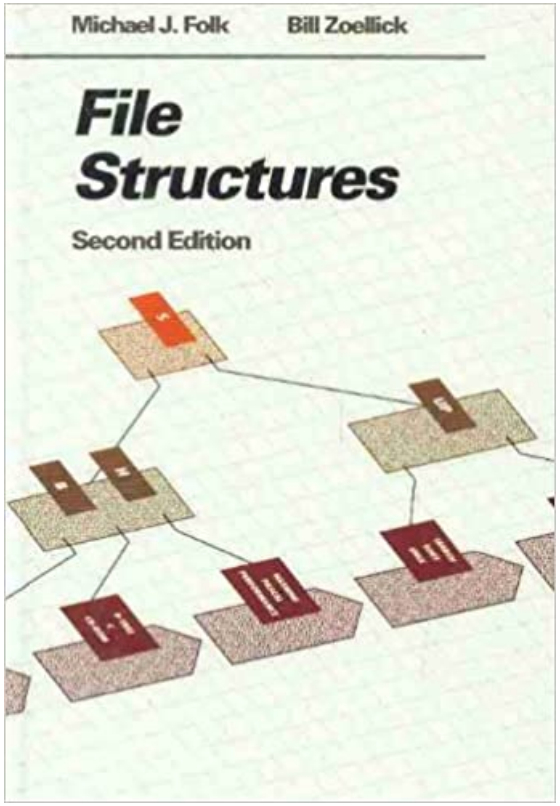
[Cormen et al, 2018]

[Drozdek, 2017]

Referências sugeridas



[Ziviani, 2010]



[Folk & Zoellick, 1992]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br