

EDCO4B

ESTRUTURAS DE DADOS 2

Aula 05 - Merge Sort

Prof. Rafael G. Mantovani
Prof. Luiz Fernando Carvalho

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

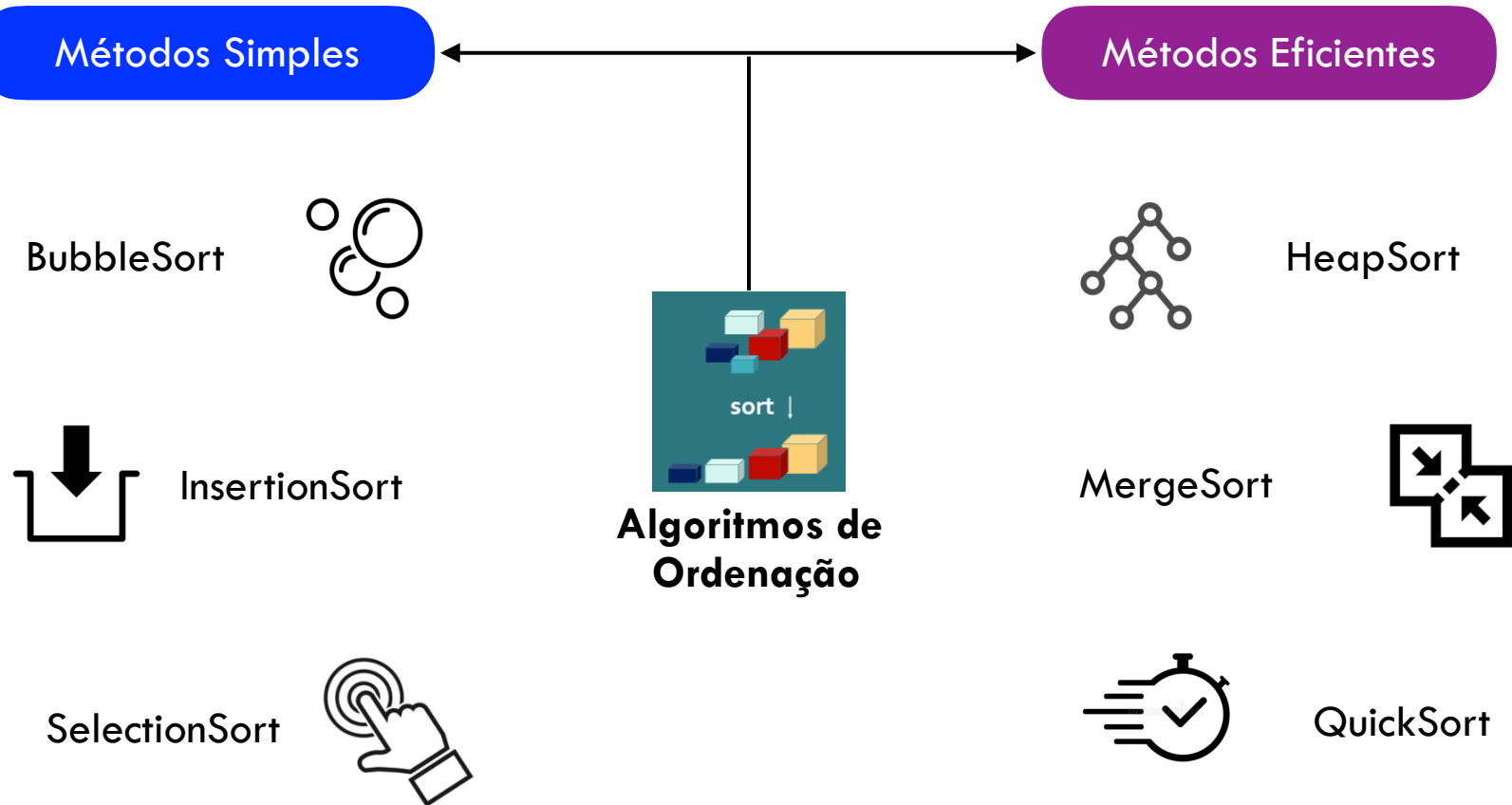
Métodos Simples

Métodos Eficientes

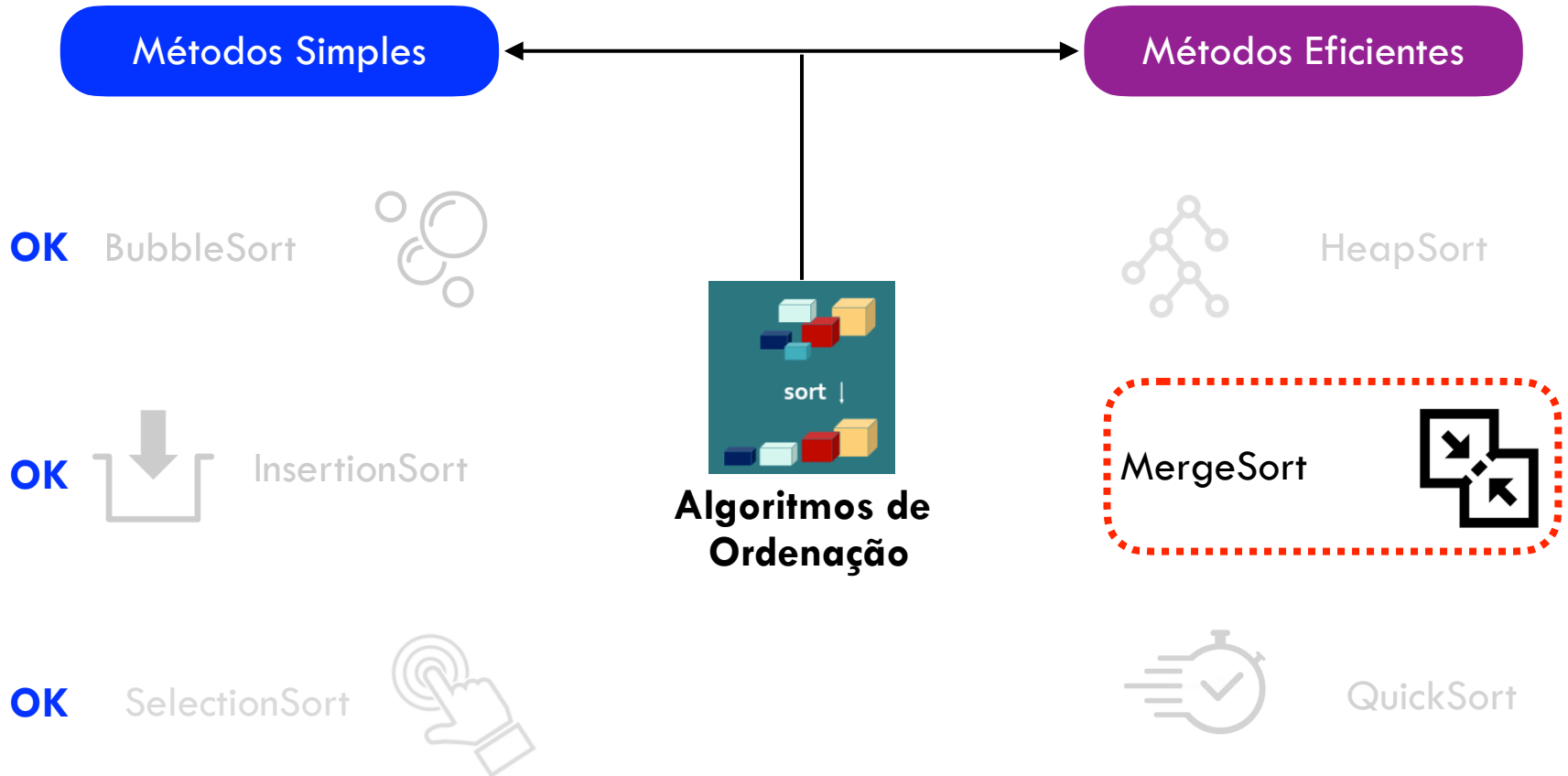


**Algoritmos de
Ordenação**

Introdução



Introdução



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

Nível 1

Lv 65



Lv 16



Lv 54



Lv 15



Lv 60



Meio

Lv 5



Lv 40



Lv 39



Lv 50



Merge Sort

Nível 1

Lv 65



Lv 16



Lv 54



Lv 15



Lv 60



Lv 5



Lv 40



Lv 39



Lv 50



Meio

Nível 2

Lv 65



Lv 16



Lv 54



Lv 15



Lv 60



Lv 5



Lv 40



Lv 39



Lv 50



Subproblema: esquerda

Subproblema: direita

Merge Sort

Nível 2

Lv 65



Lv 16



Lv 54



Meio

Lv 15



Lv 60



Merge Sort

Nível 2



Nível 3



Subproblema: esquerda

Subproblema: direita

Merge Sort

Nível 3

Lv 65



Lv 16



Lv 54



Meio

Merge Sort

Nível 3

Lv 65



Lv 16



Lv 54



Meio

Nível 4

Lv 65



Lv 16



Meio

Lv 54



Subproblema: esquerda

Subproblema: direita

Merge Sort

Nível 4

Lv 65



Meio

Lv 16



Lv 54



Merge Sort

Nível 4

Lv 65



Lv 16



Lv 54



Meio

Nível 5

Lv 65



Lv 16



Subproblema: esquerda

Subproblema: direita

Merge Sort

Nível 4

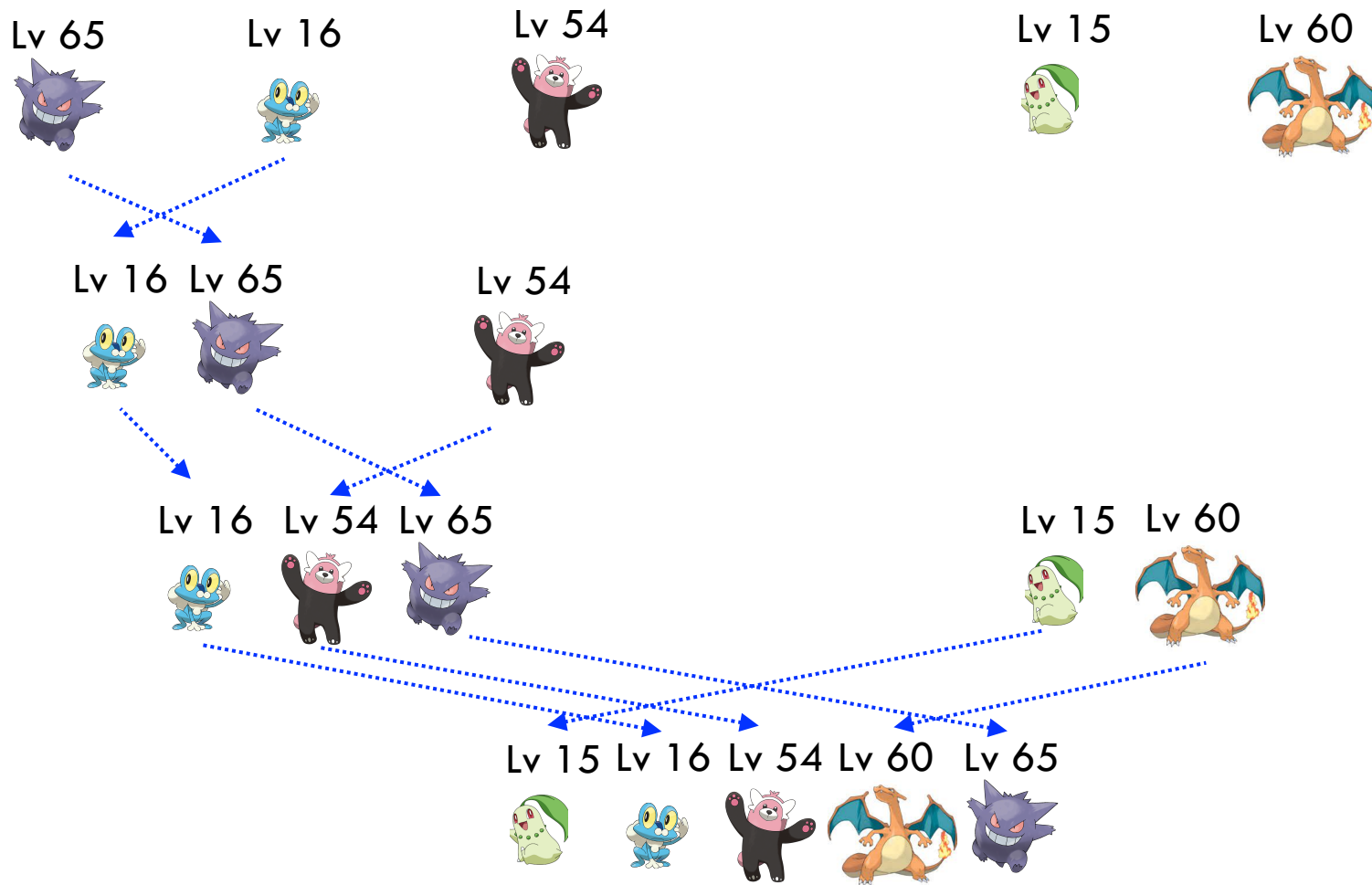


Condição de parada da recursão: **vetores unitários**
Próximo passo: **recombinar!**

Nível 5



Merge Sort



Merge Sort



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

4	23	67	-8	21
---	----	----	----	----

Merge Sort

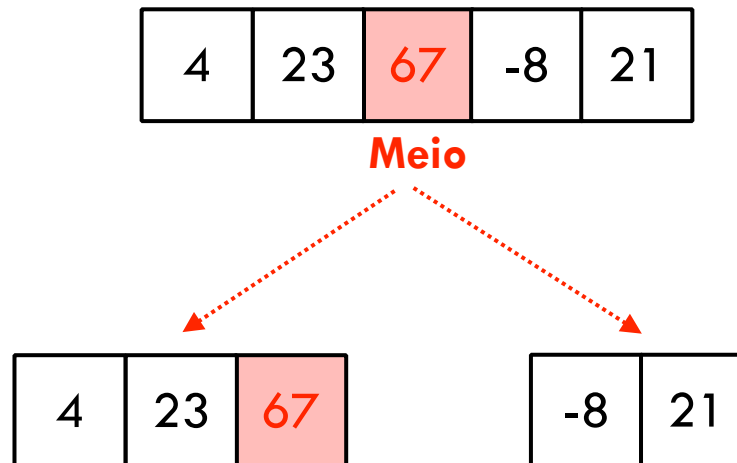
- Dividir

4	23	67	-8	21
---	----	----	----	----

Meio

Merge Sort

□ Dividir



Merge Sort

□ Recombinar

4	23	67
---	----	----

-8	21
----	----

Merge Sort

□ Recombinar

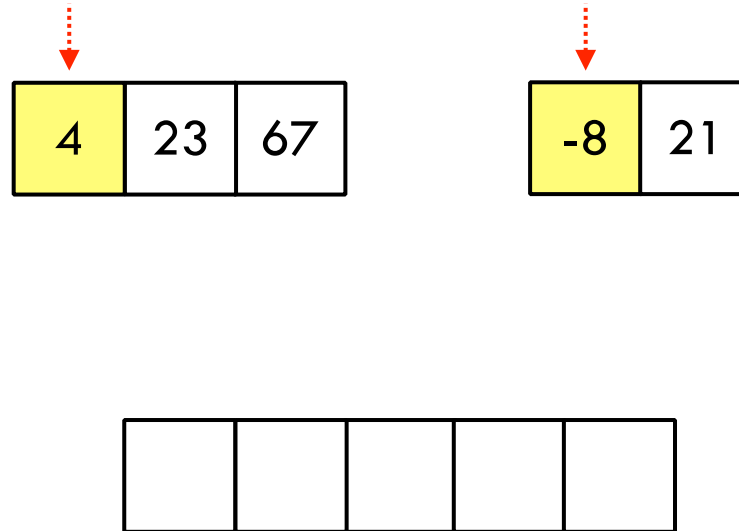
4	23	67
---	----	----

-8	21
----	----

--	--	--	--	--

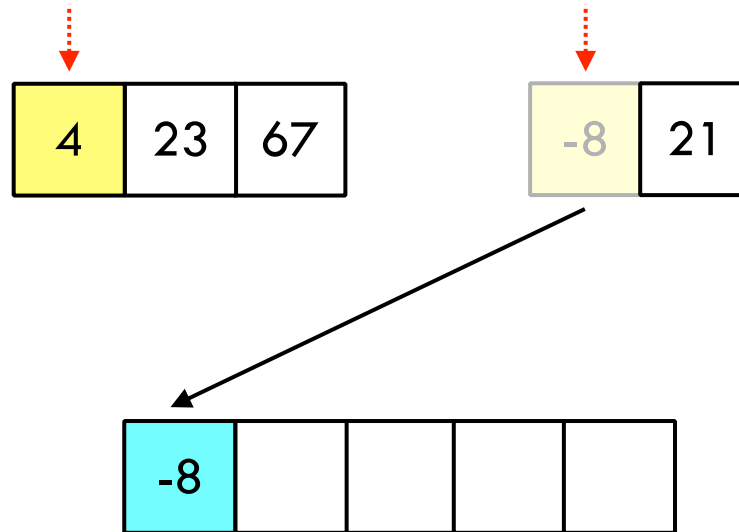
Merge Sort

□ Recombinar



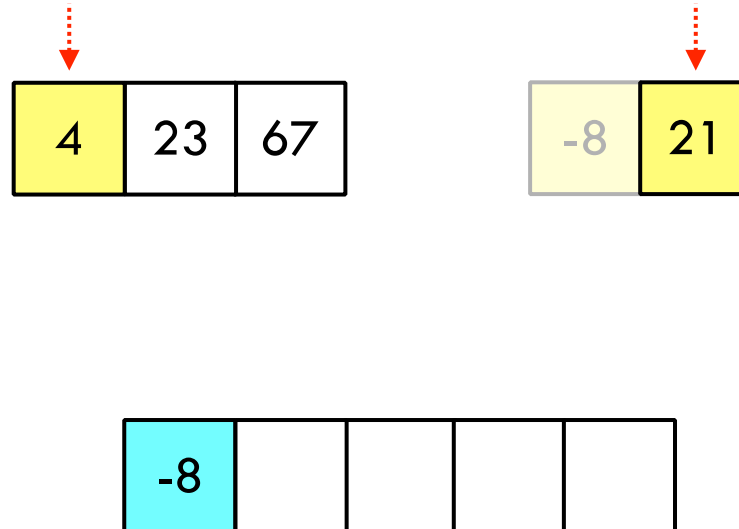
Merge Sort

□ Recombinar



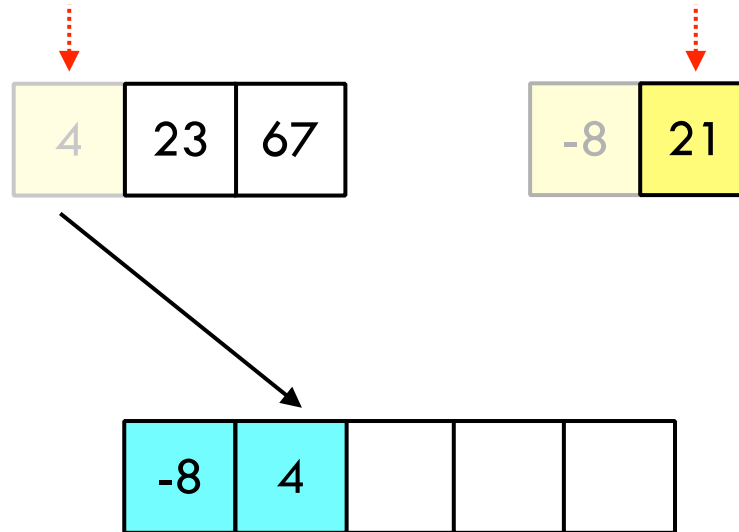
Merge Sort

□ Recombinar



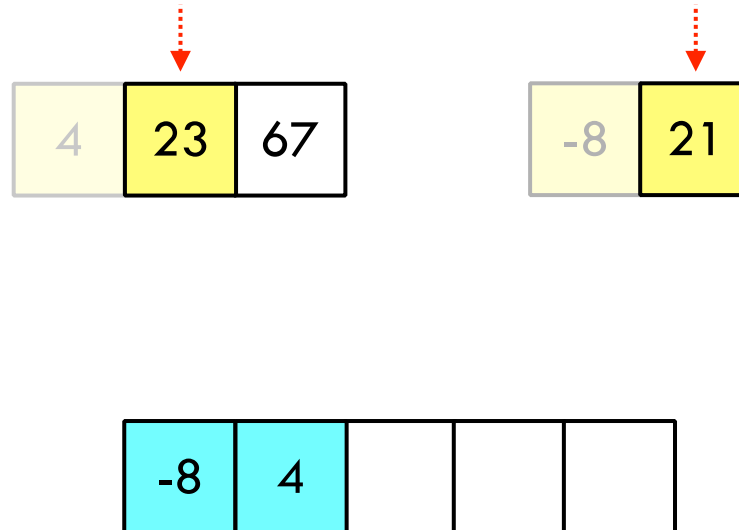
Merge Sort

□ Recombinar



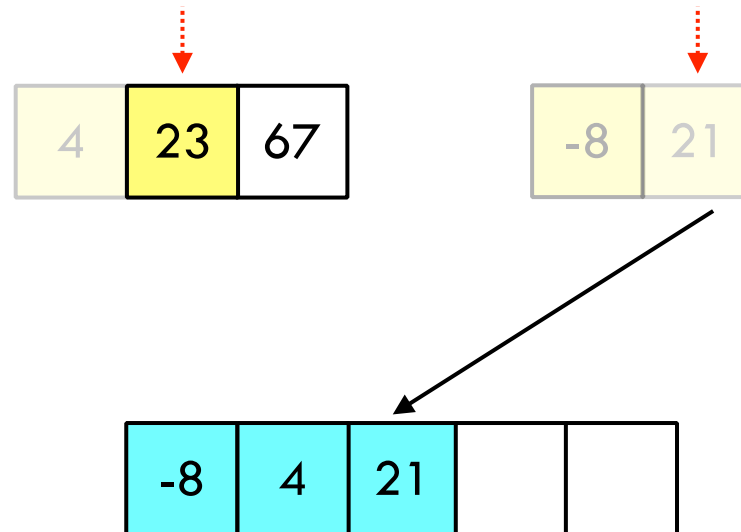
Merge Sort

□ Recombinar



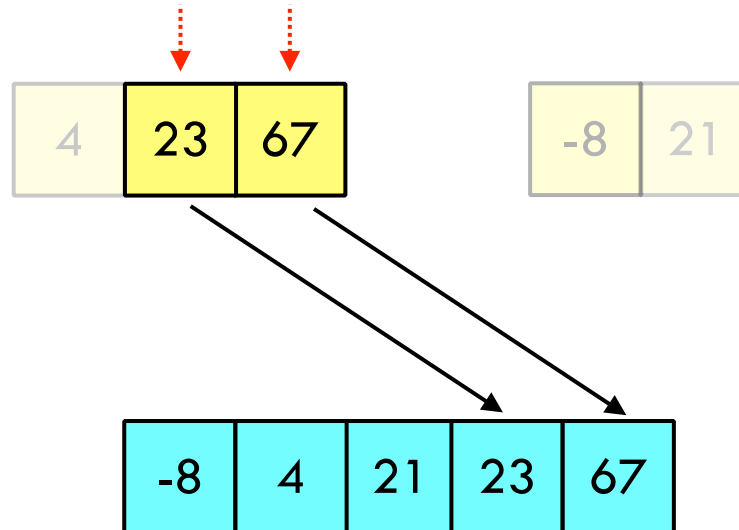
Merge Sort

□ Recombinar



Merge Sort

□ Recombinar



Merge Sort

□ Desempenho

- * **melhor caso:** $O(N \log N)$ // elementos já estão ordenados
- * **pior caso:** $O(N \log N)$ // elementos estão em ordem decrescente
- * **caso médio:** $O(N \log N)$

Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

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

Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

1. Se (**Início** < Fim) então : *//critério para controle de recursão*
2. **meio** = **round**((**Início** + Fim) /2)

Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

1. Se (**Início** < Fim) então : *//critério para controle de recursão*
2. **meio** = **round**((**Início** + Fim) /2)
 // chamada recursiva para os subproblemas
3. **MergeSort**(vetor, **Início**, **meio**) *//subproblema da esquerda*
4. **MergeSort**(vetor, **meio** + 1, Fim) *//subproblema da direita*

Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

1. Se (**Início** < Fim) então : *//critério para controle de recursão*
2. **meio** = **round**((**Início** + Fim) /2)
 // chamada recursiva para os subproblemas
3. **MergeSort**(vetor, **Início**, **meio**) *//subproblema da esquerda*
4. **MergeSort**(vetor, **meio** + 1, Fim) *//subproblema da direita*

 // função auxiliar para recombina os subproblemas
5. **Merge**(vetor, **Início**, **meio**, Fim)

Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

1. **Se** (**Início** < Fim) então : *//critério para controle de recursão*
2. **meio** = **round**((**Início** + Fim) /2)
 // chamada recursiva para os subproblemas
3. **MergeSort**(vetor, **Início**, **meio**) *//subproblema da esquerda*
4. **MergeSort**(vetor, **meio** + 1, Fim) *//subproblema da direita*

 // função auxiliar para recombina os subproblemas
5. **Merge**(vetor, **Início**, **meio**, Fim)

 *// Ao final da última chamada da função principal, o vetor “vetor”
 está ordenado*

Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

1. Se (**Início** < Fim) então :
2. **meio** = round((**Início** + Fim) /2)
3. **MergeSort**(vetor, **Início**, **meio**)
4. **MergeSort**(vetor, **meio** + 1, Fim)
5. **Merge**(vetor, **Início**, **meio**, Fim)

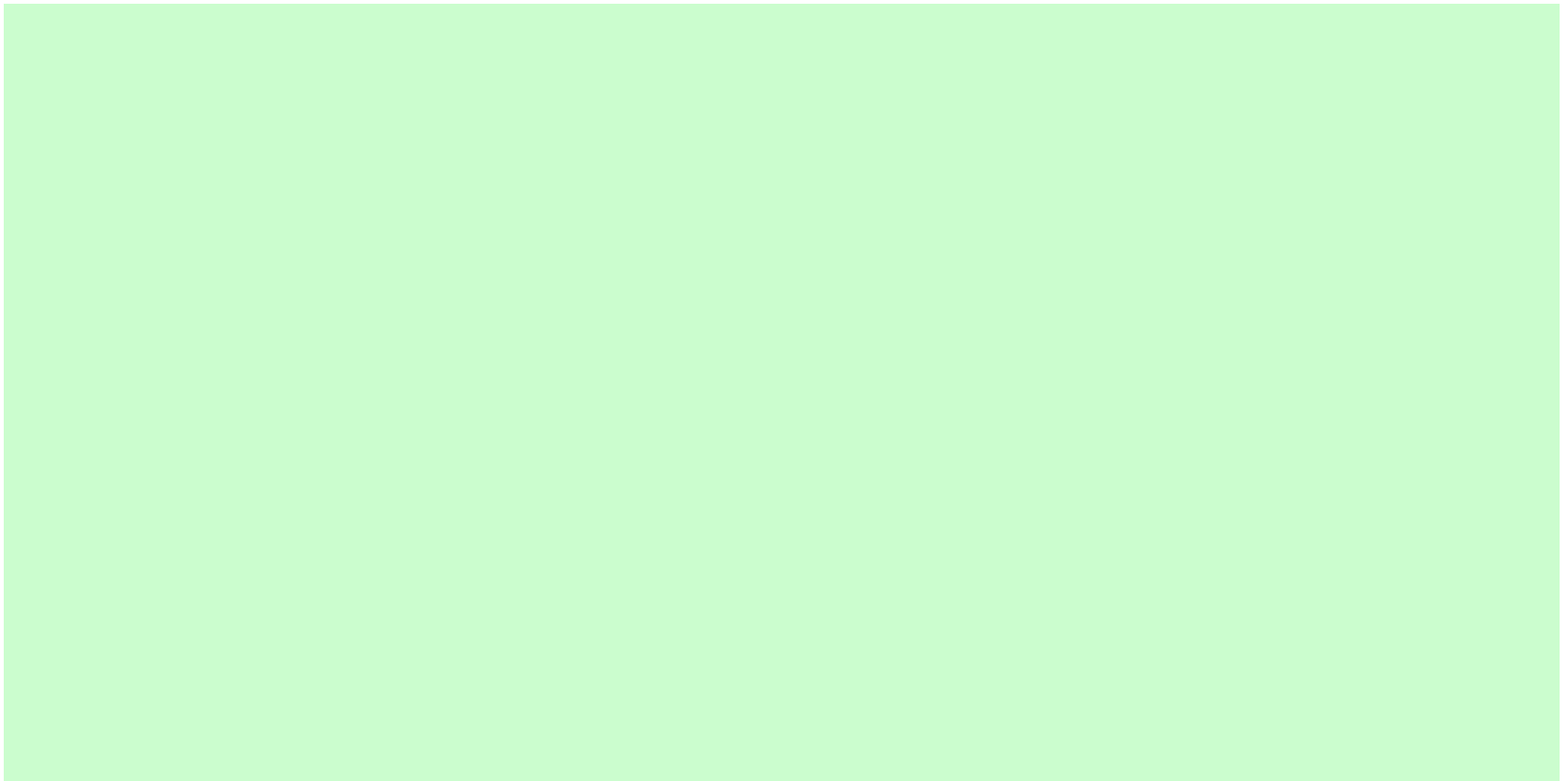
Pseudocódigo: Merge Sort

MergeSort (vetor, **Início**, Fim):

1. Se (**Início** < Fim) então :
2. **meio** = round((**Início** + Fim) /2)
3. **MergeSort**(vetor, **Início**, **meio**)
4. **MergeSort**(vetor, **meio** + 1, Fim)
5. **Merge**(vetor, **Início**, **meio**, Fim)

Função Auxiliar: Merge

Merge (vetor, **Início**, **meio**, Fim):



Função Auxiliar: Merge

Merge (vetor, **Início**, **meio**, Fim):

// iniciar variáveis locais

1. Alocar dinamicamente um **vetor auxiliar**
2. **P1** = **Início**
3. **P2** = **meio** + 1

Função Auxiliar: Merge

Merge (vetor, **Início**, **meio**, Fim):

// iniciar variáveis locais

1. Alocar dinamicamente um **vetor auxiliar**
2. **P1** = **Início**
3. **P2** = **meio** + 1

// recombinar os subproblemas parciais no vetor auxiliar

4. Enquanto (**P1** <= **meio** e **P2** <= Fim), faça:

|

Função Auxiliar: Merge

Merge (vetor, **Início**, **meio**, Fim):

// iniciar variáveis locais

1. Alocar dinamicamente um **vetor auxiliar**
2. **P1** = **Início**
3. **P2** = **meio** + 1

// recombinar os subproblemas parciais no vetor auxiliar

4. **Enquanto** (**P1** <= **meio** e **P2** <= Fim), **faça**:
5. Copiar o menor valor entre **vetor**[**P1**] e **vetor**[**P2**] para a próxima posição disponível no **vetor auxiliar**
6. Incrementa a quantidade de elementos no **vetor auxiliar** e atualiza a próxima posição disponível

Função Auxiliar: Merge

// Ao final do laço, sobrarão elementos apenas no subproblema da direita ou no subproblema da esquerda

Função Auxiliar: Merge

// Ao final do laço, sobrarão elementos apenas no subproblema da direita ou no subproblema da esquerda

7. Se (**P1** == **meio**):

//sobraram elementos no subproblema da direita

Função Auxiliar: Merge

// Ao final do laço, sobrarão elementos apenas no subproblema da direita ou no subproblema da esquerda

7. Se (**P1** == **meio**):

//sobraram elementos no subproblema da direita

8. Copiar o que sobrou a partir de **P2** para o **vetor auxiliar**

Função Auxiliar: Merge

// Ao final do laço, sobrarão elementos apenas no subproblema da direita ou no subproblema da esquerda

7. Se (**P1** == **meio**):

//sobraram elementos no subproblema da direita

8. Copiar o que sobrou a partir de **P2** para o **vetor auxiliar**

9. Senão:

//sobraram elementos no subproblema da esquerda

Função Auxiliar: Merge

// Ao final do laço, sobrarão elementos apenas no subproblema da direita ou no subproblema da esquerda

7. Se (**P1** == **meio**):

//sobraram elementos no subproblema da direita

8. Copiar o que sobrou a partir de **P2** para o **vetor auxiliar**

9. Senão:

//sobraram elementos no subproblema da esquerda

10. Copiar o que sobrou a partir de **P1** para o **vetor auxiliar**

Função Auxiliar: Merge

// Ao final do laço, sobrarão elementos apenas no subproblema da direita ou no subproblema da esquerda

7. Se (**P1** == **meio**):

//sobraram elementos no subproblema da direita

8. Copiar o que sobrou a partir de **P2** para o **vetor auxiliar**

9. Senão:

//sobraram elementos no subproblema da esquerda

10. Copiar o que sobrou a partir de **P1** para o **vetor auxiliar**

11. Copiar o **vetor auxiliar** para o **vetor** original

Roteiro



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

Exemplo

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

vetor não ordenado

Exemplo

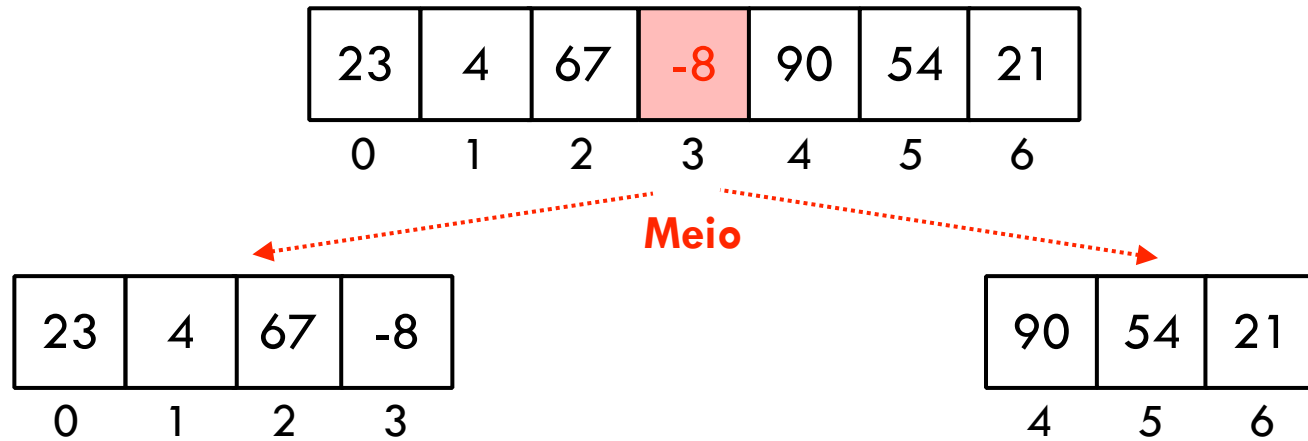
23	4	67	-8	90	54	21
0	1	2	3	4	5	6

Exemplo

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

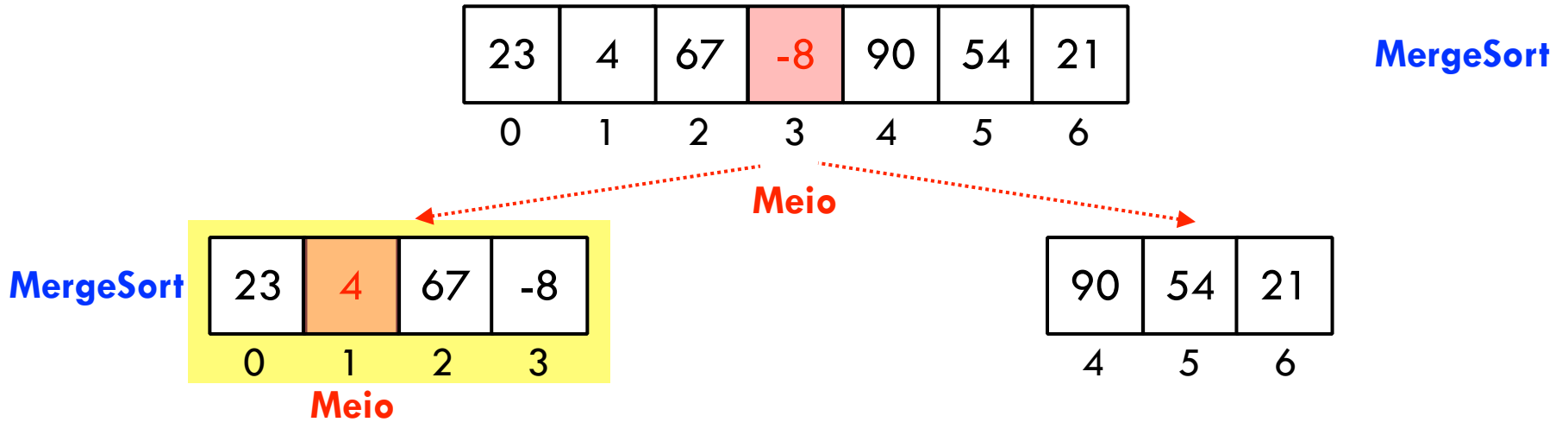
Meio

Exemplo



MergeSort

Exemplo



Exemplo

MergeSort

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

Meio

MergeSort

23	4	67	-8
0	1	2	3

90	54	21
4	5	6

Meio

23	4
0	1

67	-8
2	3

Exemplo

MergeSort

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

Meio

MergeSort

23	4	67	-8
0	1	2	3

90	54	21
4	5	6

Meio

23	4
0	1

67	-8
2	3

Meio

Exemplo

MergeSort

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

Meio

MergeSort

23	4	67	-8
0	1	2	3

90	54	21
4	5	6

Meio

23	4
0	1

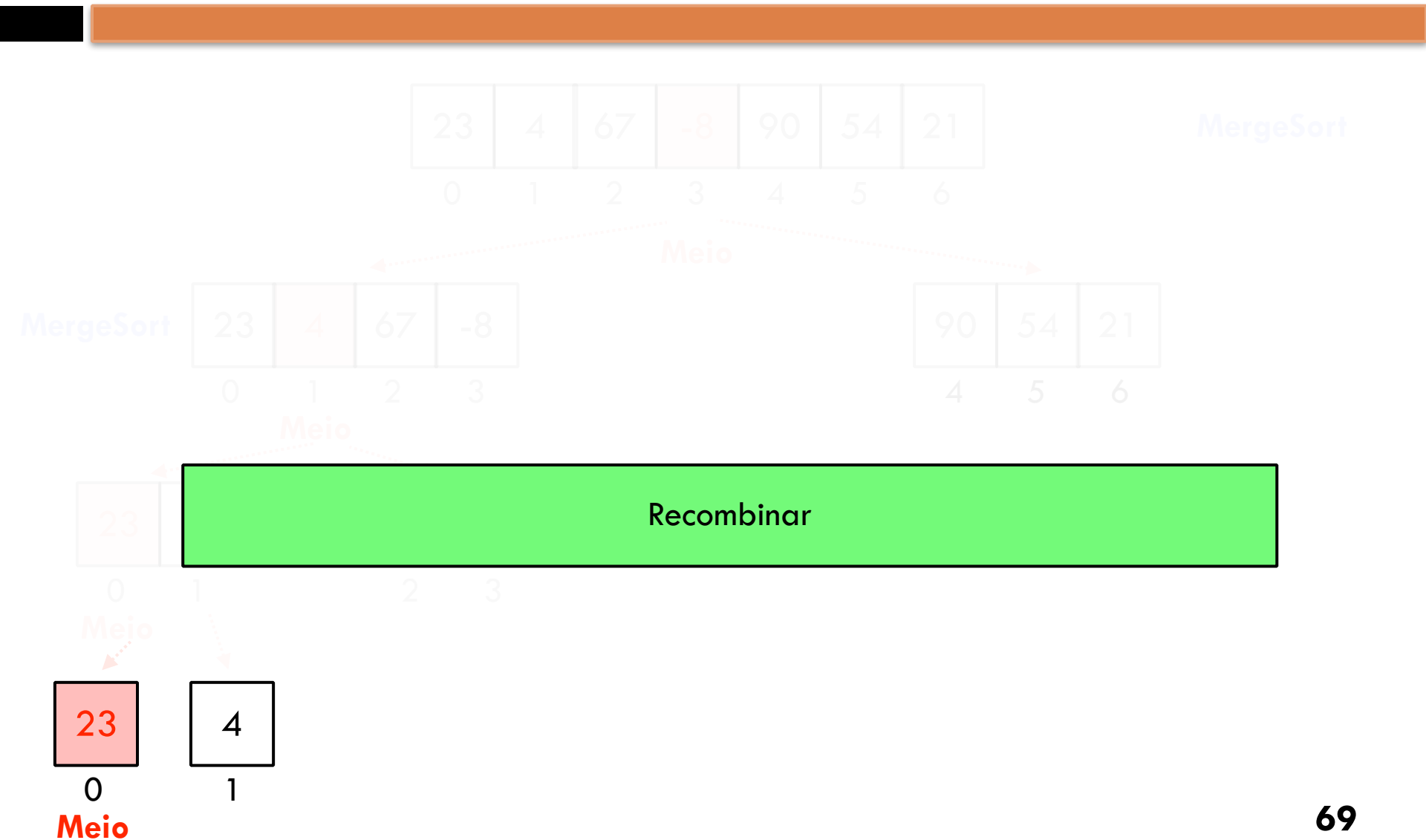
67	-8
2	3

Meio

23	4
0	1

Meio

Exemplo



Exemplo

MergeSort

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

Meio

MergeSort

23	4	67	-8
0	1	2	3

90	54	21
4	5	6

Meio

4	23
0	1

67	-8
2	3

Exemplo

MergeSort

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

Meio

MergeSort

23	4	67	-8
0	1	2	3

90	54	21
4	5	6

Meio

4	23
0	1

67	-8
2	3

Meio

Exemplo

MergeSort

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

Meio

MergeSort

23	4	67	-8
0	1	2	3

90	54	21
4	5	6

Meio

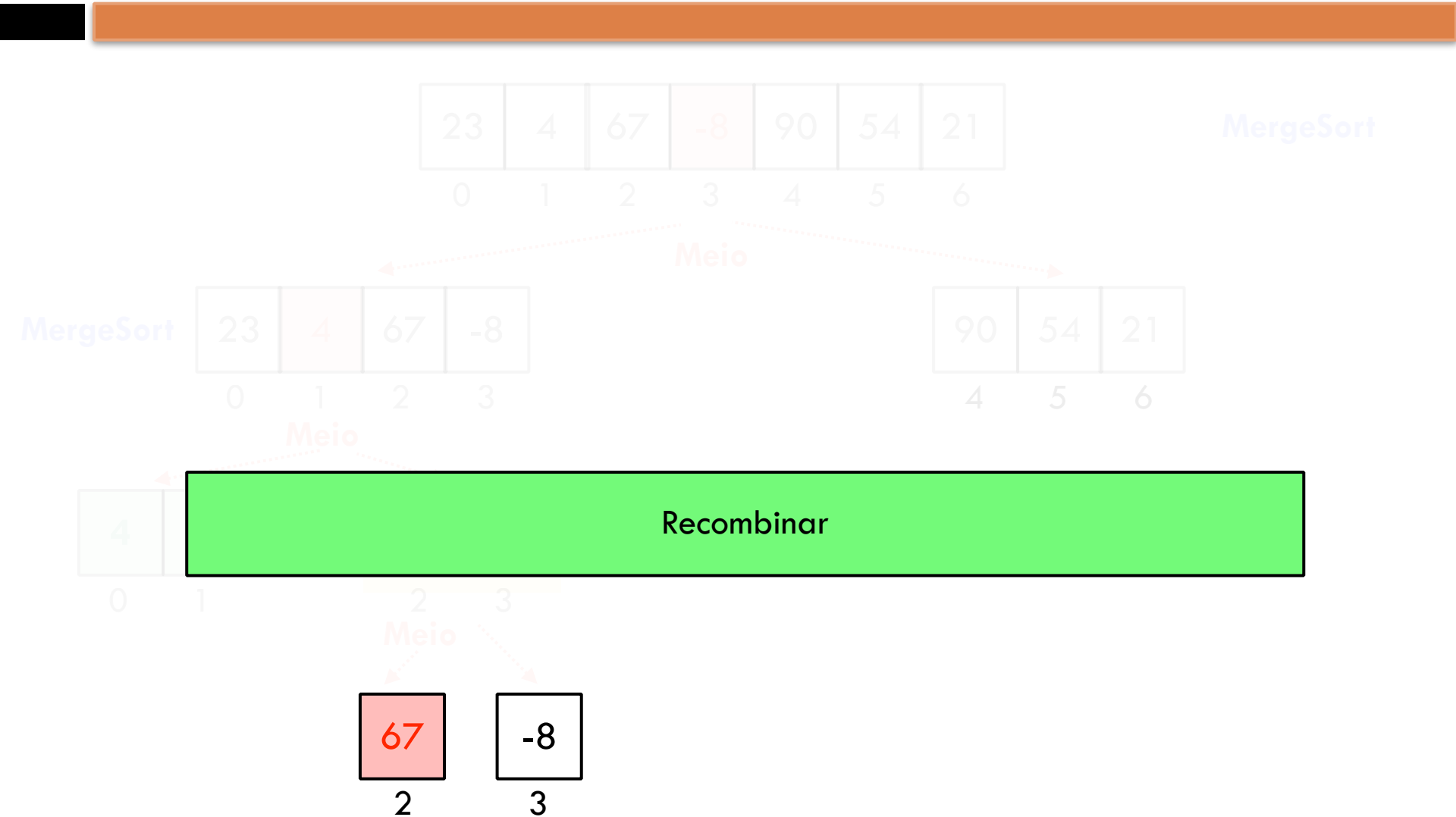
4	23
0	1

67	-8
2	3

Meio

67	-8
2	3

Exemplo



Exemplo

MergeSort

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

Meio

MergeSort

23	4	67	-8
0	1	2	3

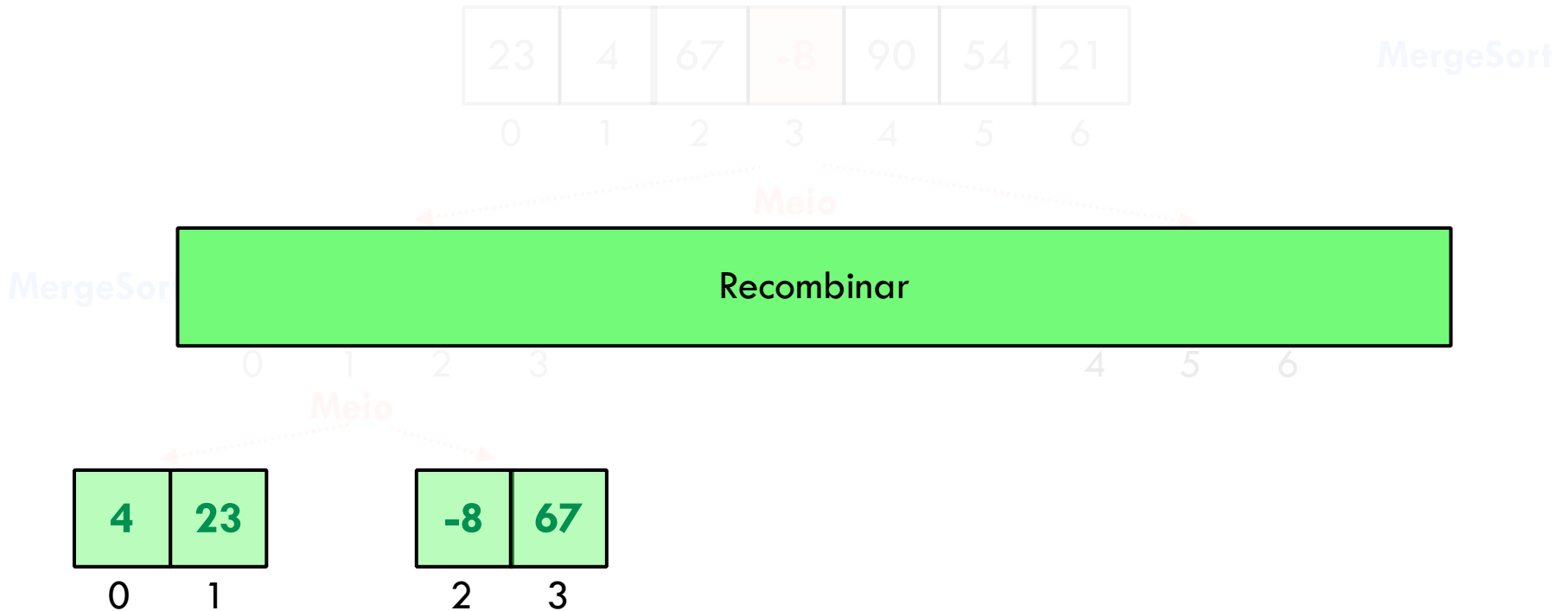
90	54	21
4	5	6

Meio

4	23
0	1

-8	67
2	3

Exemplo

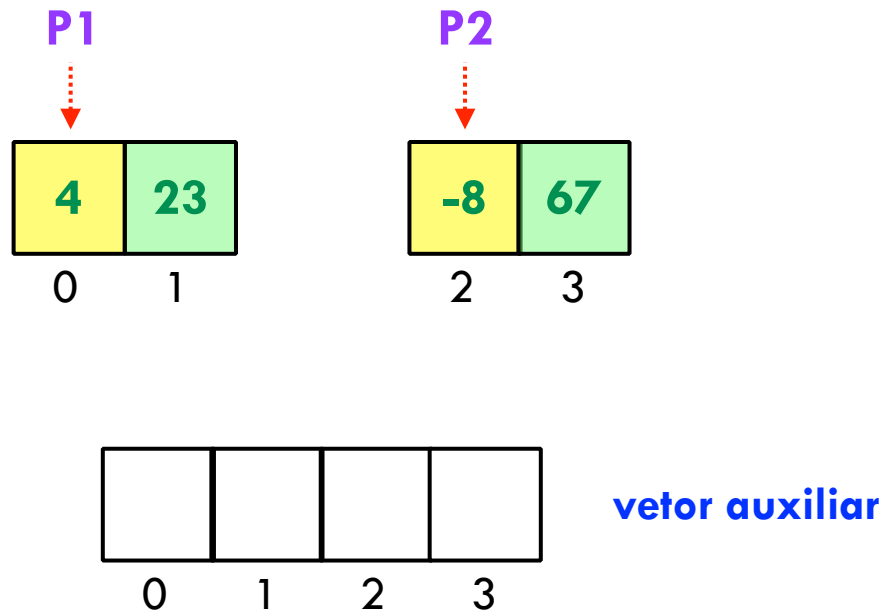


Exemplo

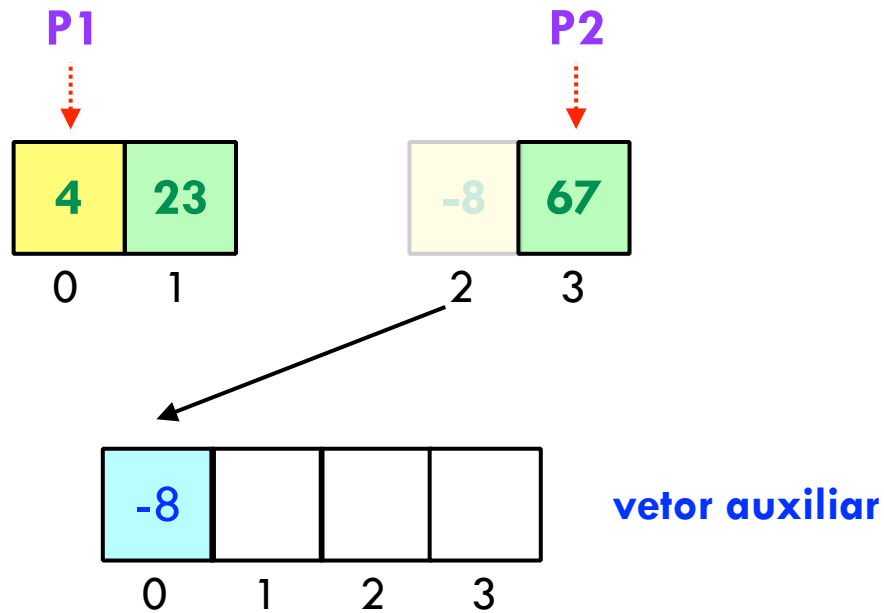
4	23
0	1

-8	67
2	3

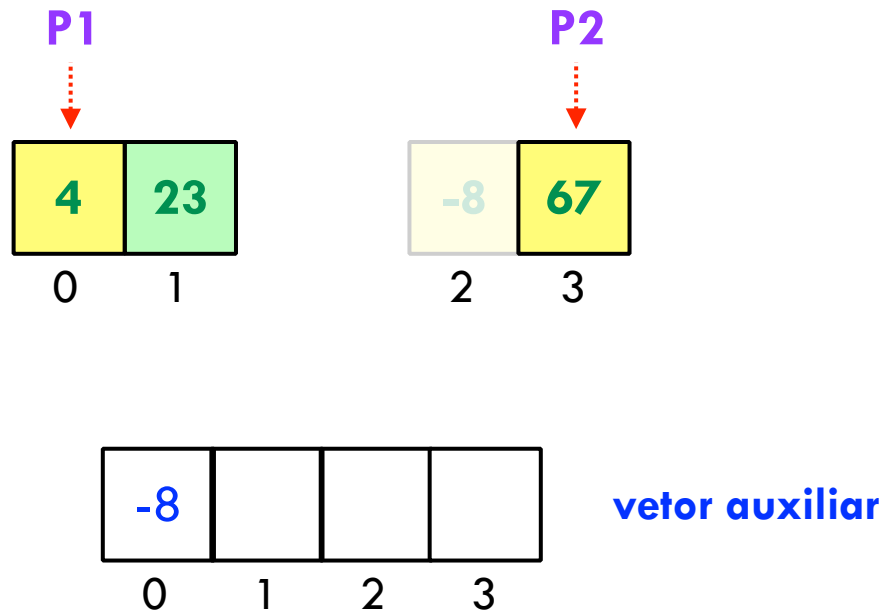
Exemplo



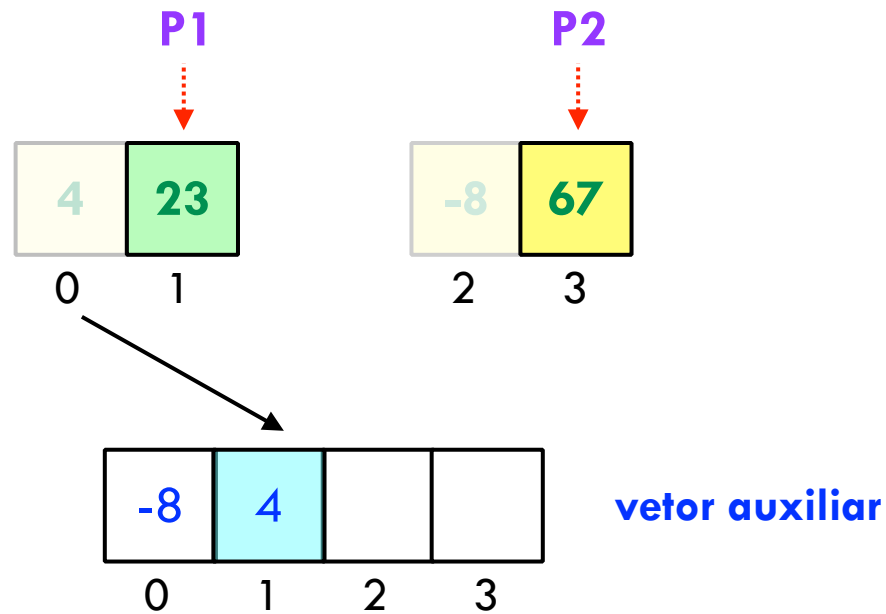
Exemplo



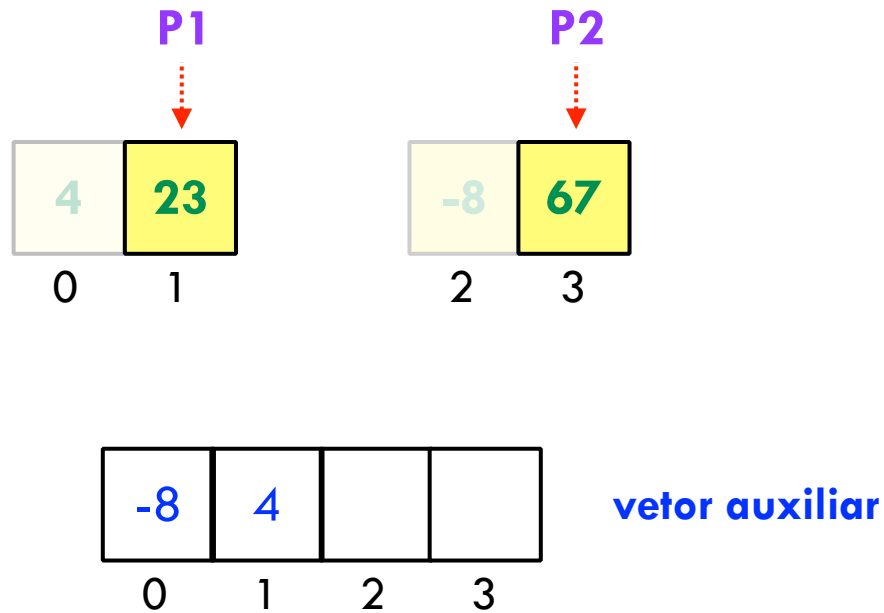
Exemplo



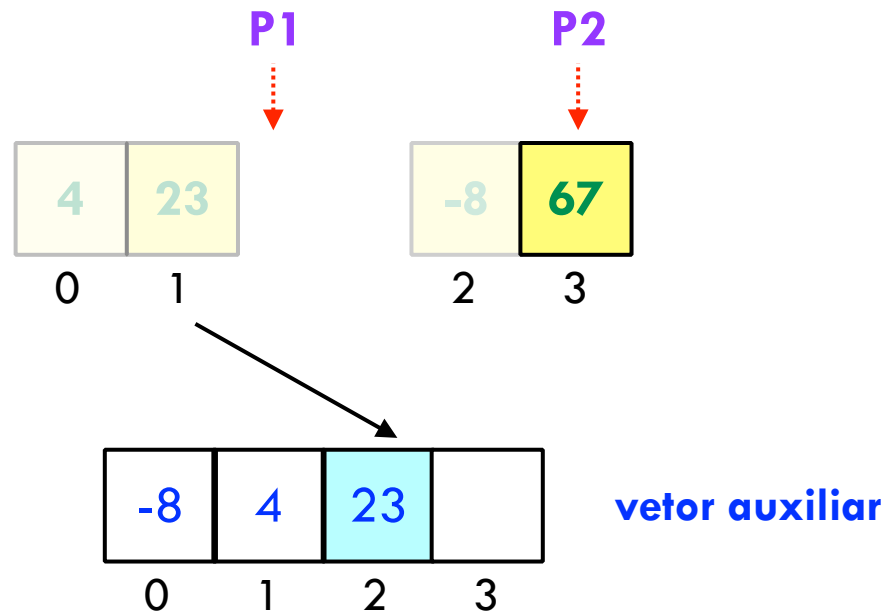
Exemplo



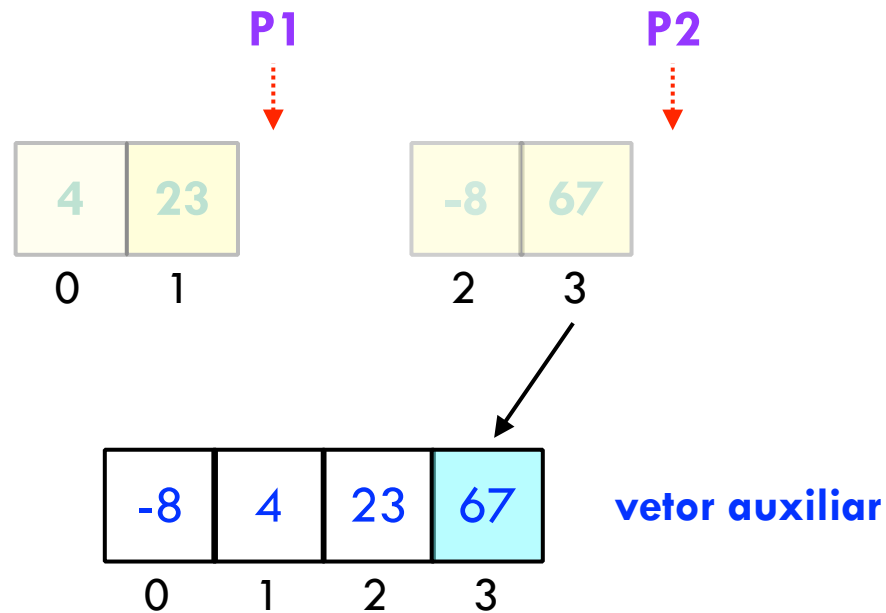
Exemplo



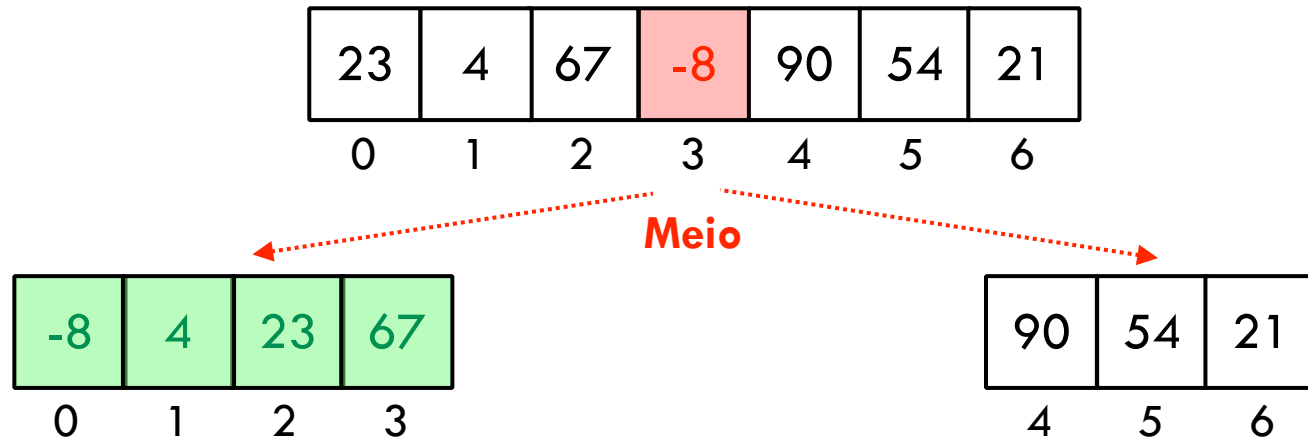
Exemplo



Exemplo

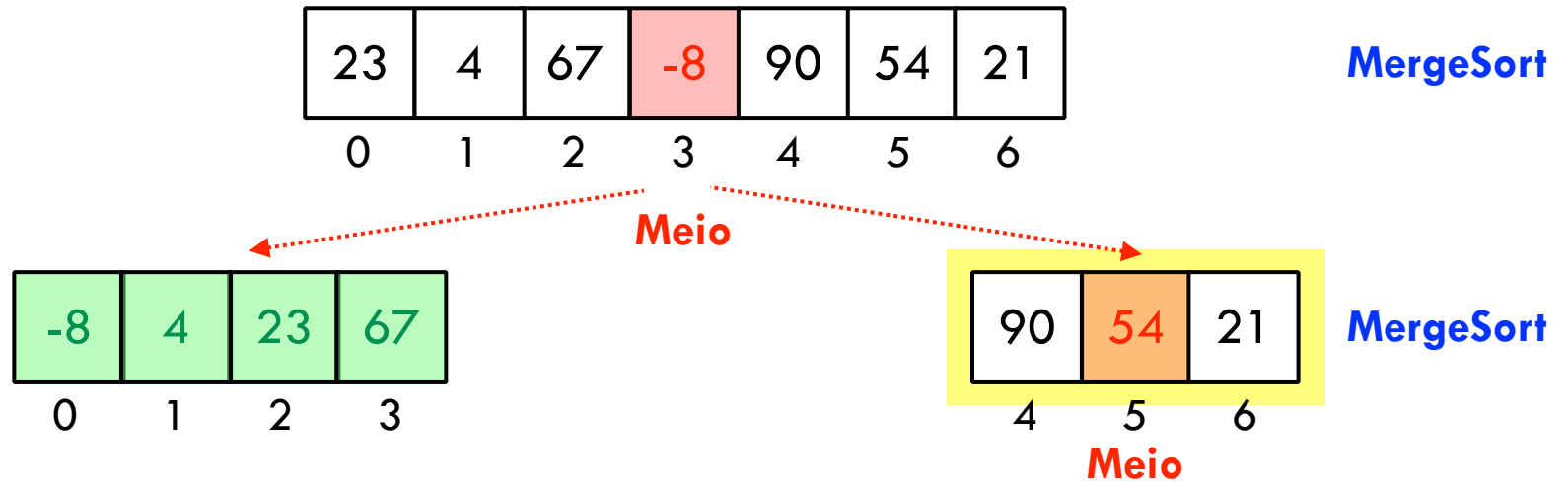


Exemplo

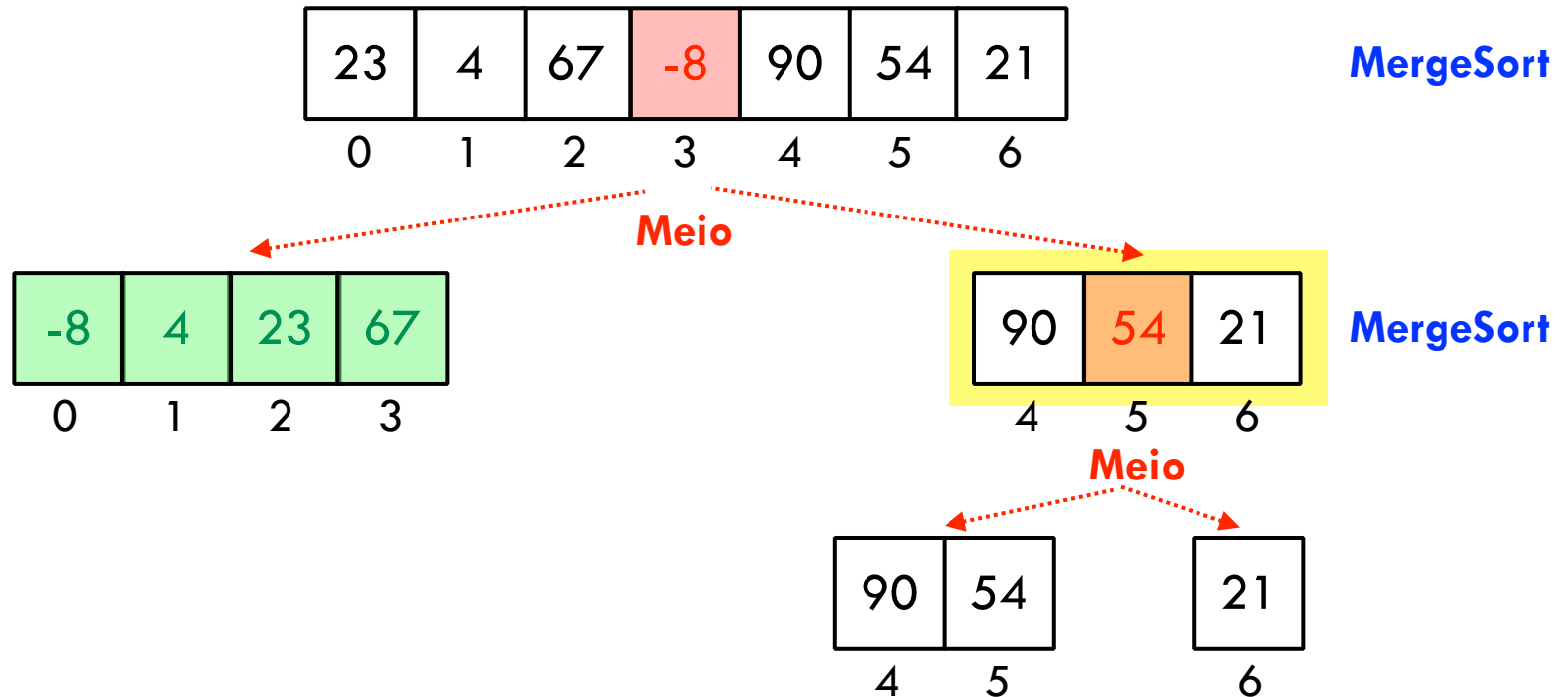


MergeSort

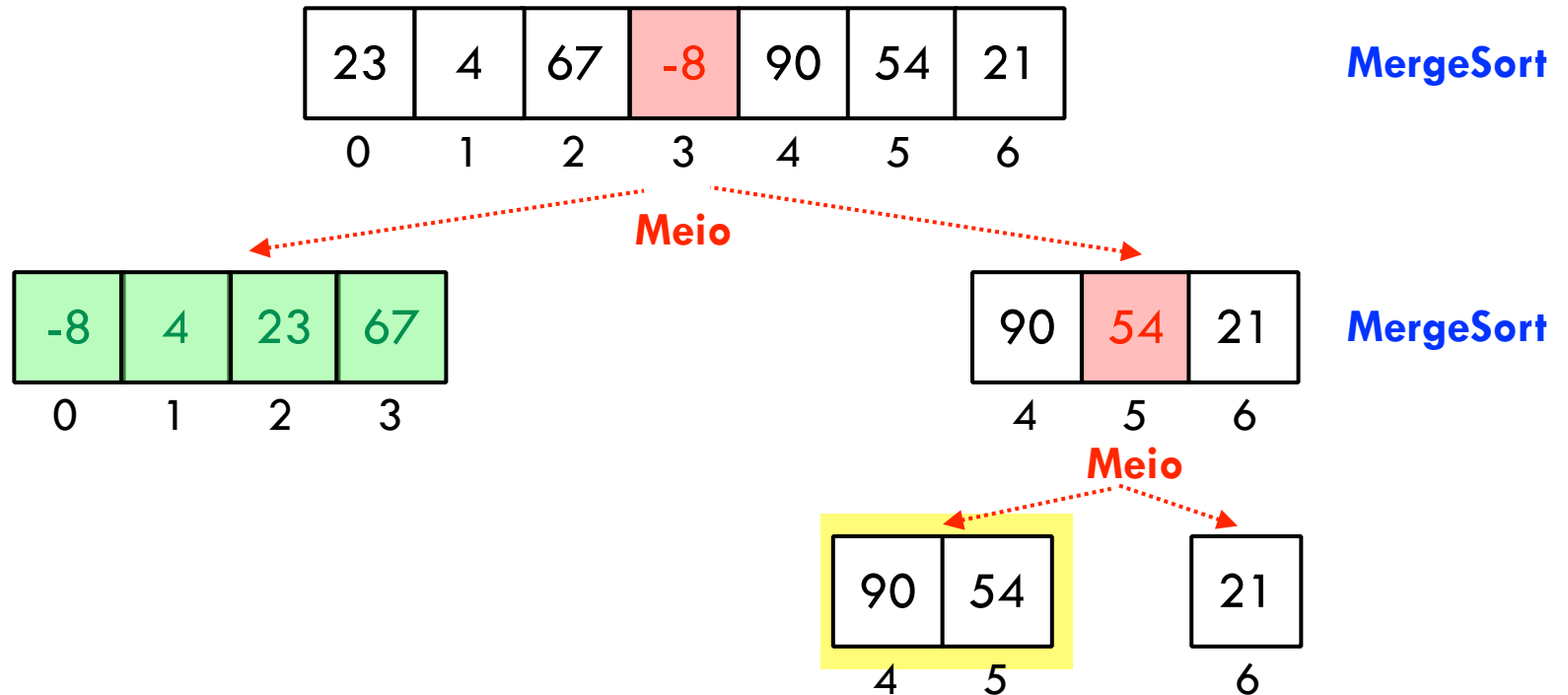
Exemplo



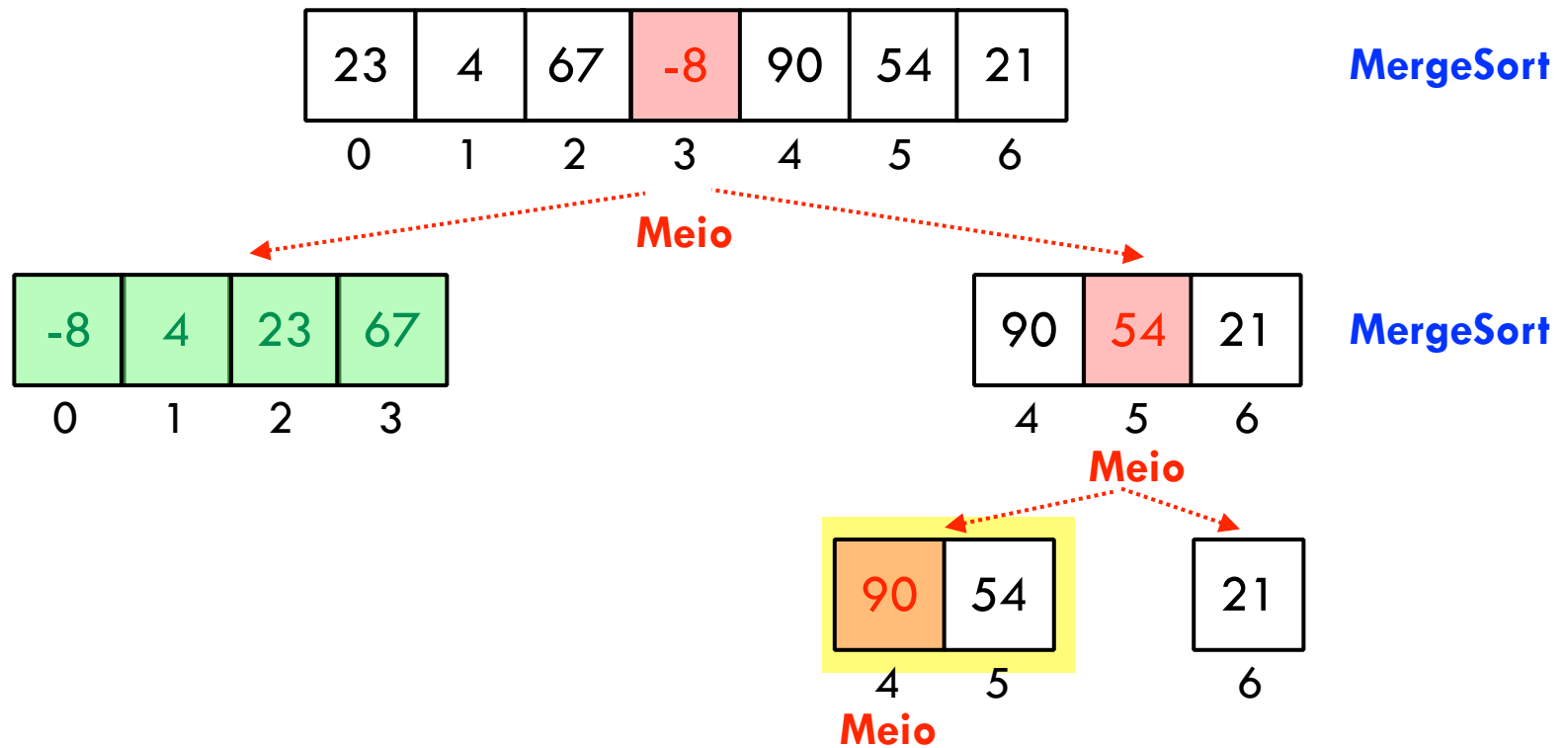
Exemplo



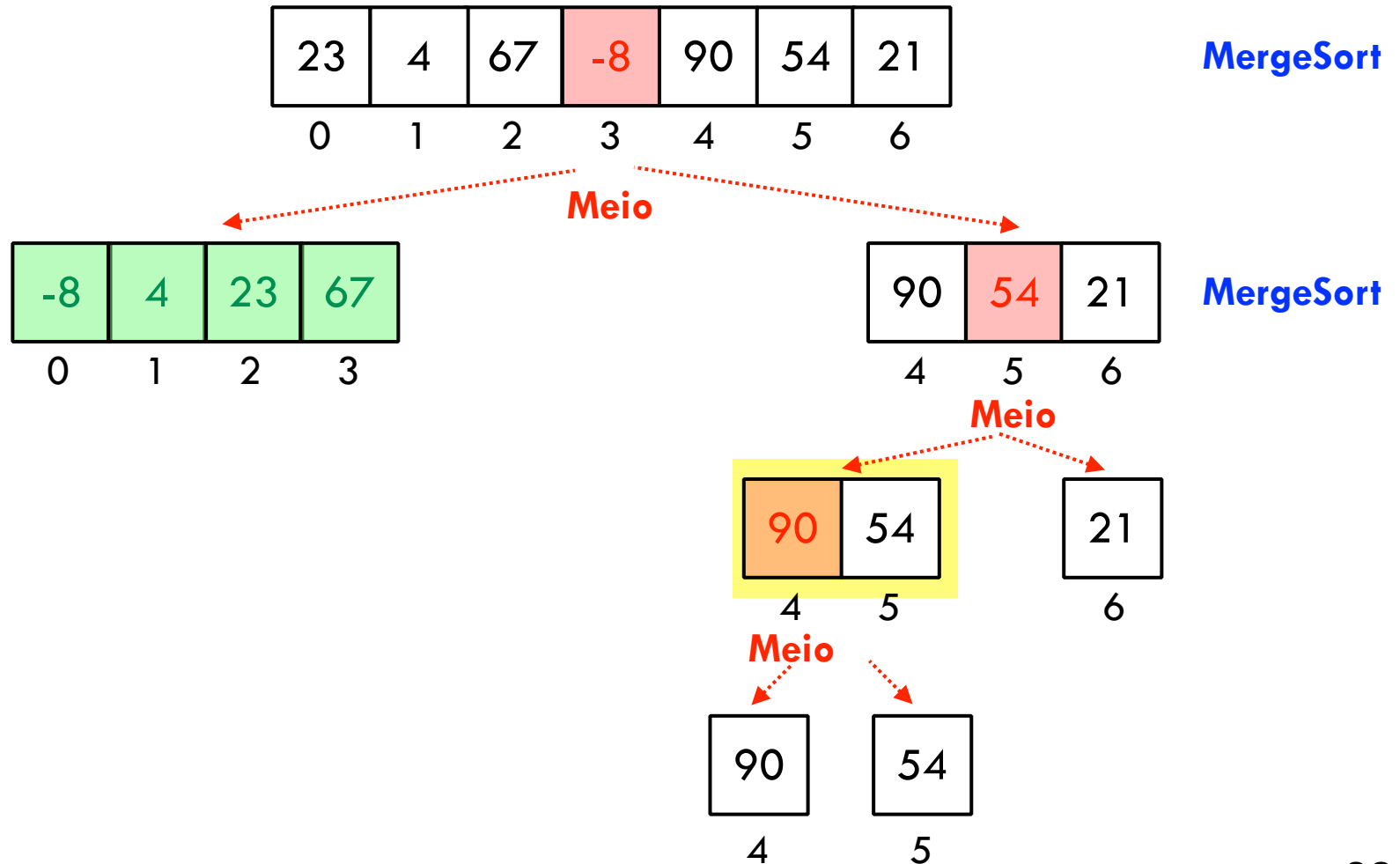
Exemplo



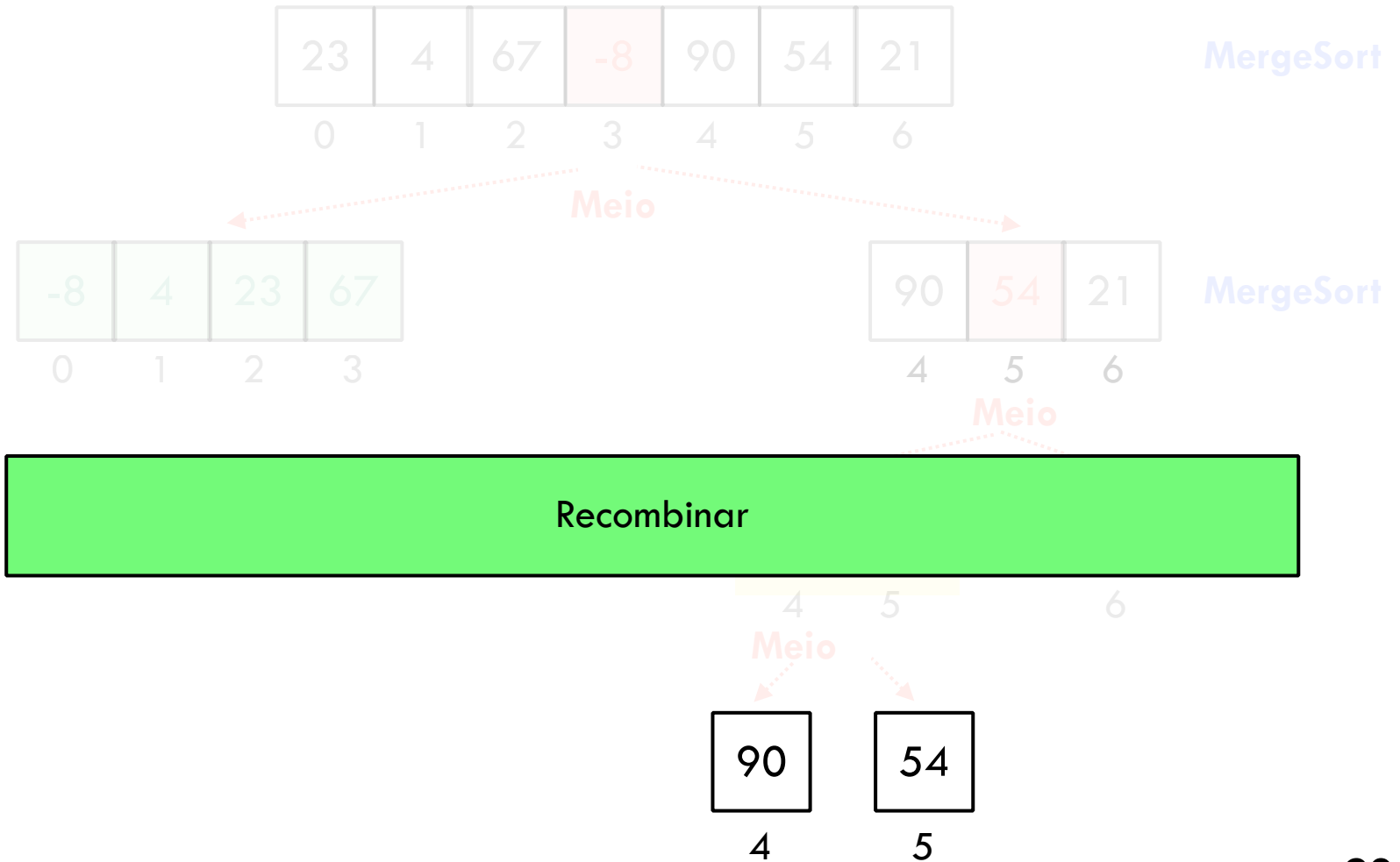
Exemplo



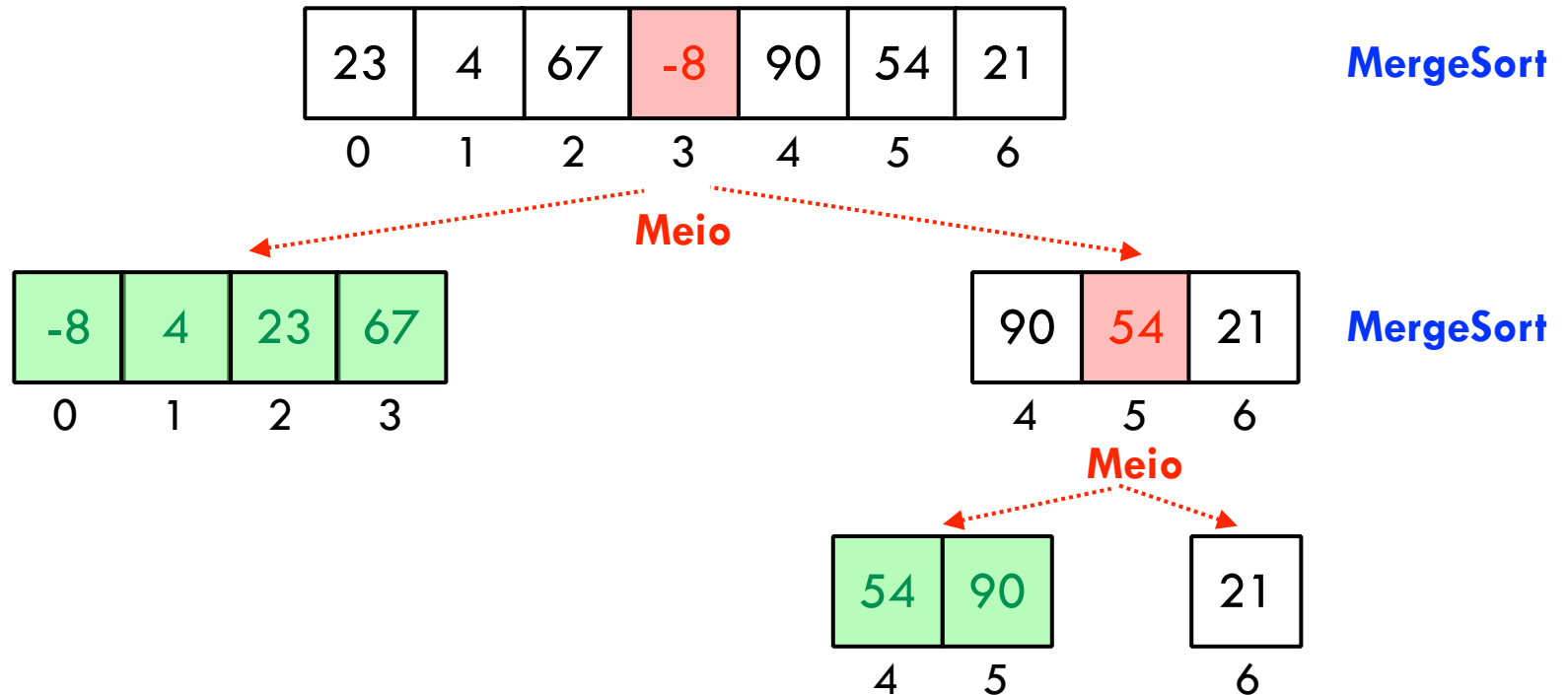
Exemplo



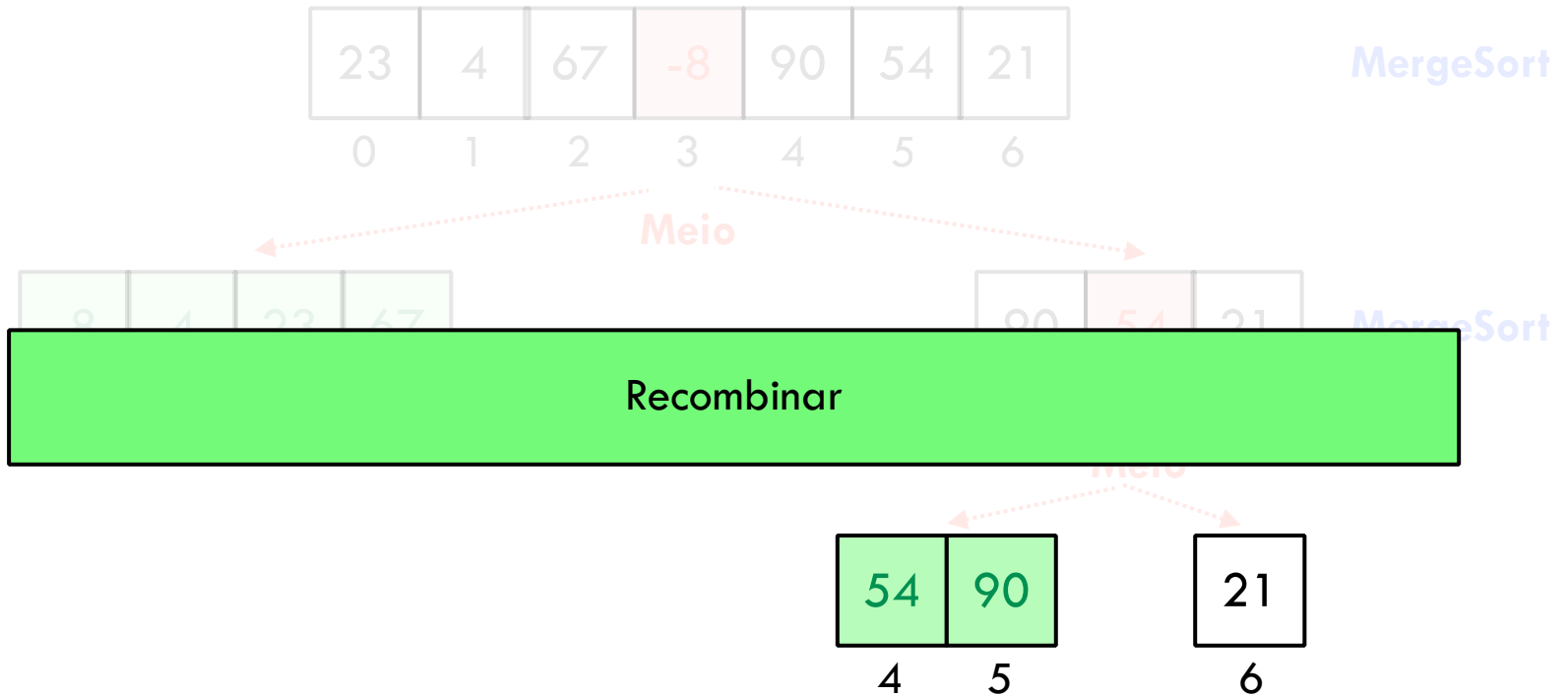
Exemplo



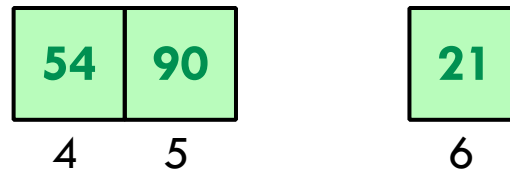
Exemplo



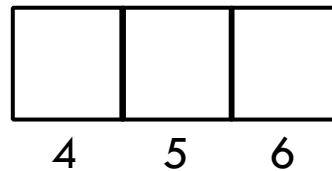
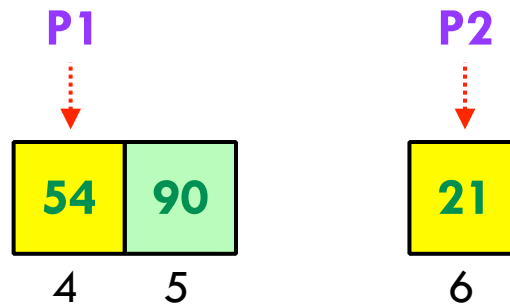
Exemplo



Exemplo

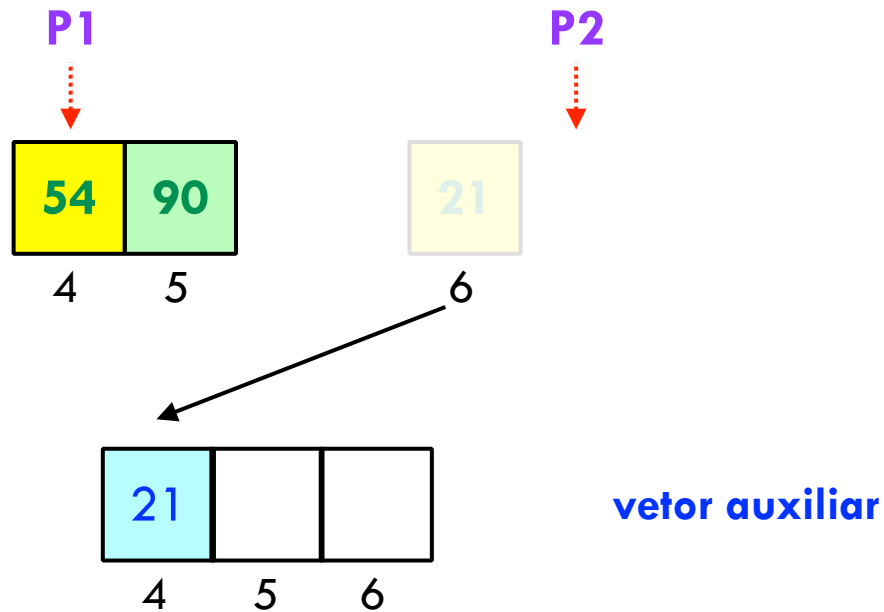


Exemplo

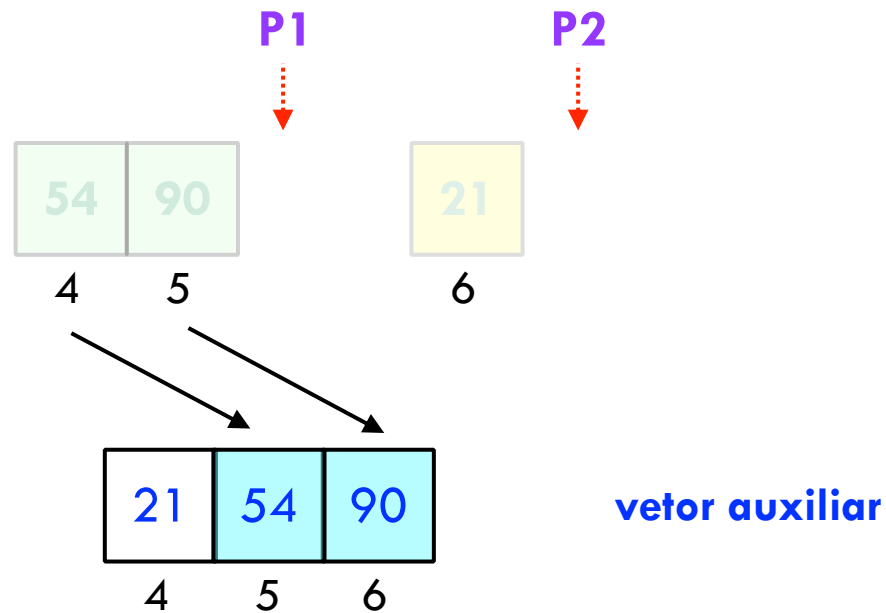


vetor auxiliar

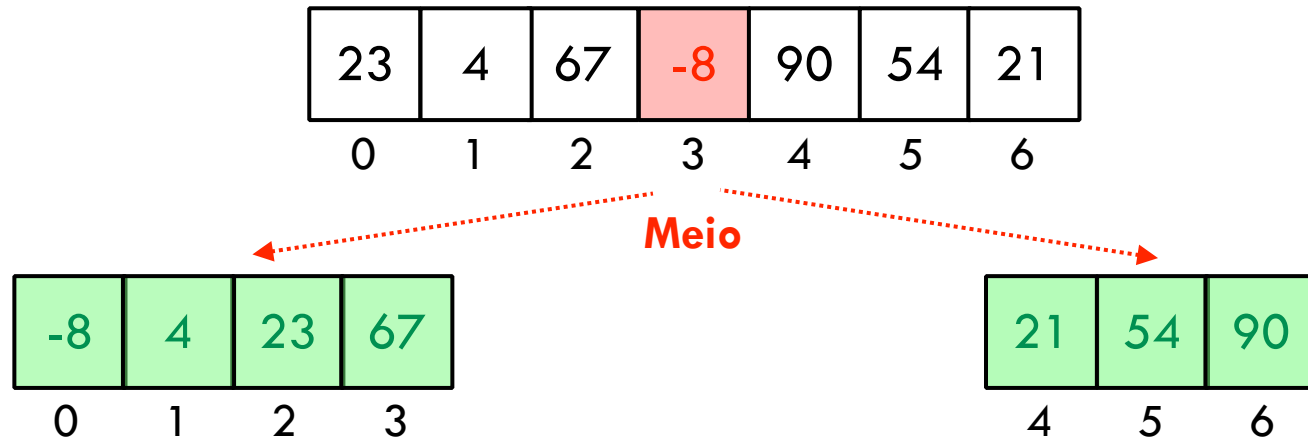
Exemplo



Exemplo

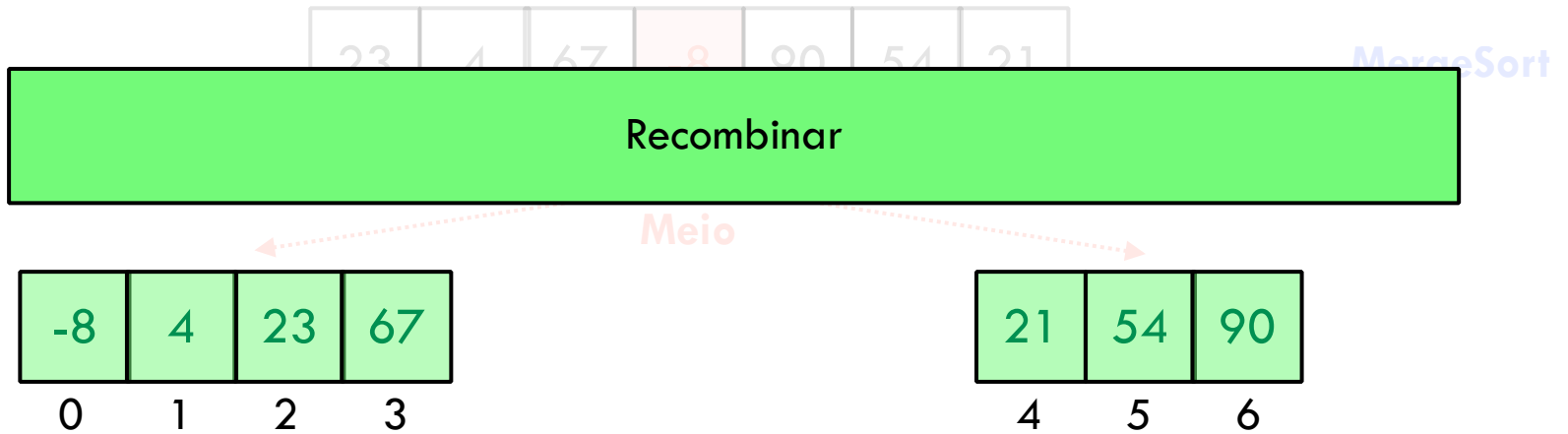


Exemplo



MergeSort

Exemplo

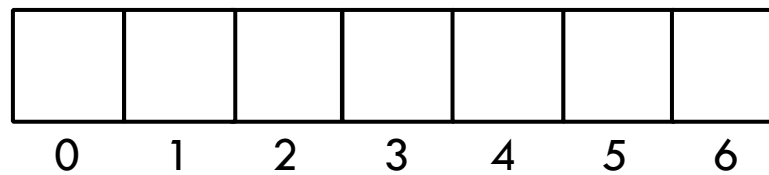
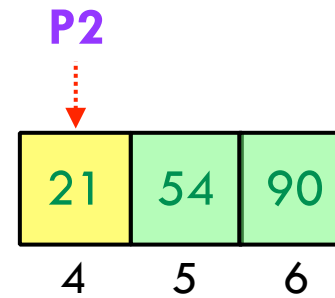
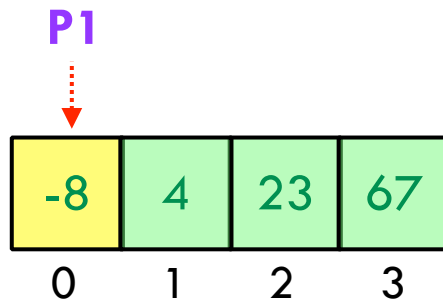


Exemplo

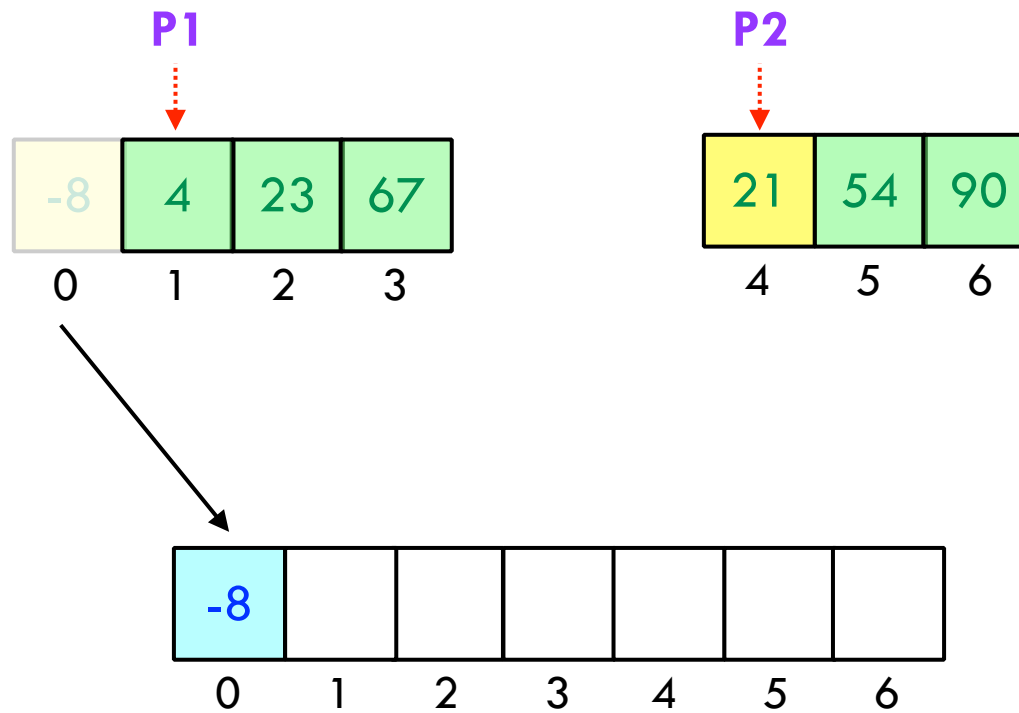
-8	4	23	67
0	1	2	3

21	54	90
4	5	6

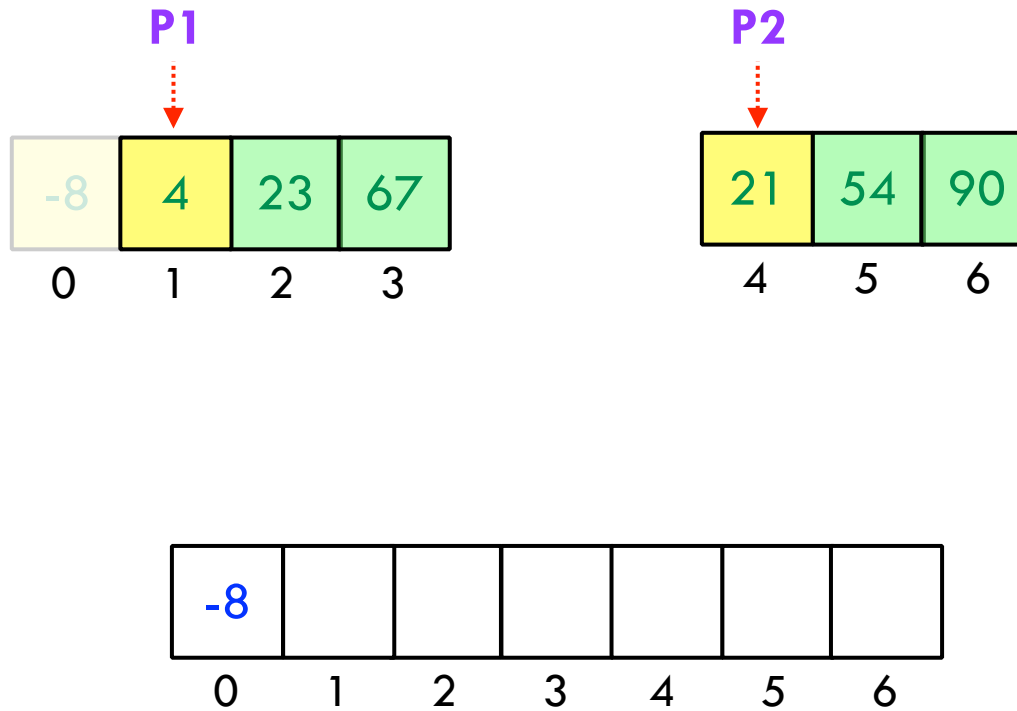
Exemplo



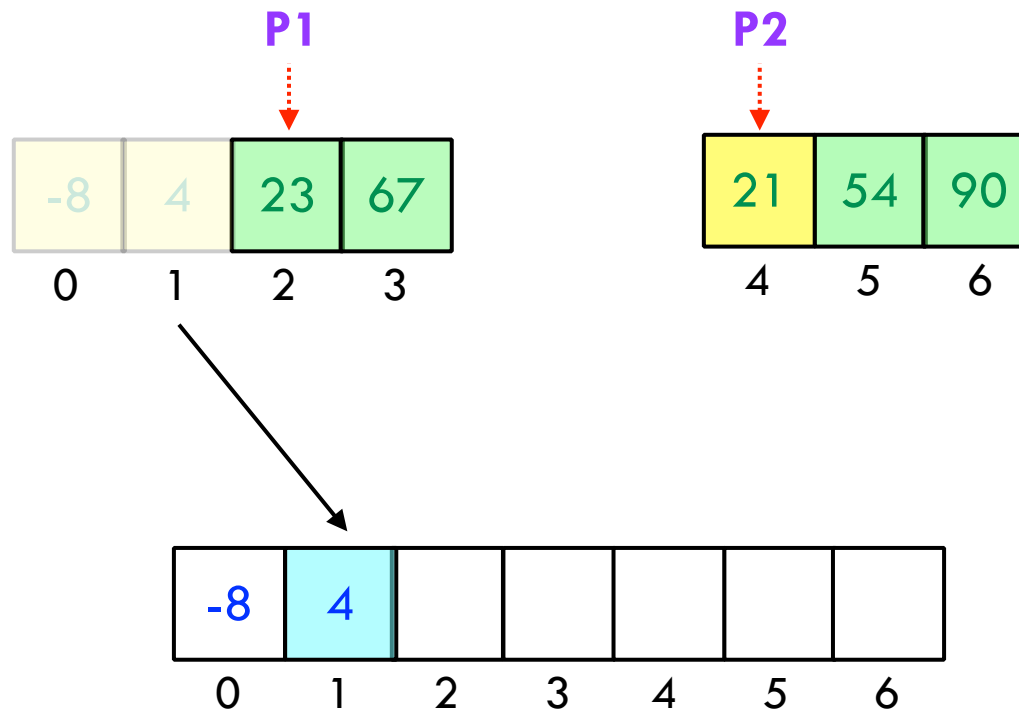
Exemplo



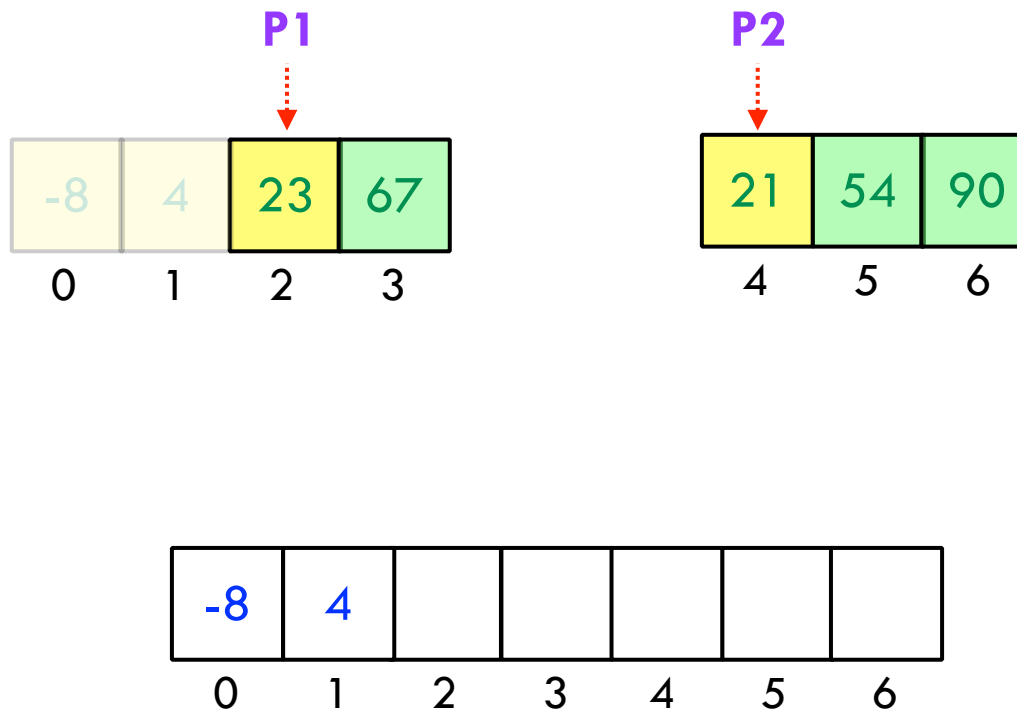
Exemplo



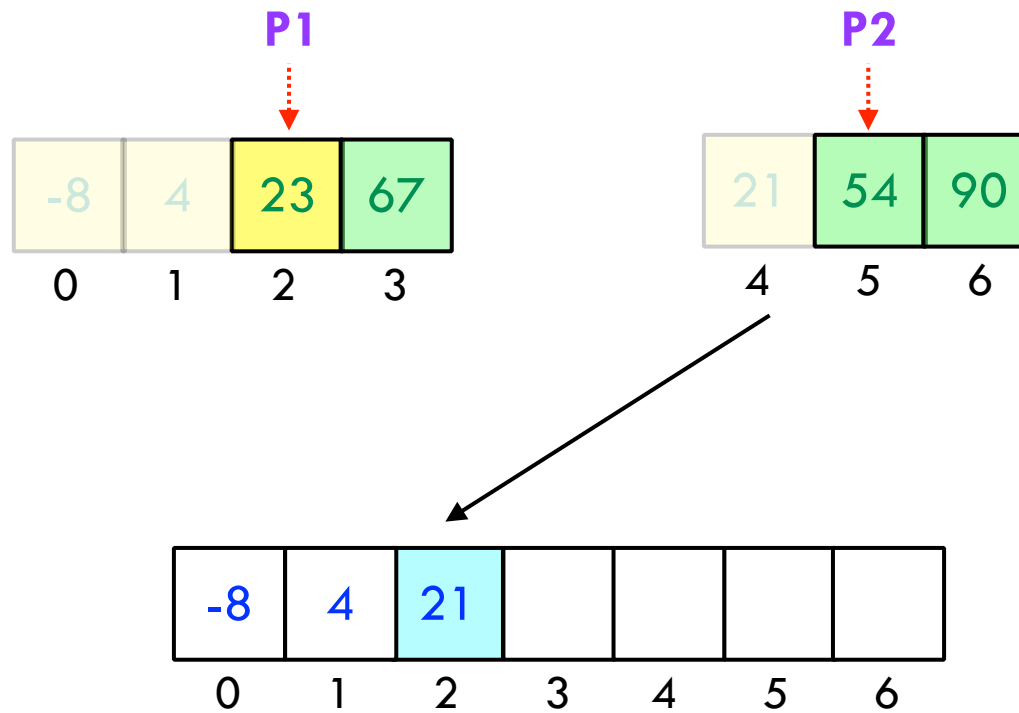
Exemplo



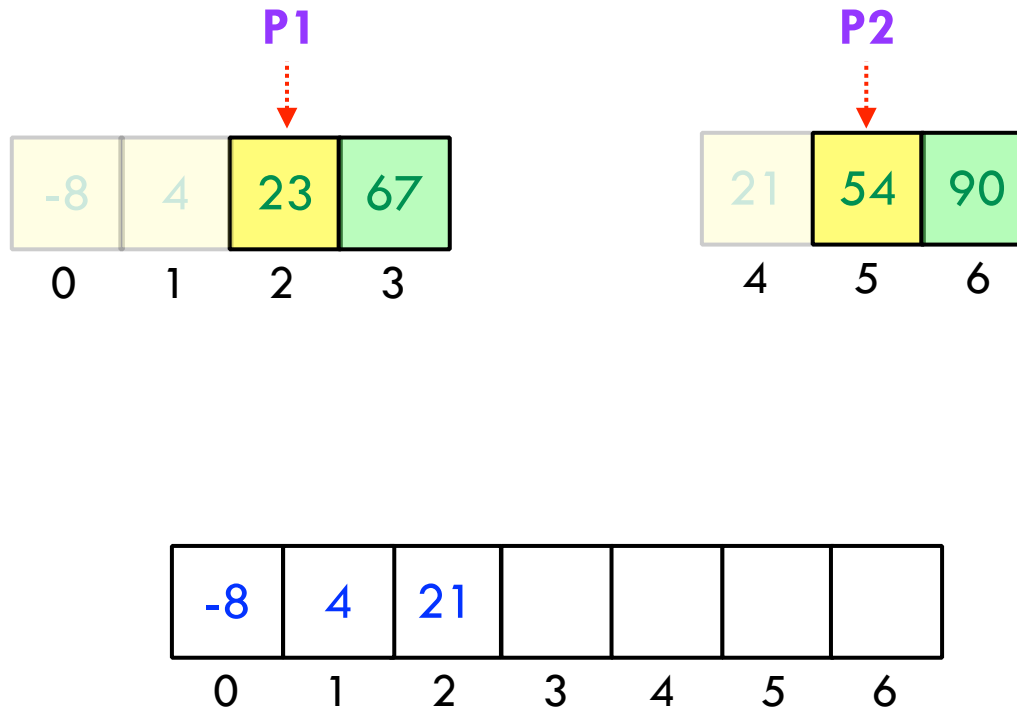
Exemplo



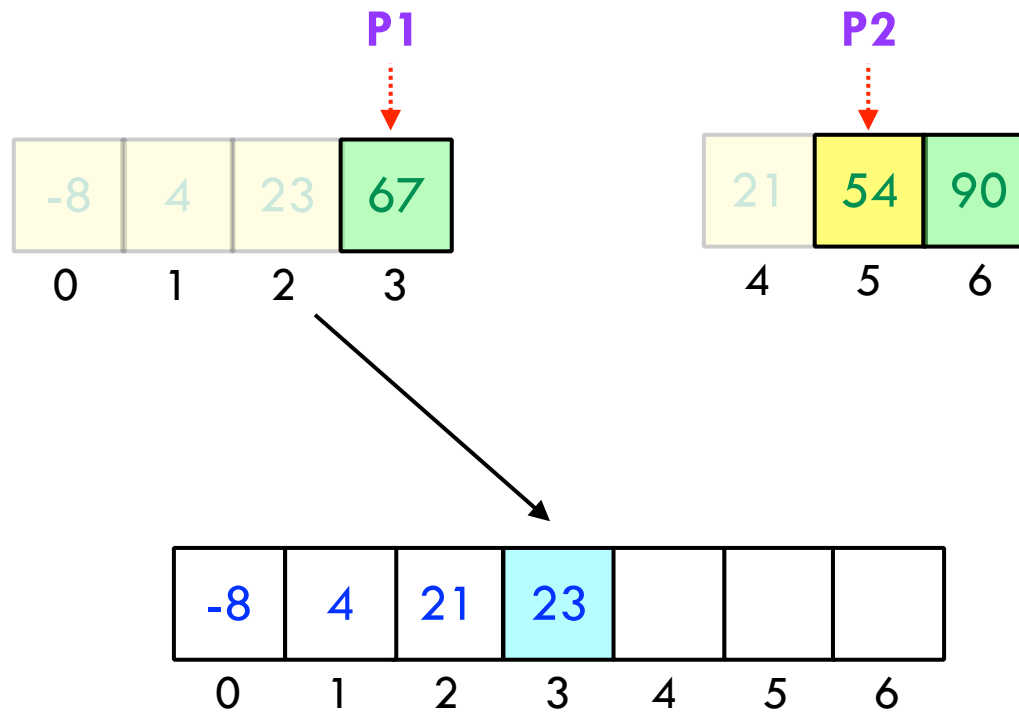
Exemplo



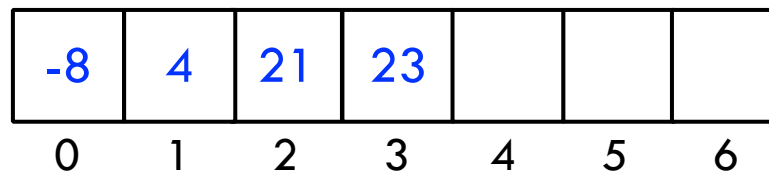
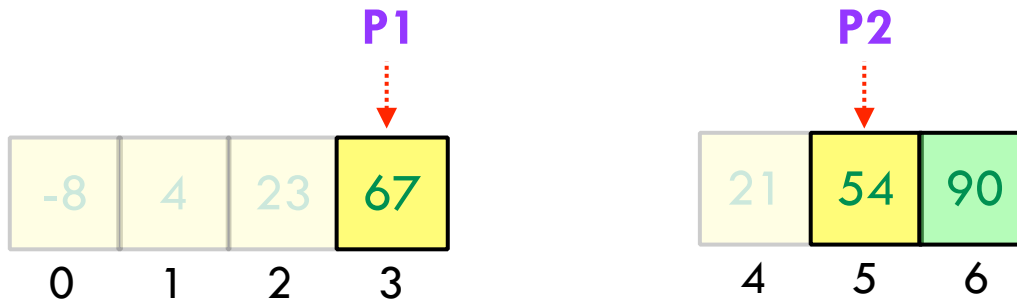
Exemplo



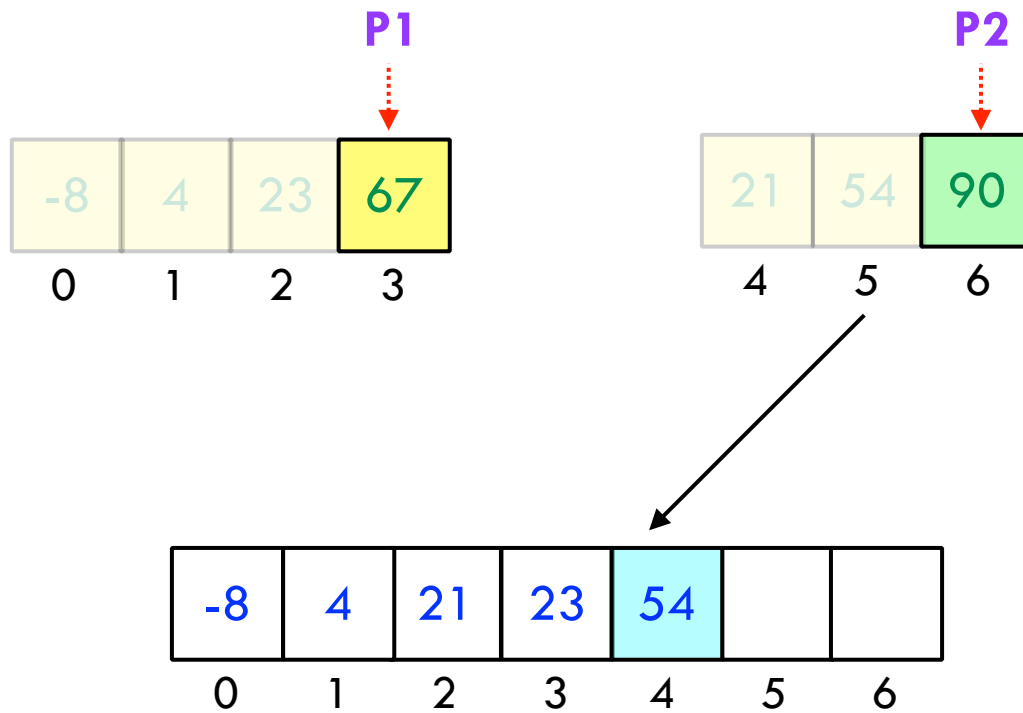
Exemplo



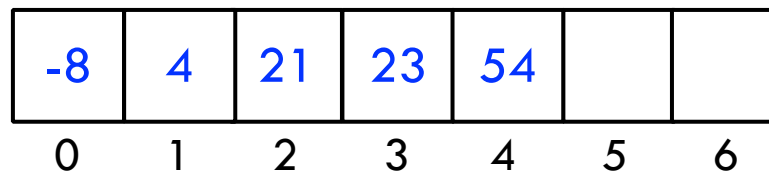
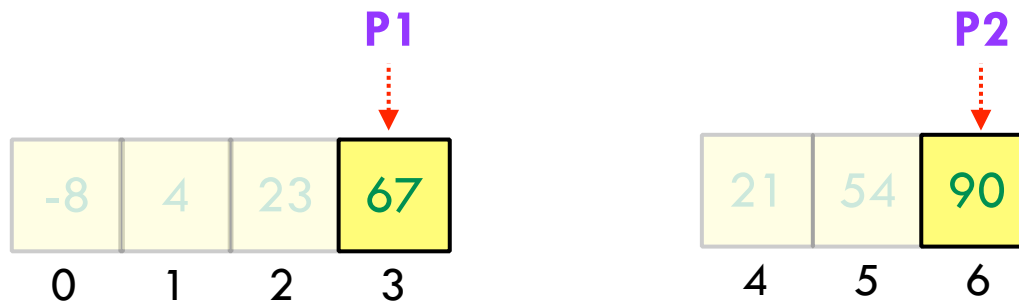
Exemplo



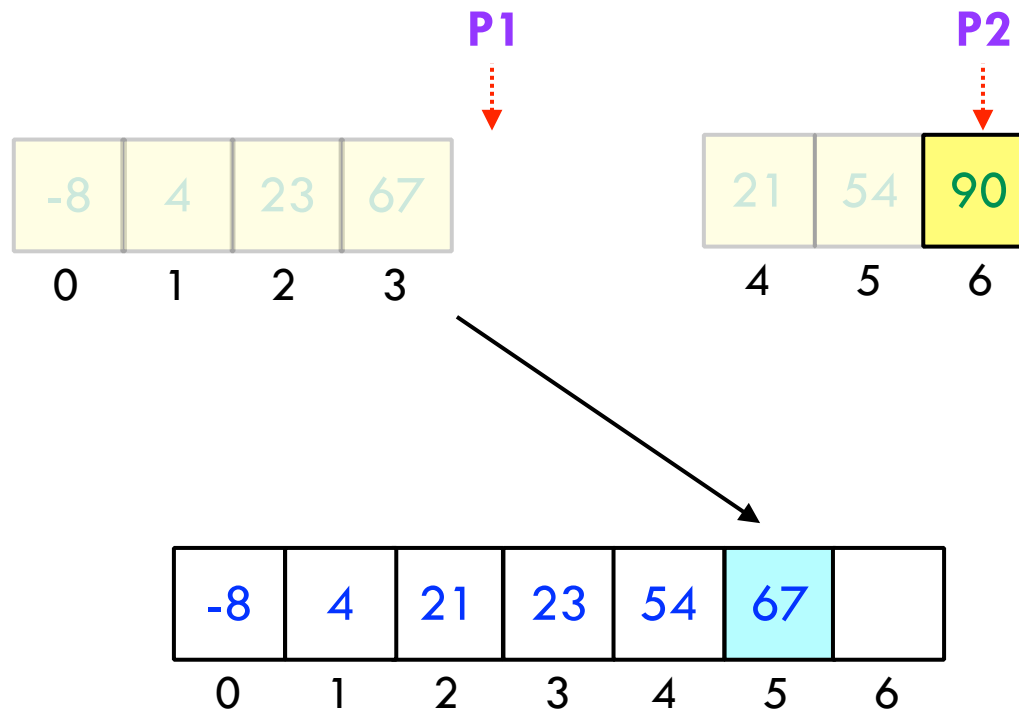
Exemplo



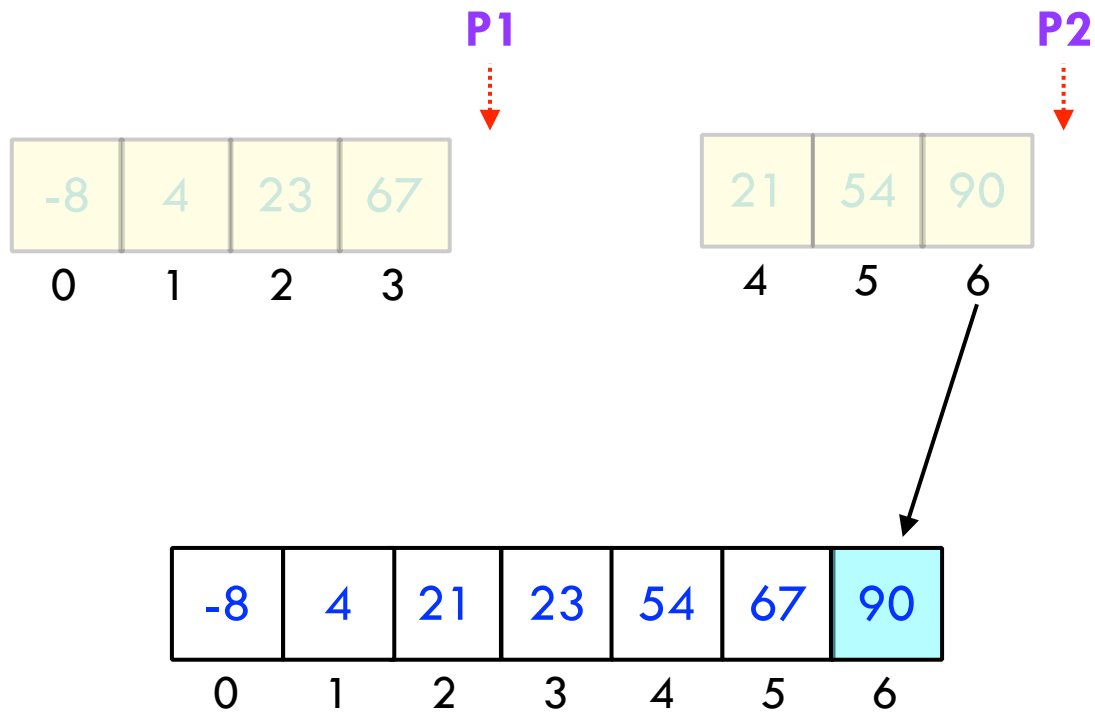
Exemplo




Exemplo



Exemplo



Exemplo



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

vetor ordenado

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) Execute o teste de mesa (simulação) do algoritmo **Merge 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 Merge Sort
```

```
Parâmetros:
```

```
array: vetor a ser ordenado
```

```
option: 1 - ordenação crescente, 2 - ordenação decrescente
```

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

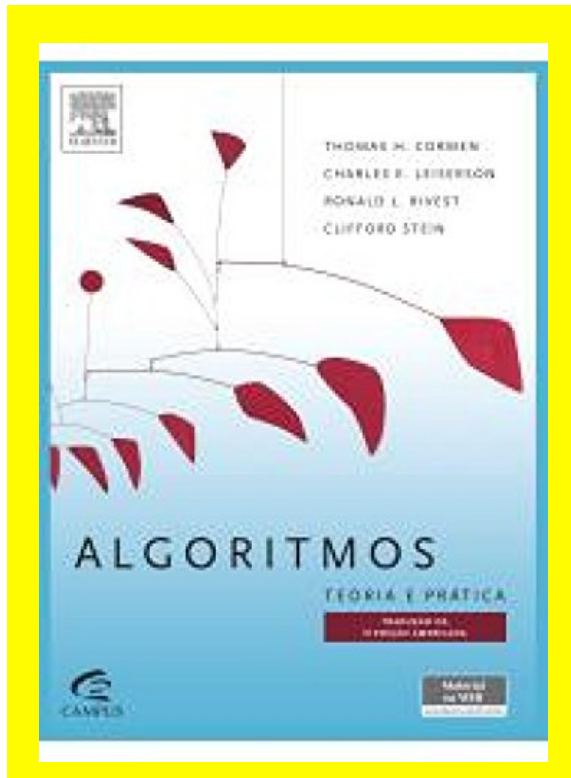
```
def mergeSort(array, option):
```

Roteiro



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

Referências sugeridas

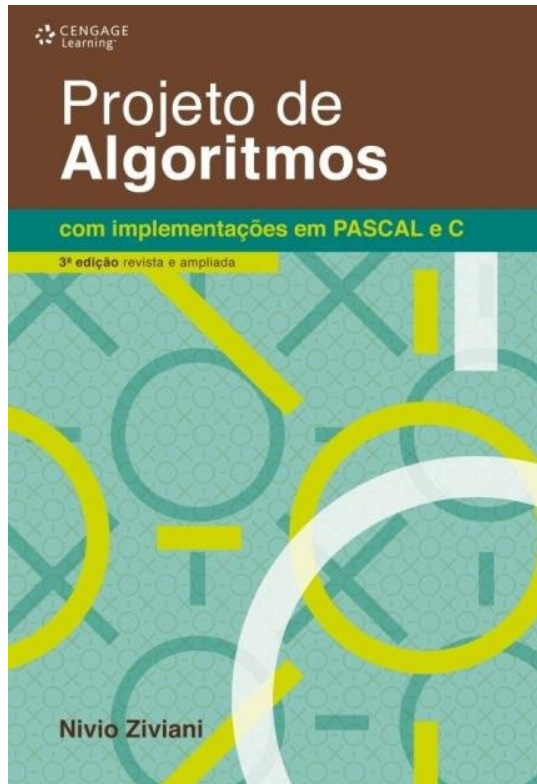


[Cormen et al, 2018]

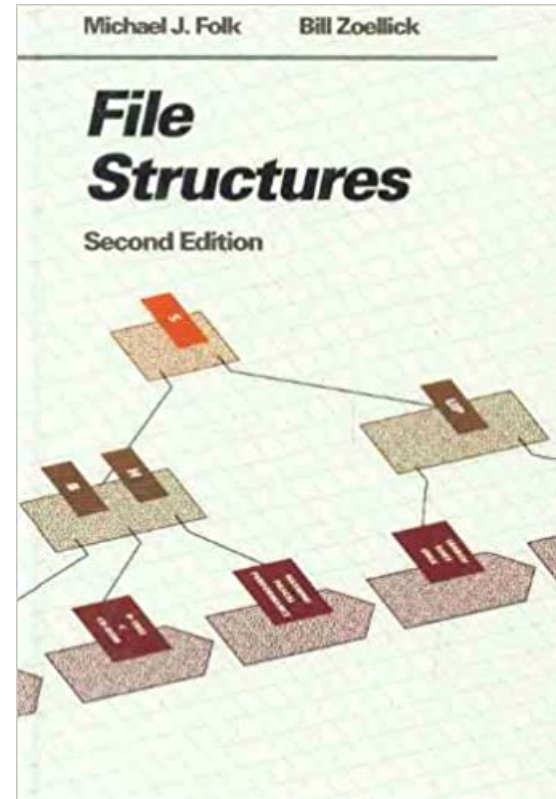


[Drozdek, 2017]

Referências sugeridas



[Ziviani, 2010]



[Folk & Zoellick, 1992]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br