

EDCO4B

ESTRUTURAS DE DADOS 2

Aula 07 - Heap Sort

Prof. Rafael G. Mantovani

Roteiro



- 1** Introdução
- 2** Heaps
- 3** Heap Sort
- 4** Exemplo
- 5** Exercícios
- 6** Referências

Roteiro

- 1** Introdução
- 2** Heaps
- 3** Heap Sort
- 4** Exemplo
- 5** Exercícios
- 6** Referências

Introdução



**Algoritmos de
Ordenação**

Introdução

Métodos Simples

Métodos Eficientes

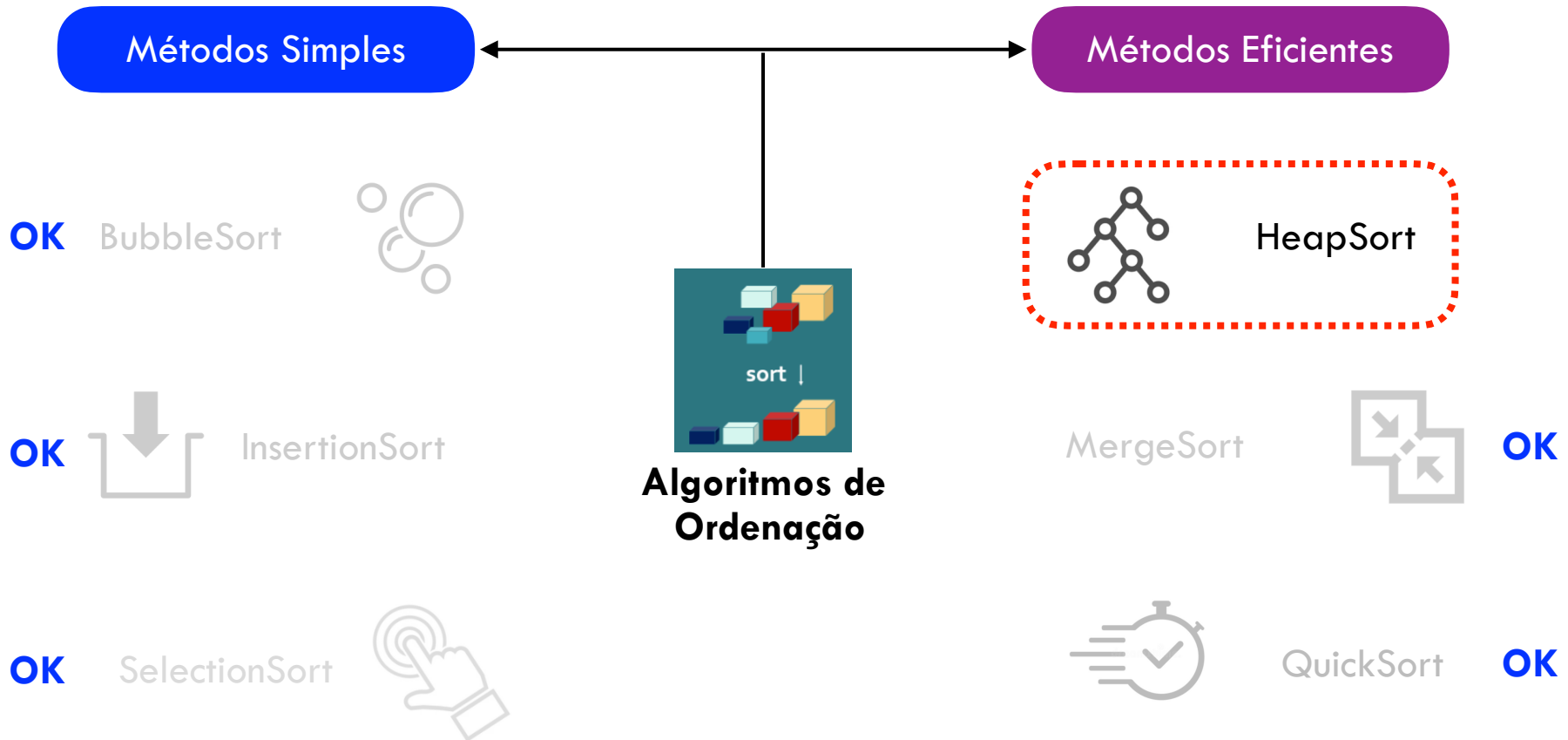


**Algoritmos de
Ordenação**

Introdução



Introdução



Roteiro

- 1 Introdução
- 2 Heaps
- 3 Heap Sort
- 4 Exemplo
- 5 Exercícios
- 6 Referências

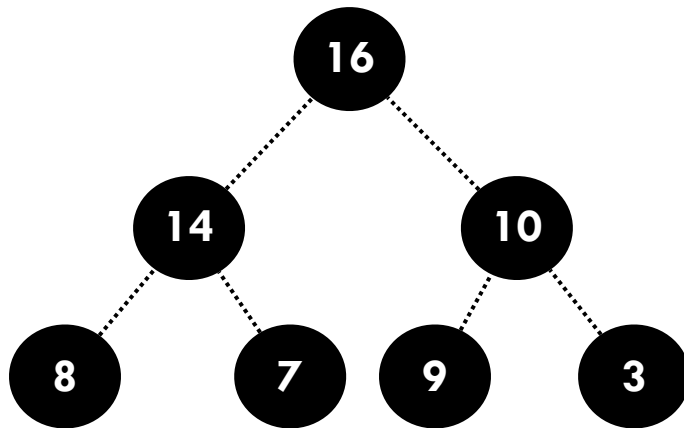
Heap



- **Heap:** estrutura de dados em array que pode ser vista como uma árvore binária quase completa

Heap

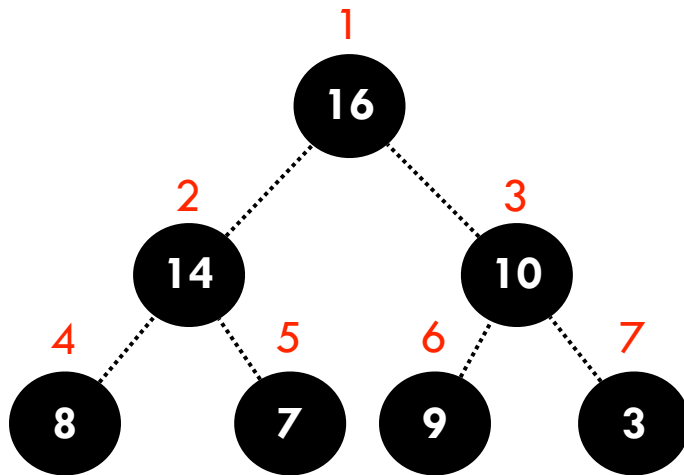
- **Heap:** estrutura de dados em array que pode ser vista como uma árvore binária quase completa



a) heap como árvore binária

Heap

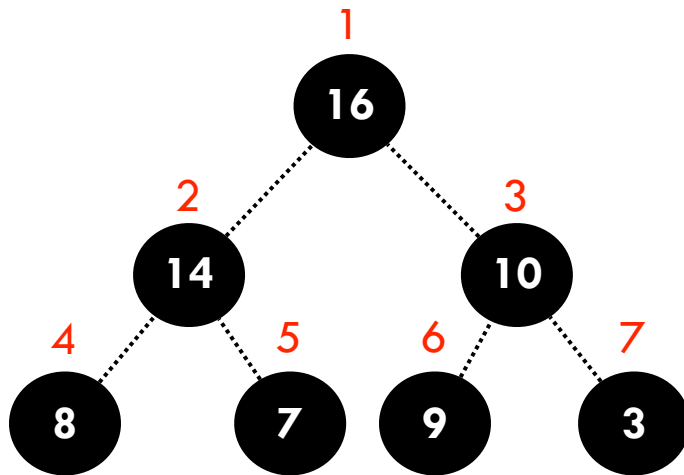
- **Heap:** estrutura de dados em array que pode ser vista como uma árvore binária quase completa



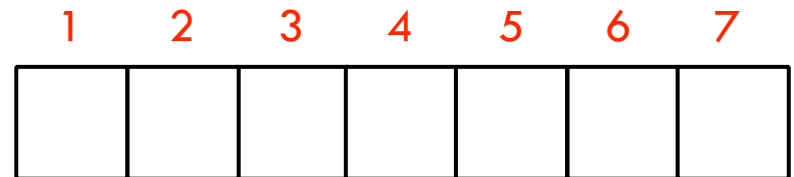
a) heap como árvore binária

Heap

- **Heap:** estrutura de dados em array que pode ser vista como uma árvore binária quase completa



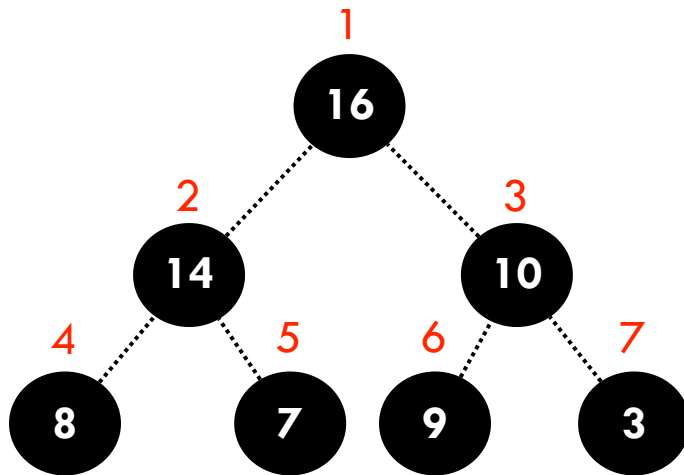
a) heap como árvore binária



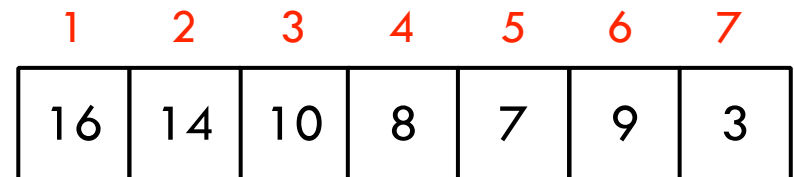
b) heap como array

Heap

- **Heap:** estrutura de dados em array que pode ser vista como uma árvore binária quase completa



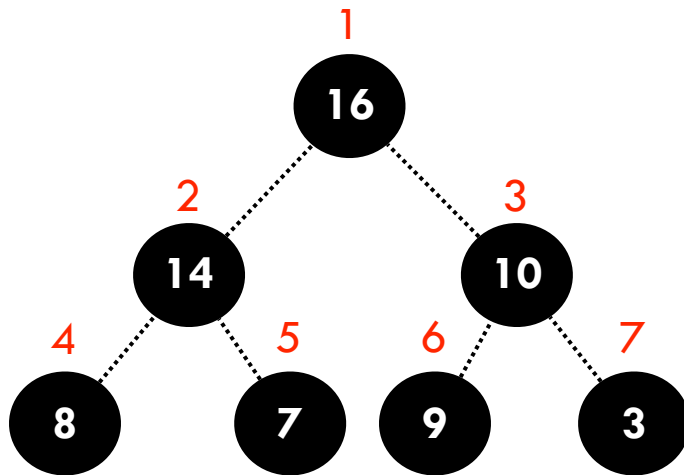
a) heap como árvore binária



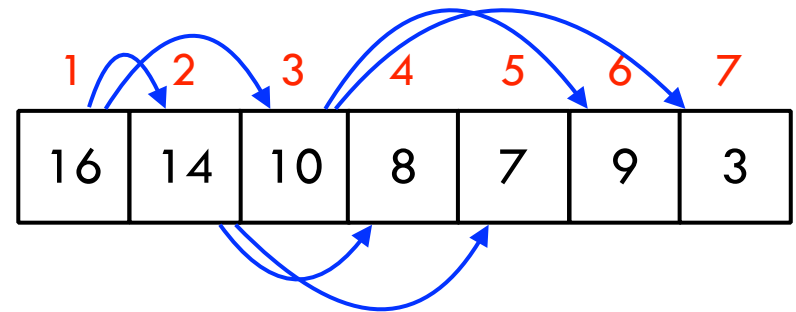
b) heap como array

Heap

- **Heap:** estrutura de dados em array que pode ser vista como uma árvore binária quase completa



a) heap como árvore binária



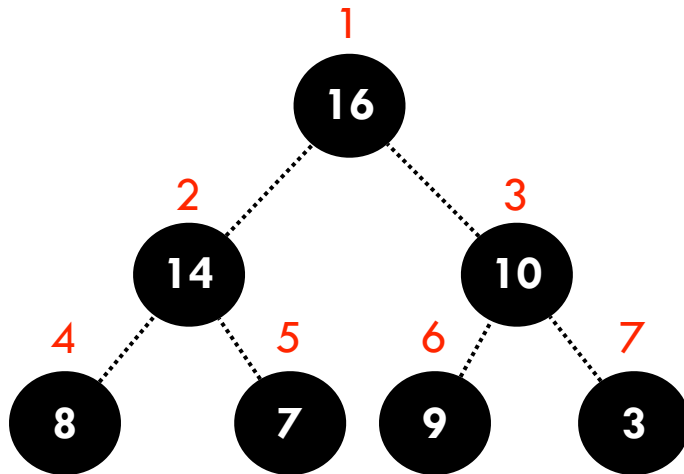
b) heap como array

Heap

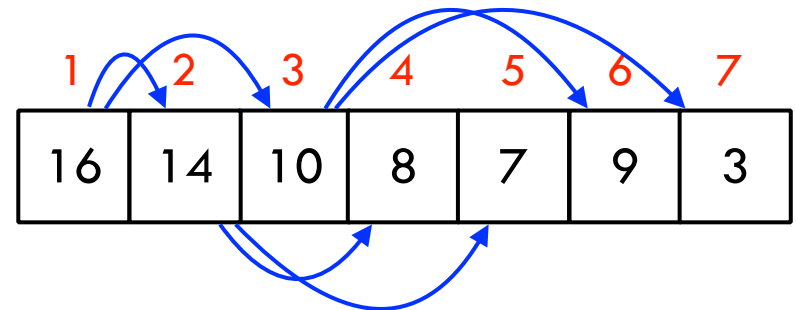
- * $\text{Pai}(i) = \text{floor}(i/2)$
- * $\text{Esquerda}(i) = 2*i$
- * $\text{Direita}(i) = 2*i + 1$

Heap

* $\text{Pai}(i) = \text{floor}(i/2)$
* $\text{Esquerda}(i) = 2*i$
* $\text{Direita}(i) = 2*i + 1$



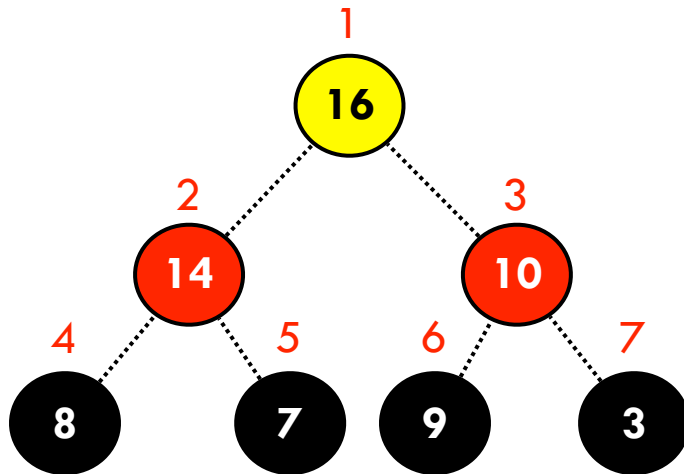
a) heap como árvore binária



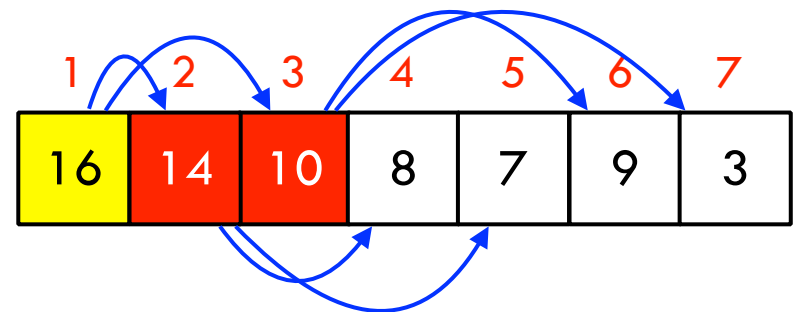
b) heap como array

Heap

* Esquerda(1) = $2 \cdot 1 = 2$
* Direita(1) = $2 \cdot 1 + 1 = 3$



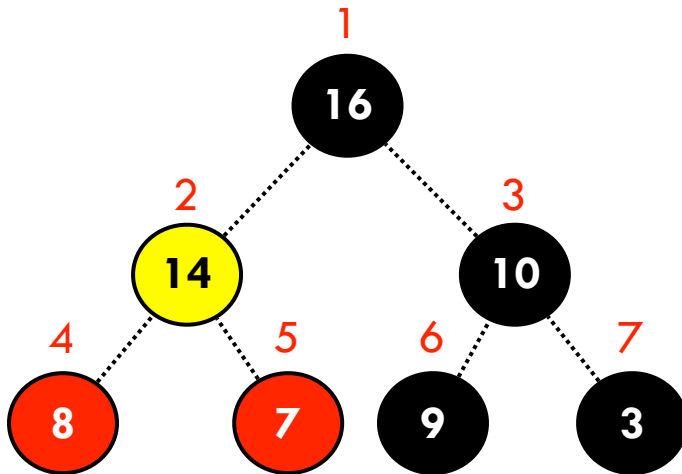
a) heap como árvore binária



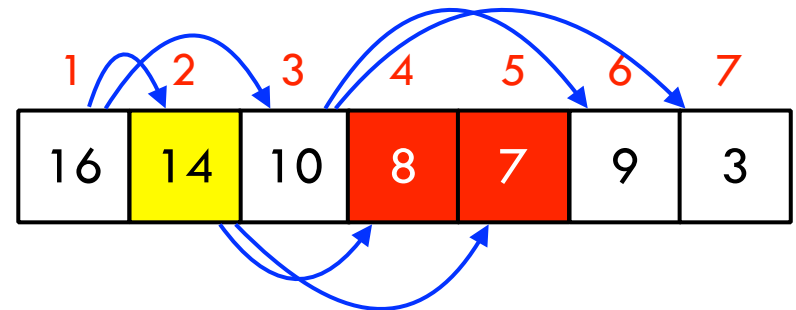
b) heap como array

Heap

* $\text{Pai}(2) = \text{floor}(2/2) = 1$
* $\text{Esquerda}(2) = 2*2 = 4$
* $\text{Direita}(2) = 2*2 + 1 = 5$



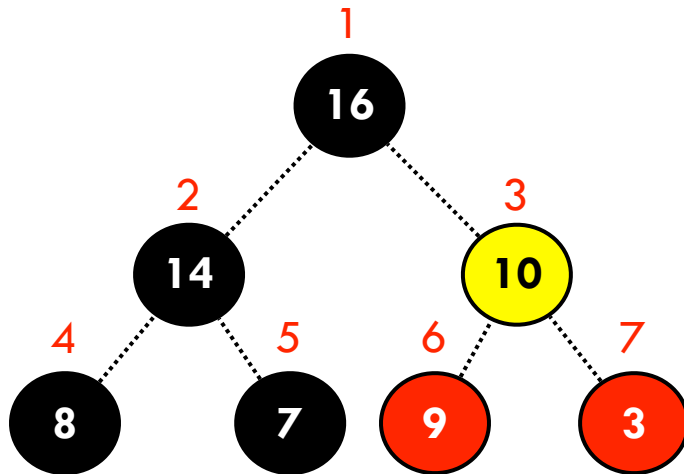
a) heap como árvore binária



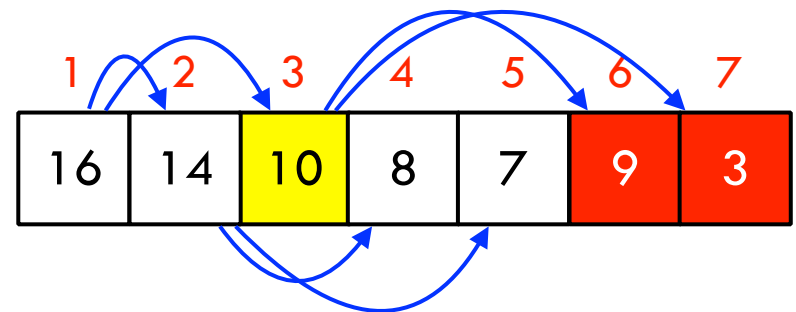
b) heap como array

Heap

* $\text{Pai}(3) = \text{floor}(3/2) = 1$
* $\text{Esquerda}(3) = 2*3 = 6$
* $\text{Direita}(3) = 2*3 + 1 = 7$



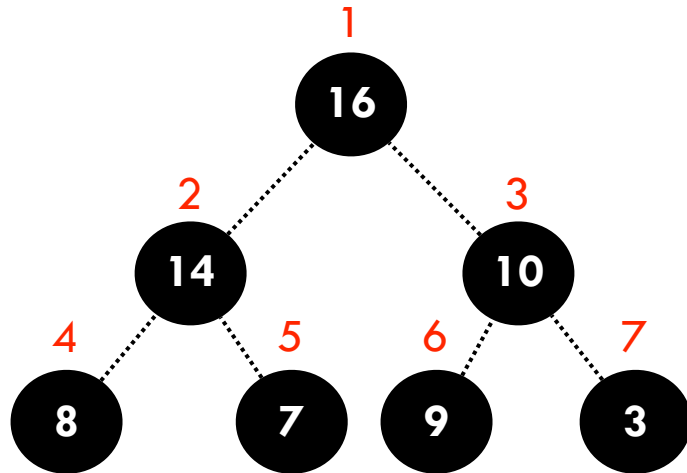
a) heap como árvore binária



b) heap como array

Heap

- **Heap de máximo:**
 - $A[\text{parent}(i)] \geq A[i]$

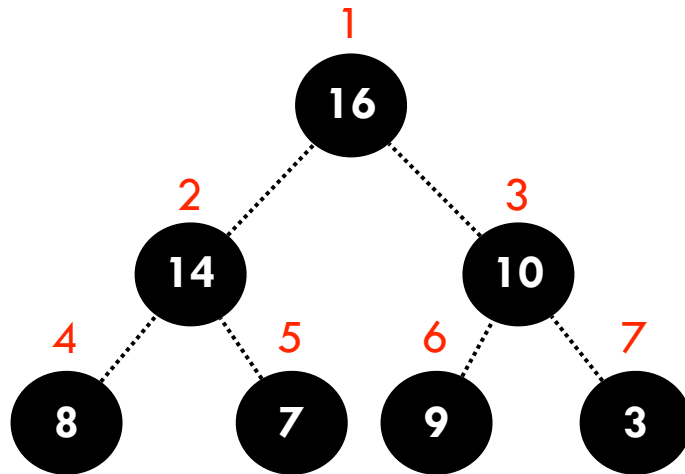


Heap

- **Heap de máximo:**

- $A[\text{Pai}(i)] \geq A[i]$

Maiores
valores



Menores
valores

Heap

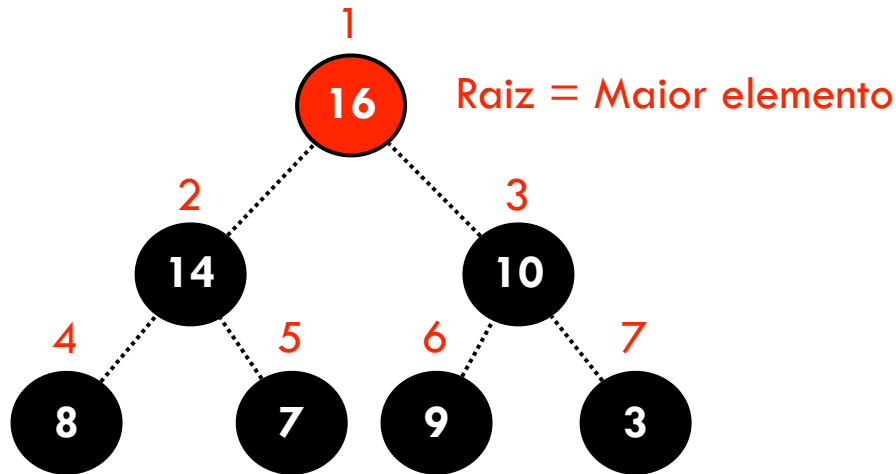
- **Heap de máximo:**

- $A[\text{Pai}(i)] \geq A[i]$

Maiores
valores

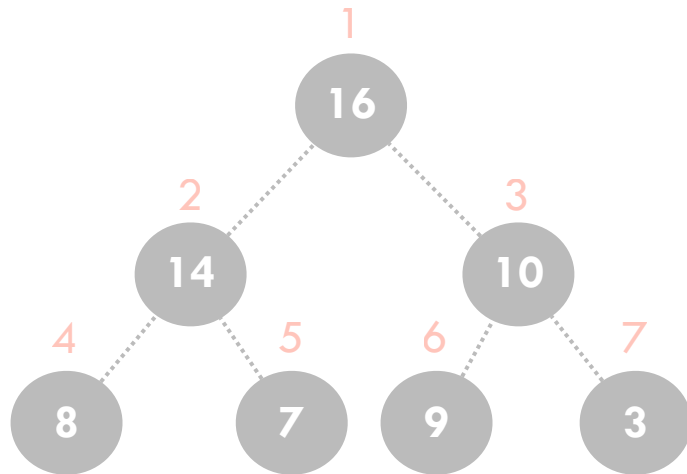


Menores
valores



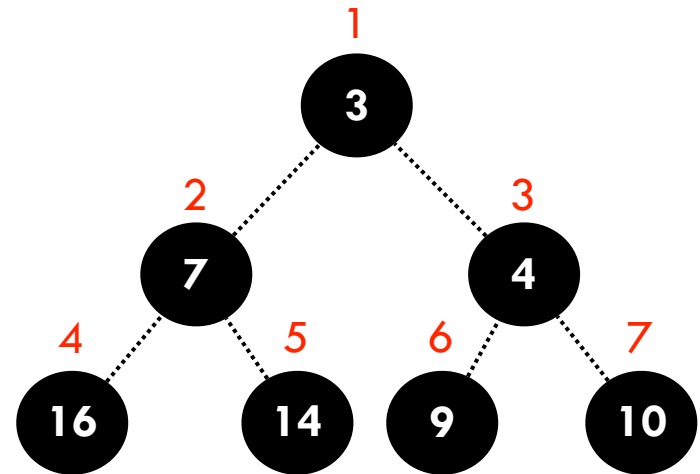
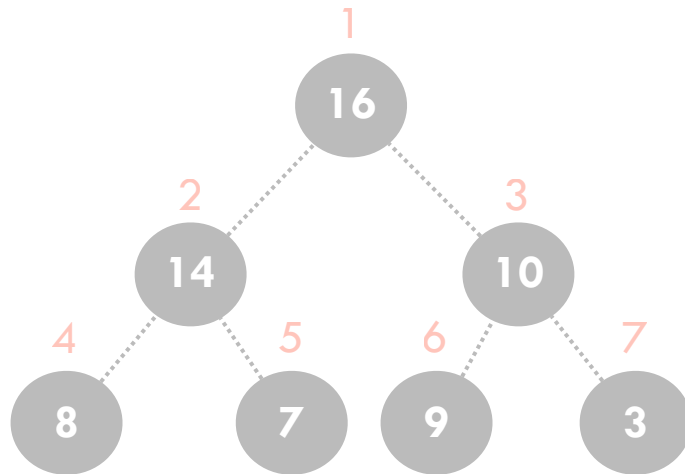
Heap

- **Heap de máximo:**
 - $A[\text{Pai}(i)] \geq A[i]$



Heap

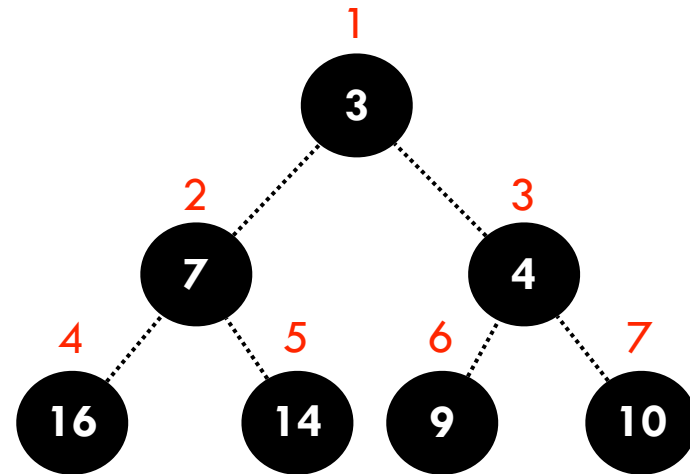
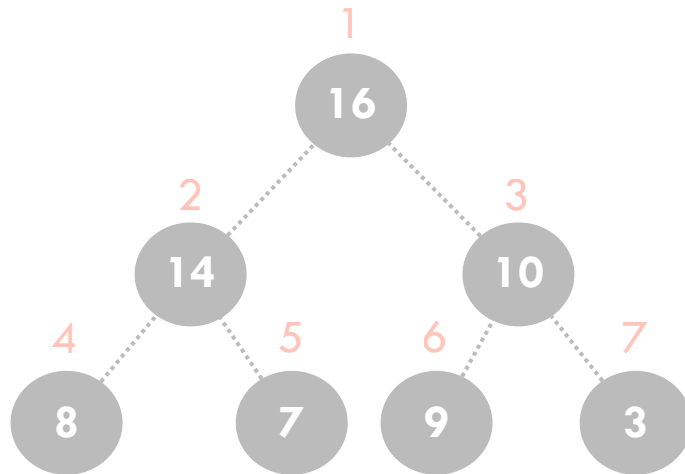
- **Heap de mínimo:**
 - $A[\text{Pai}(i)] \leq A[i]$



Heap

- **Heap de mínimo:**

- $A[\text{Pai}(i)] \leq A[i]$



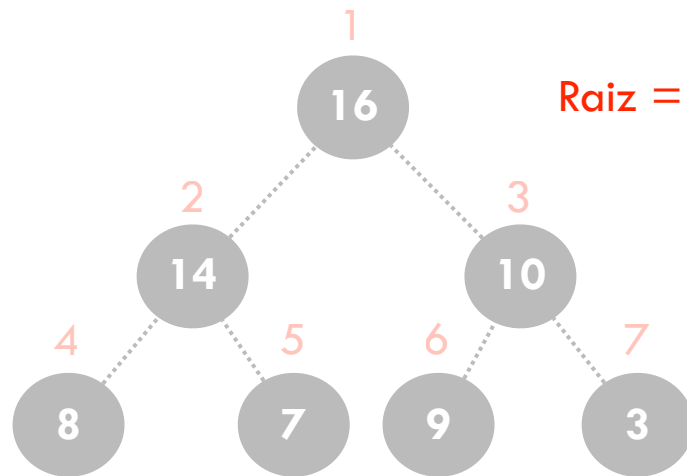
Menores
valores

Maiores
valores

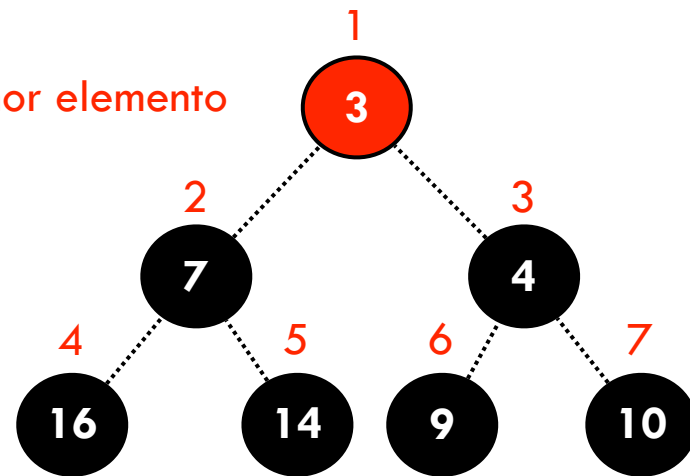
Heap

- **Heap de mínimo:**

- $A[\text{Pai}(i)] \leq A[i]$



Raiz = Menor elemento



Menores
valores

Maiores
valores

Roteiro

- 1 Introdução
- 2 Heaps
- 3 Heap Sort
- 4 Exemplo
- 5 Exercícios
- 6 Referências

Heap Sort



Heap Sort

□ Funcionamento

* Passos

1. Transformar o heap (vetor) em um max-heap
2. Mover a raiz (maior valor) para o final do vetor (última posição)
3. Reconstrói o heap do vetor restante

Heap Sort

□ Desempenho

* **melhor caso:** $O(N \log N)$

* **pior caso:** $O(N \log N)$

* **caso médio:** $O(N \log N)$

Heap Sort

□ Desempenho

* **melhor caso:** $O(N \log N)$

* **pior caso:** $O(N \log N)$

* **caso médio:** $O(N \log N)$

Obs: Na prática, o HeapSort é mais lento que o QuickSort, exceto no pior caso.

Heap Sort

□ Pseudocódigo

1. **HEAPSORT**: ordena o vetor usando heap auxiliar
2. **BUILD-MAX-HEAP**: cria um heap de máximo a partir dos valores aleatórios de um vetor
3. **MAX-HEAPIFY**: mantém a propriedade de heap de máximo de um conjunto de valores

Heap Sort

- **Pseudocódigo** (mantém heap)

1. **MAX-HEAPIFY** (V, i)

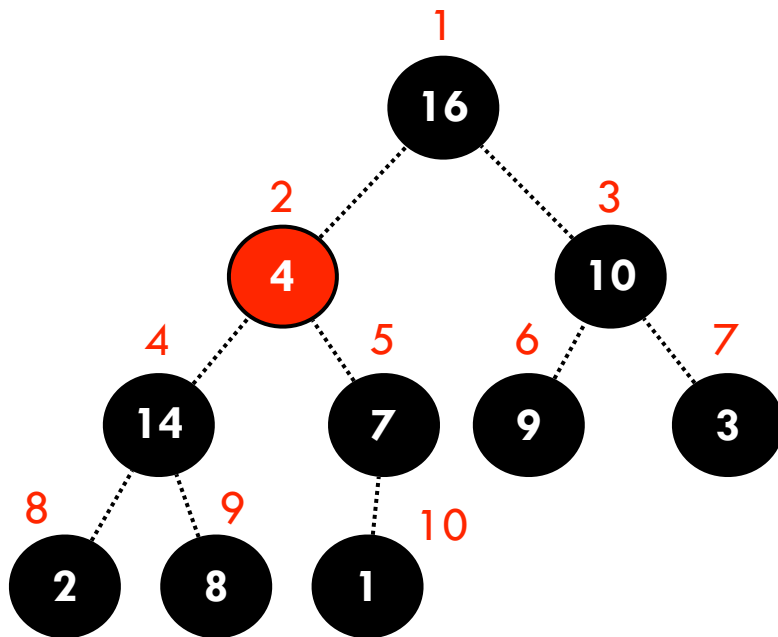
- Entradas:
 - **V**: vetor de elementos
 - **i**: índice da posição para se criar max-heap
- Além disso, em V podemos obter:
 - **V.tamanho-heap**: tamanho a ser considerado para criar heap
 - **V.tamanho**: tamanho do vetor

Heap Sort

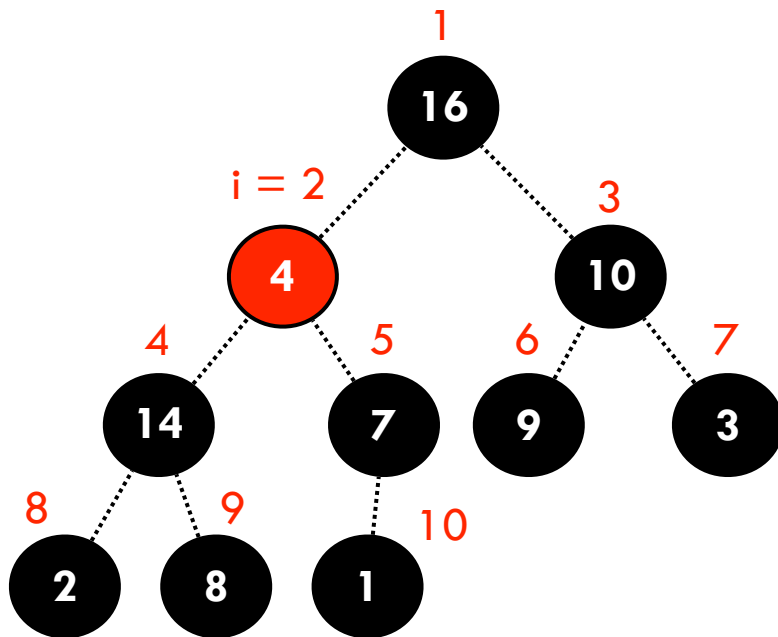
□ Pseudocódigo (cria heap)

```
1. MAX-HEAPIFY (V, i)
2.   L = Esquerda(i) //  $2*i + 1$ 
3.   R = Direita(i)   //  $2*i + 2$ 
4.   Maior = i
5.   se L <= V.tamanho-heap E V[L] > V[i]
6.       maior = L
7.   se R <= V.tamanho-heap E V[R] > V[maior]
8.       maior = R
9.   se maior != i
10.       troca(V[i], V[maior])
11.       MAX-HEAPIFY (V, maior)
```

MAX-HEAPIFY



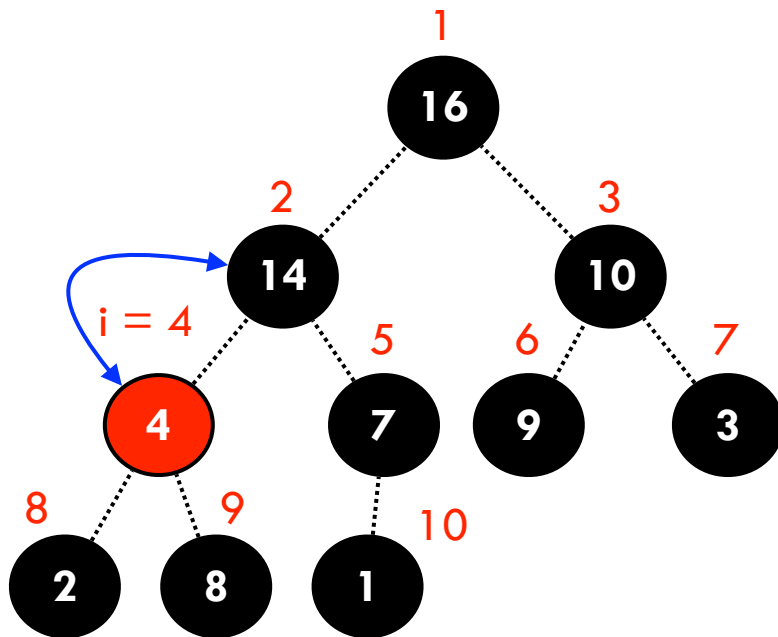
MAX-HEAPIFY



* Configuração inicial ($i = 2$)
 $V[2]$ viola a propriedade de heap máximo:
 $V[2] < V[4]$ e $V[2] < V[5]$

* **Solução:** SWAP($V[2]$, $V[4]$)

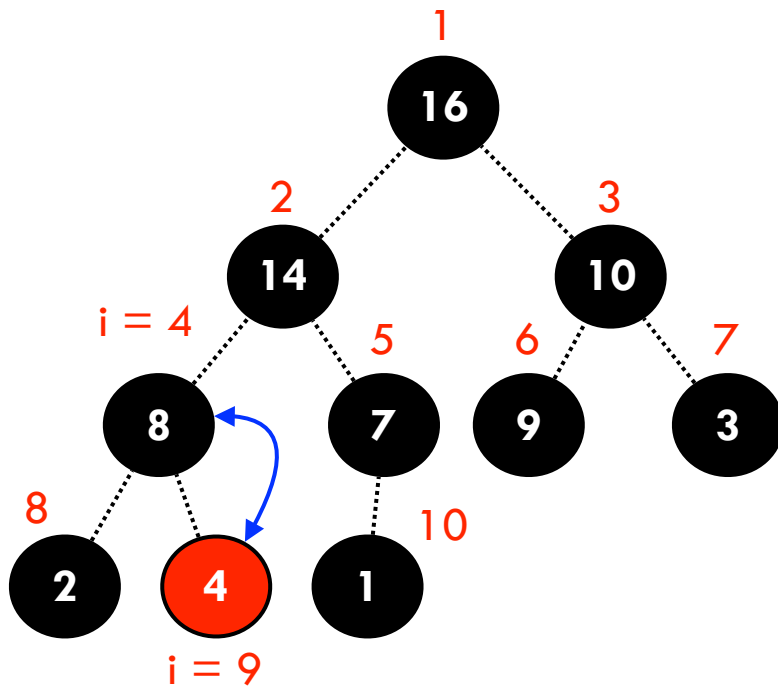
MAX-HEAPIFY



* Configuração atual ($i = 4$)
 $v[4]$ viola a propriedade de heap máximo:
 $V[4] < V[9]$

Solução: SWAP($V[4]$, $V[9]$)

MAX-HEAPIFY



Tudo certo agora :)

Heap Sort

- **Pseudocódigo** (cria max-heap)

2. **BUILD-MAX-HEAP** (V)

- Entradas:
 - **V**: vetor de elementos
- Usa **MAX-HEAPIFY** em uma abordagem *bottom up* convertendo o array V em um max-heap
 - iterar de $N/2$... até 1

Heap Sort

□ Pseudocódigo (cria heap)

1. **BUILD-MAX-HEAP** (V)
2. V.tamanho-heap = V.tamanho
3. Para i = floor(V.tamanho/2) até 1
4. **MAX-HEAPIFY**(V, i)

BUILD-MAX-HEAP

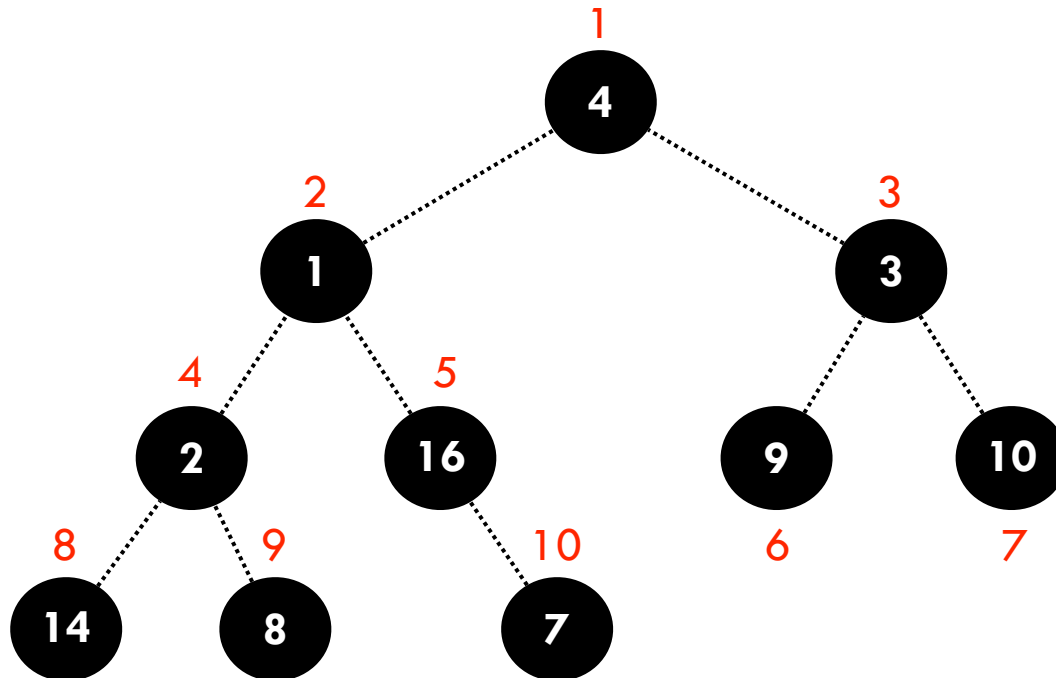
V =

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

BUILD-MAX-HEAP

V =

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7



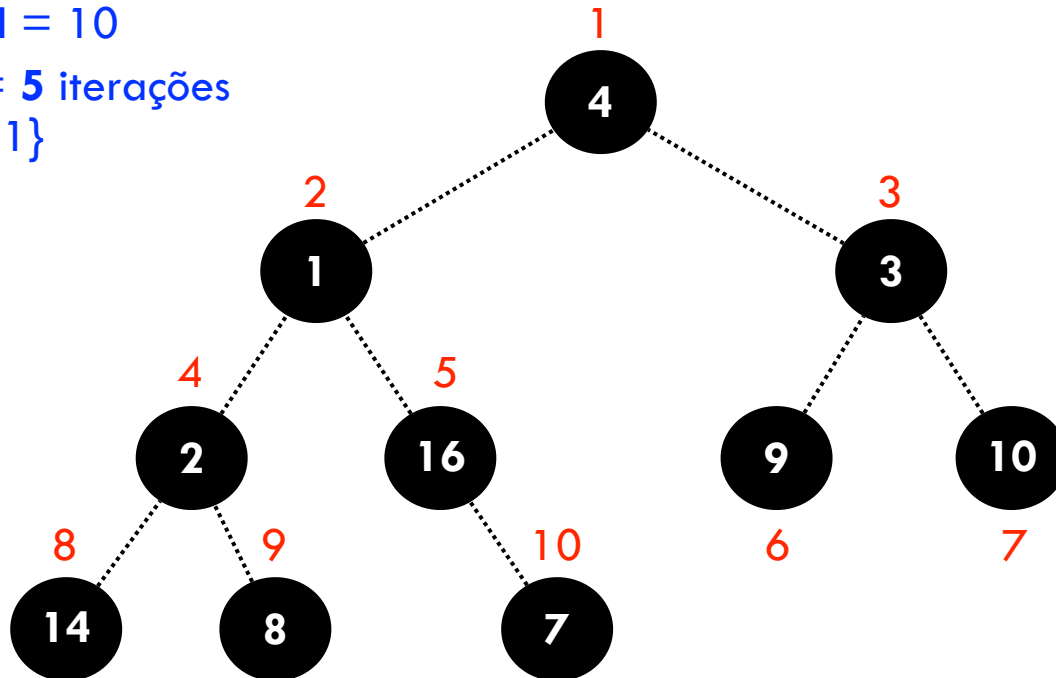
BUILD-MAX-HEAP

	1	2	3	4	5	6	7	8	9	10
V =	4	1	3	2	16	9	10	14	8	7

V.tamanho = **N** = 10

N/2 = 10/2 = **5** iterações

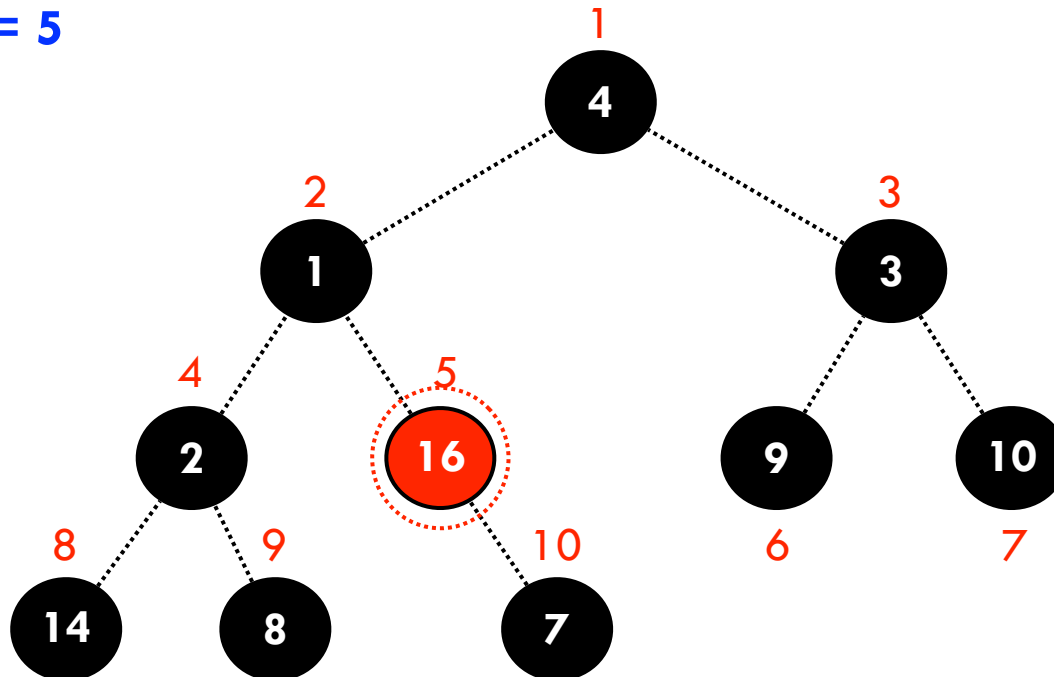
i = {5, 2, 3, 4, 1}



BUILD-MAX-HEAP

	1	2	3	4	5	6	7	8	9	10
V =	4	1	3	2	16	9	10	14	8	7

□ iteração $i = 5$

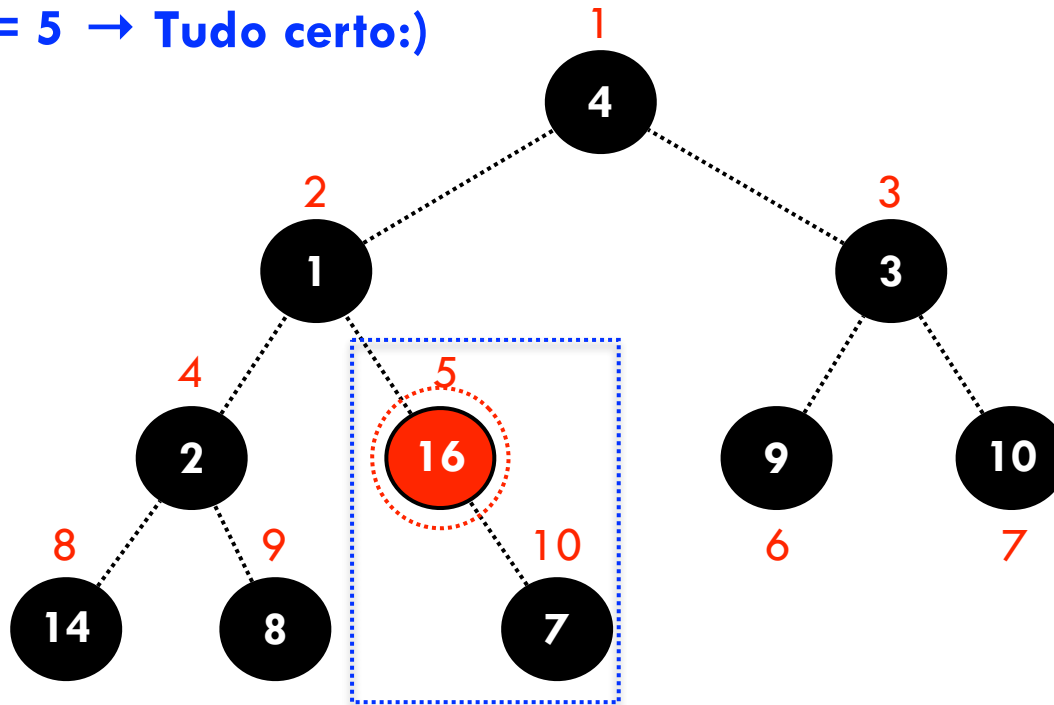


BUILD-MAX-HEAP

V =

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

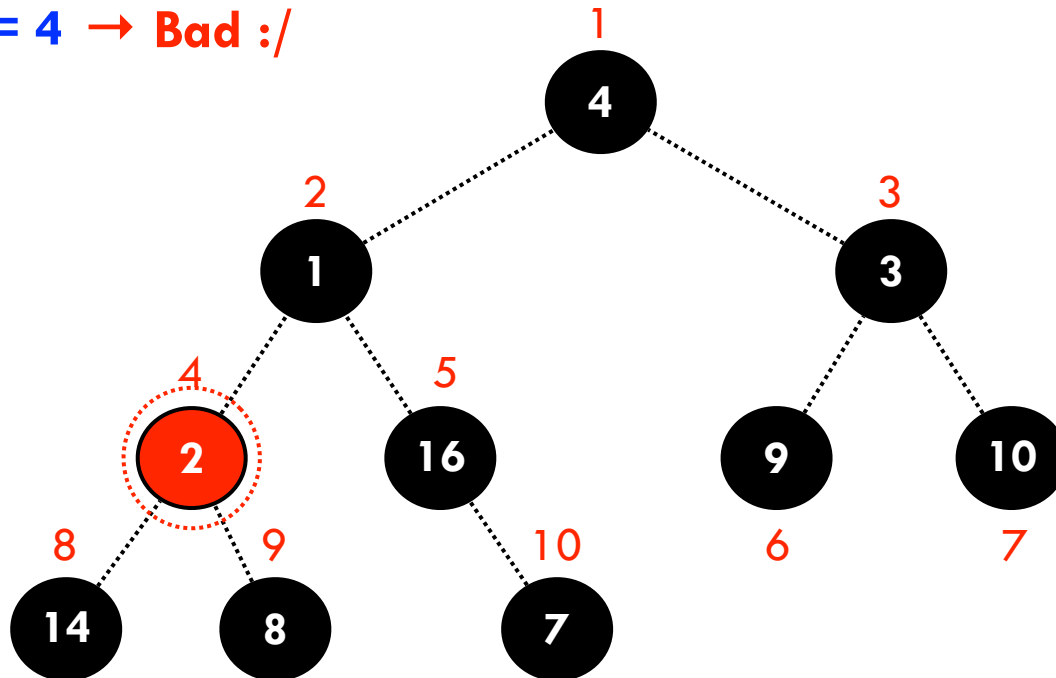
□ iteração $i = 5 \rightarrow$ Tudo certo:)



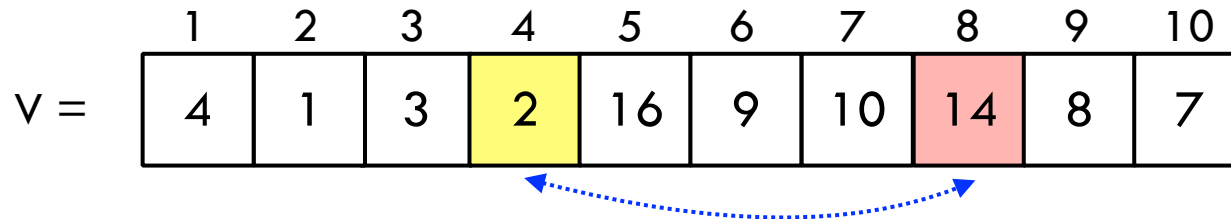
BUILD-MAX-HEAP

	1	2	3	4	5	6	7	8	9	10
V =	4	1	3	2	16	9	10	14	8	7

□ iteração $i = 4 \rightarrow$ **Bad** :/

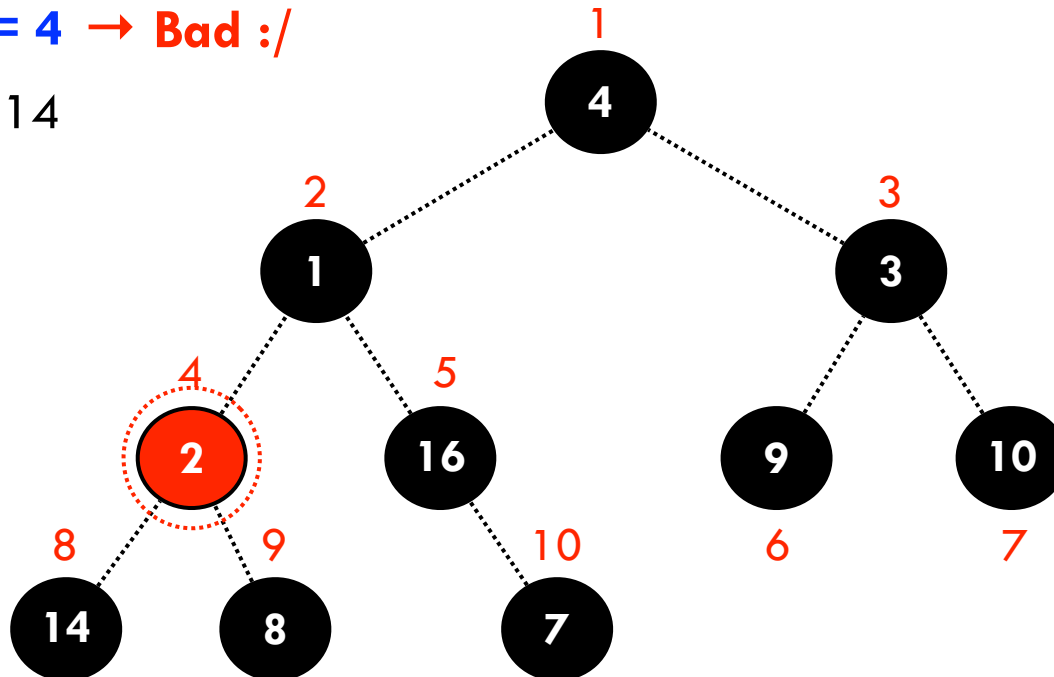


BUILD-MAX-HEAP

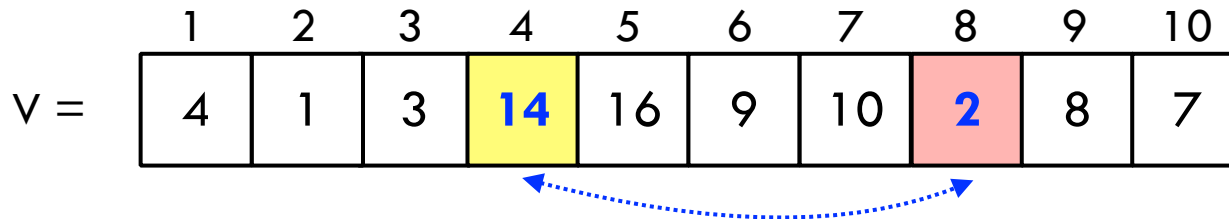


□ iteração $i = 4 \rightarrow$ **Bad** :/

□ Trocar 2 e 14

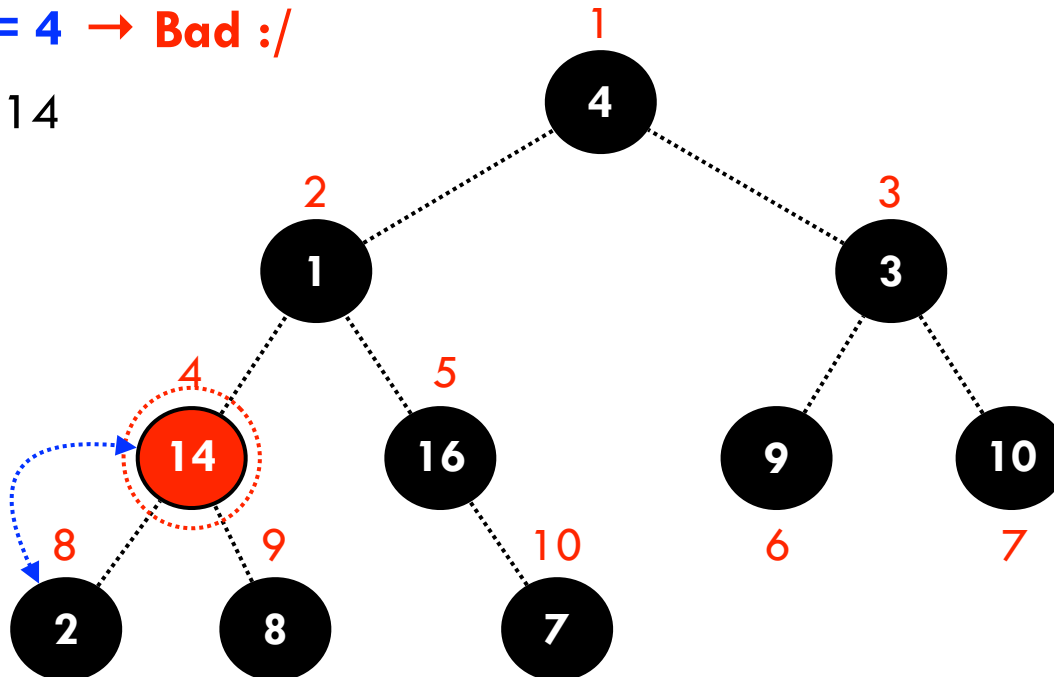


BUILD-MAX-HEAP



□ **iteração $i = 4 \rightarrow$ Bad :/**

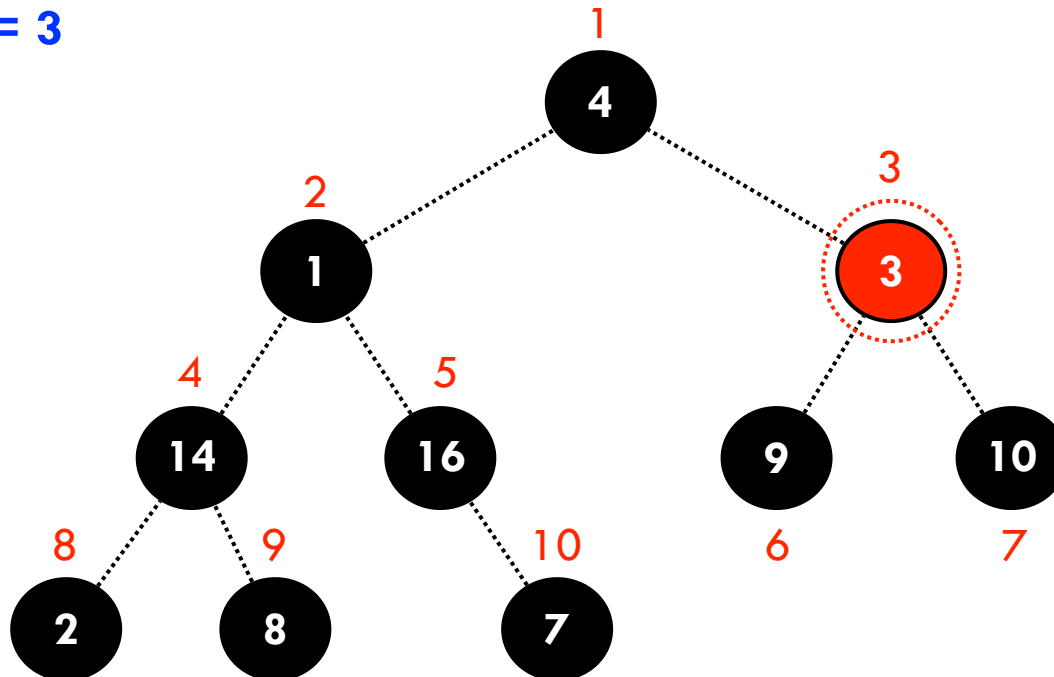
□ Trocar 2 e 14



BUILD-MAX-HEAP

	1	2	3	4	5	6	7	8	9	10
V =	4	1	3	14	16	9	10	2	8	7

□ iteração $i = 3$

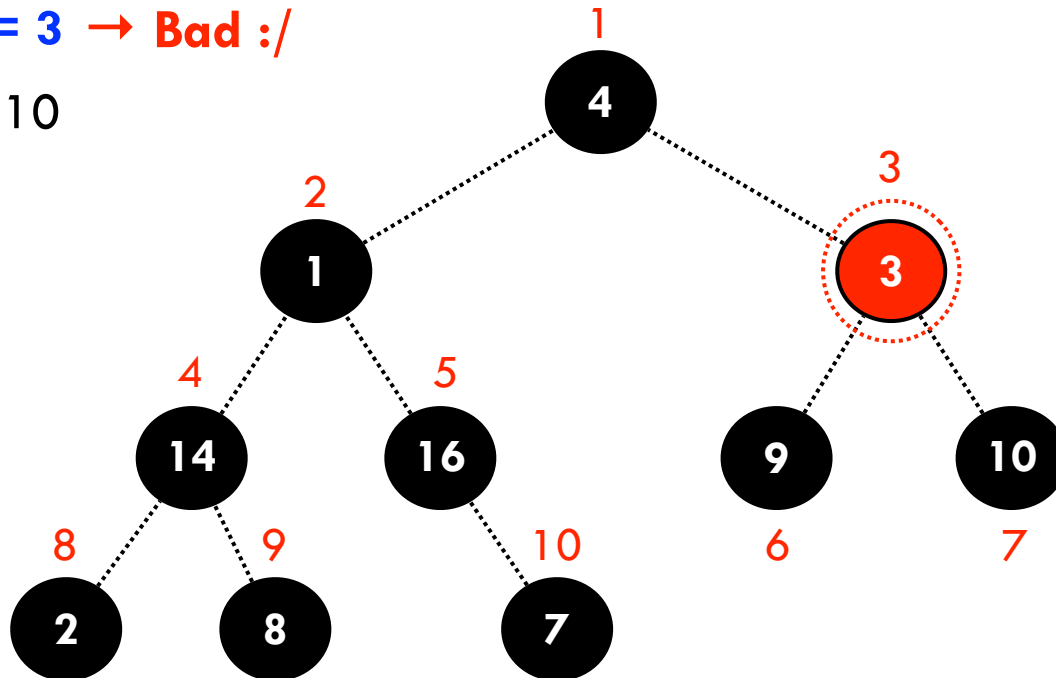


BUILD-MAX-HEAP

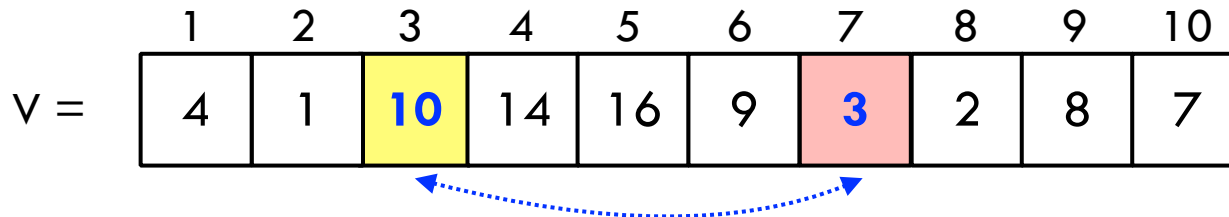
	1	2	3	4	5	6	7	8	9	10
V =	4	1	3	14	16	9	10	2	8	7

□ iteração $i = 3 \rightarrow$ **Bad** :/

□ Trocar 3 e 10

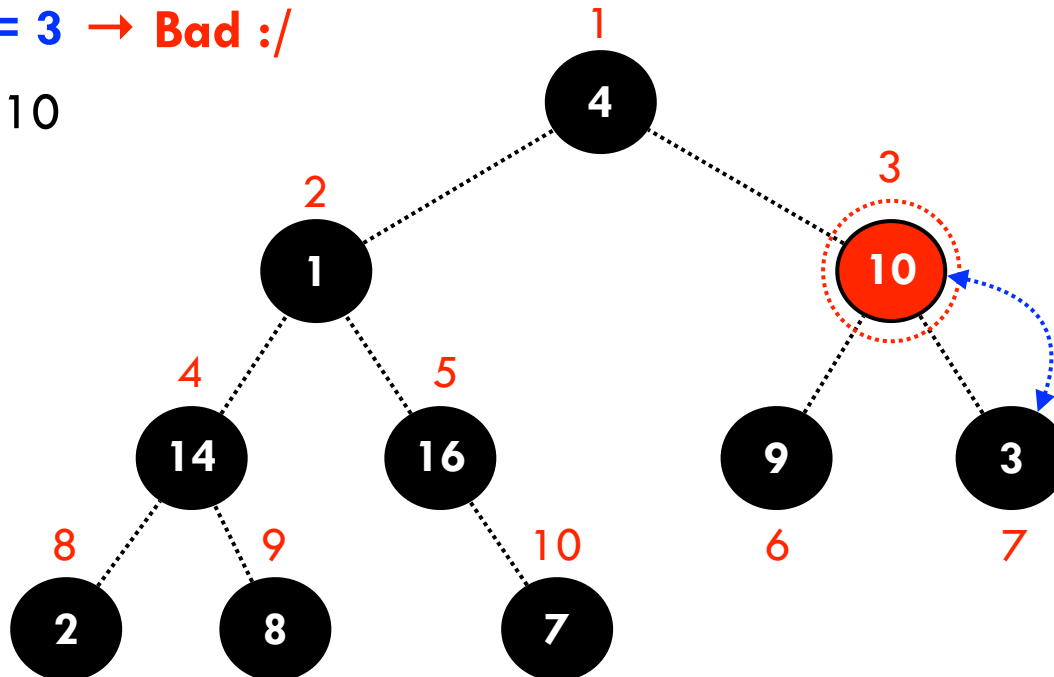


BUILD-MAX-HEAP



□ iteração $i = 3 \rightarrow$ **Bad** :/

□ Trocar 3 e 10

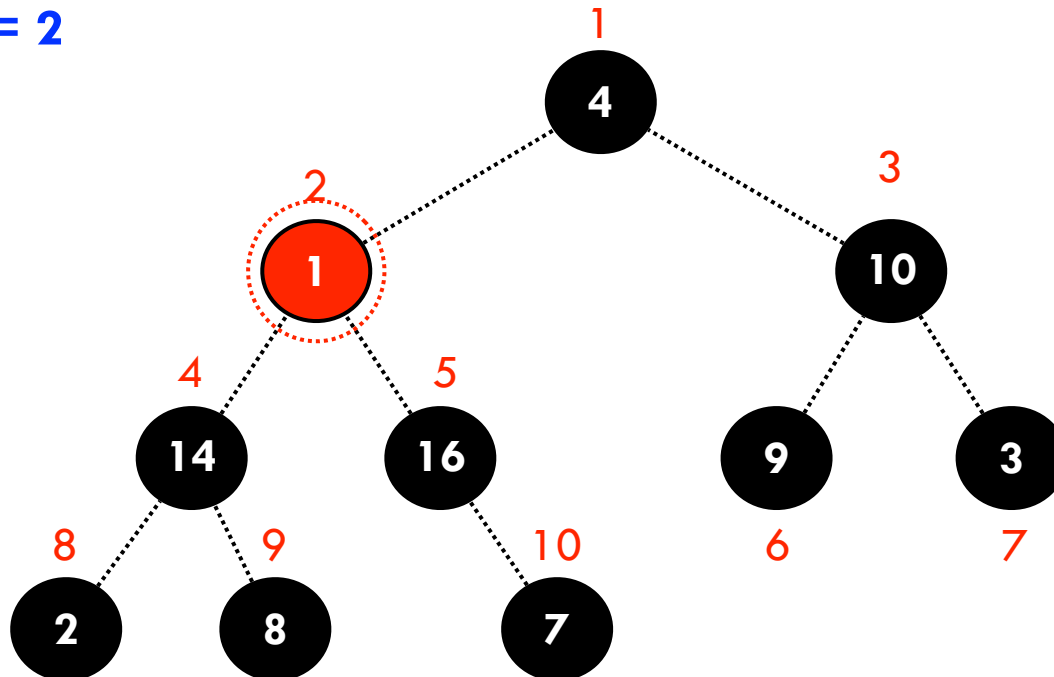


BUILD-MAX-HEAP

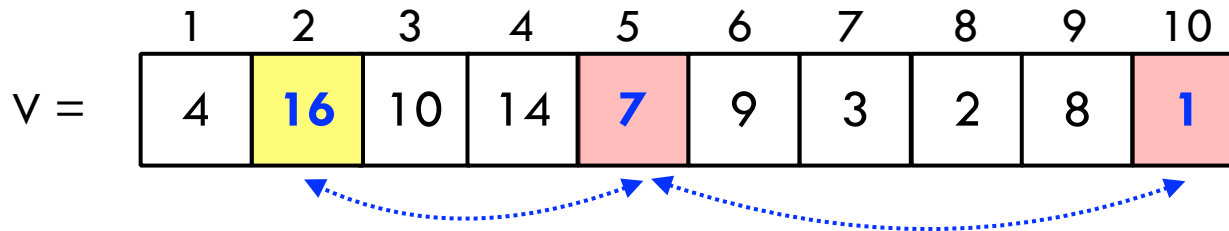
V =

1	2	3	4	5	6	7	8	9	10
4	1	10	14	16	9	3	2	8	7

□ iteração $i = 2$



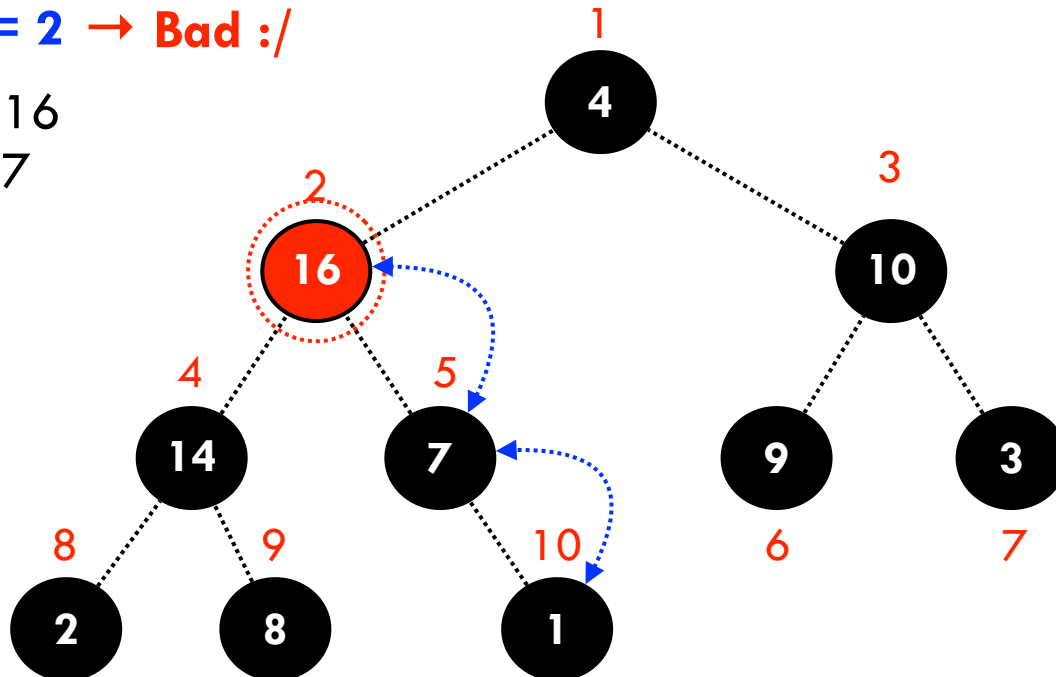
BUILD-MAX-HEAP



□ **iteração $i = 2 \rightarrow$ Bad :/**

□ Trocar 1 e 16

□ Trocar 1 e 7

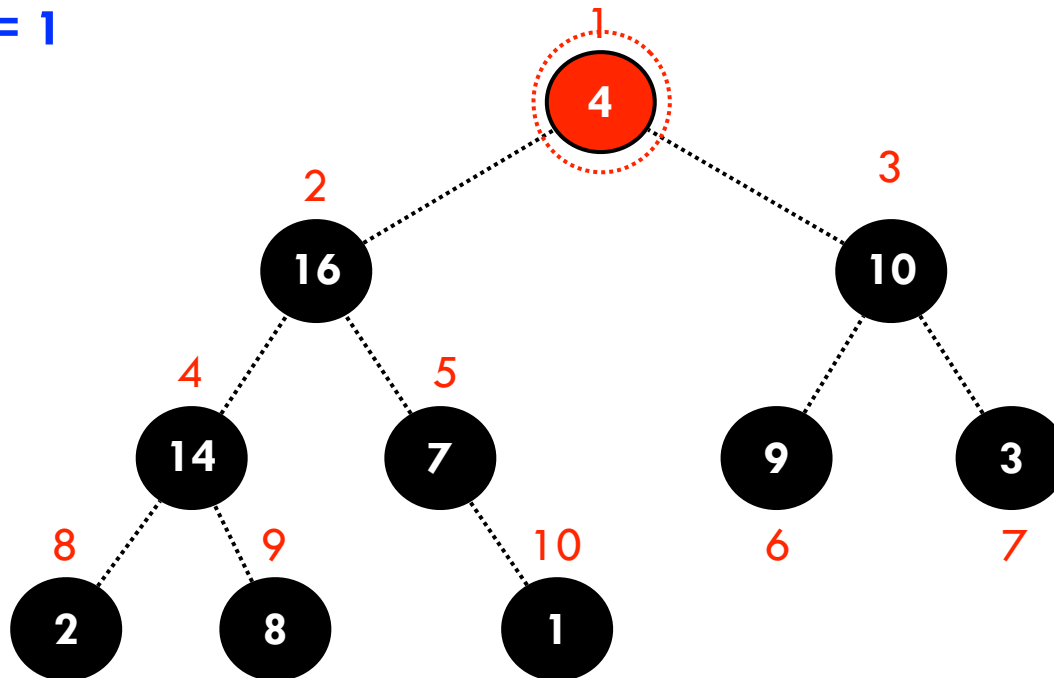


BUILD-MAX-HEAP

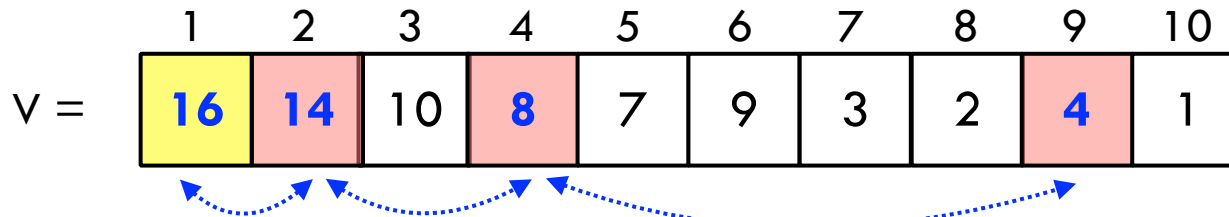
V =

1	2	3	4	5	6	7	8	9	10
4	16	10	14	7	9	3	2	8	1

□ iteração $i = 1$



BUILD-MAX-HEAP

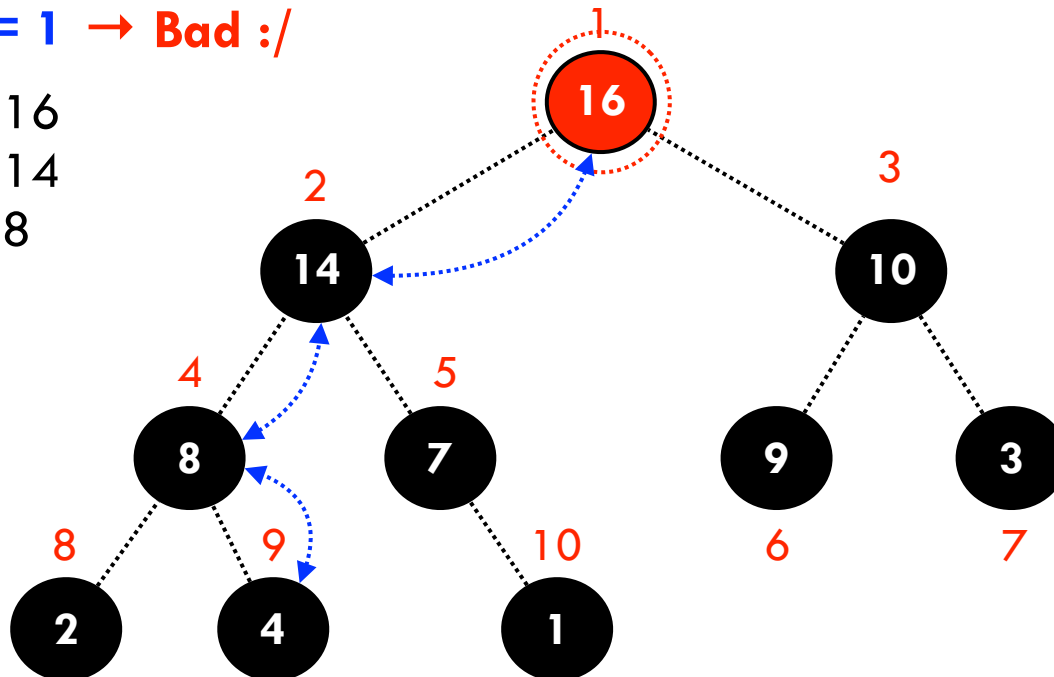


□ **iteração $i = 1 \rightarrow \text{Bad} :/$**

□ Trocar 4 e 16

□ Trocar 4 e 14

□ Trocar 4 e 8

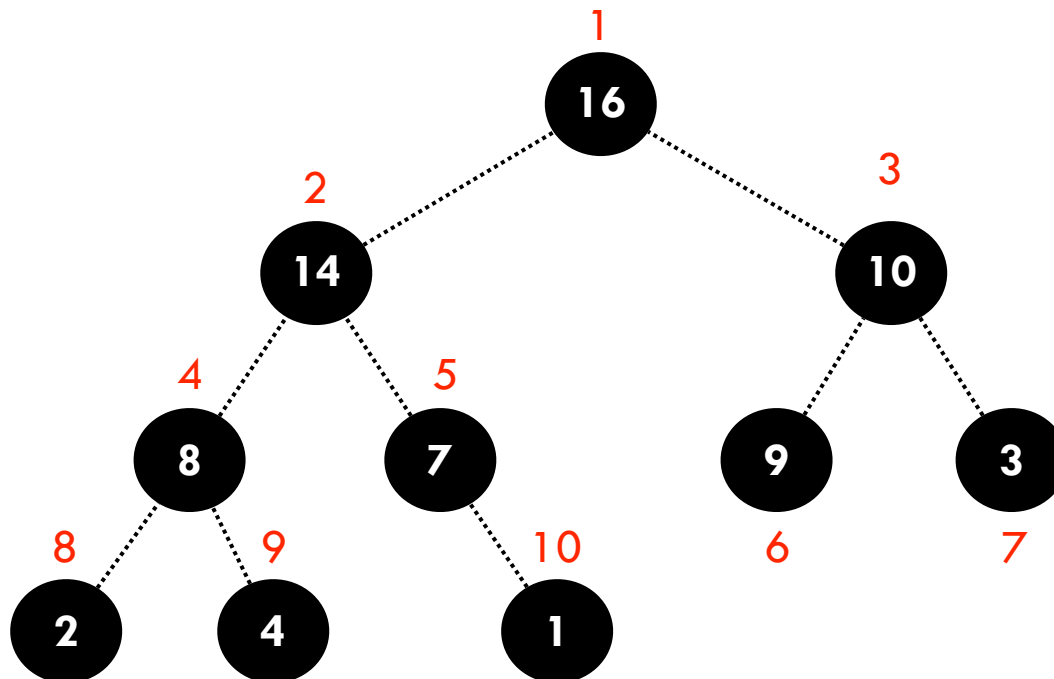


BUILD-MAX-HEAP

V =

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

□ Final!



Heap Sort

□ Pseudocódigo (função principal)

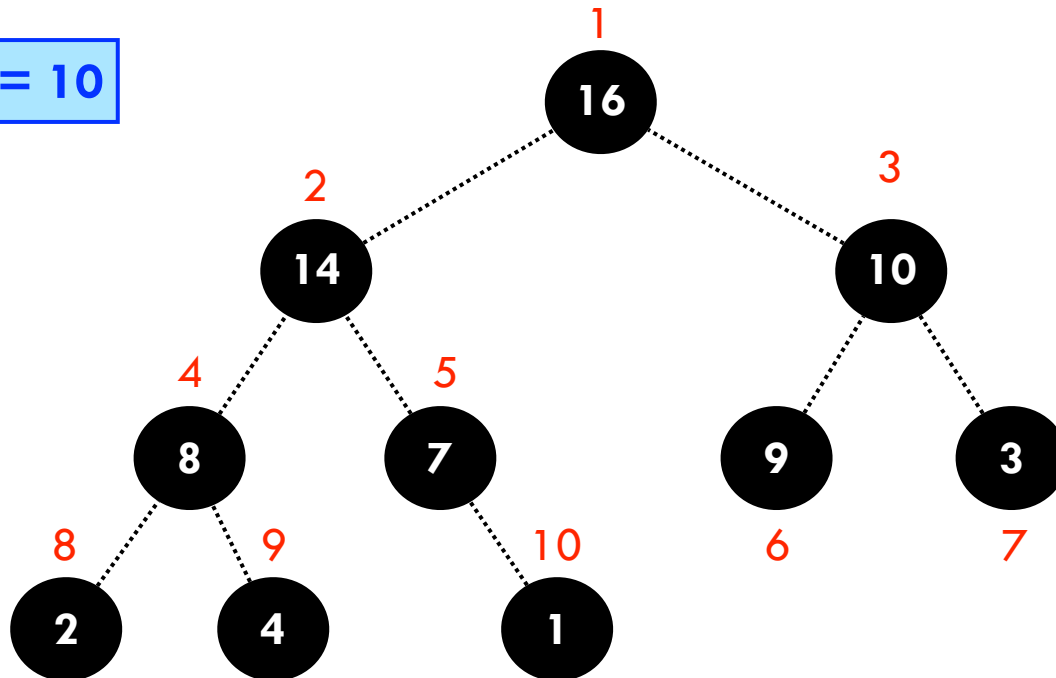
1. **HEAPSORT** (V)
2. **BUILD-MAX-HEAP**(A)
3. **Para** $i = V.tamanho$ até 2
4. Troca($V[1]$, $V[i]$)
5. $V.tamanho\text{-}heap = V.tamanho\text{-}heap - 1$
6. **MAX-HEAPIFY**(V, 1)

HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

□ iteração $i = 10$



HEAP SORT

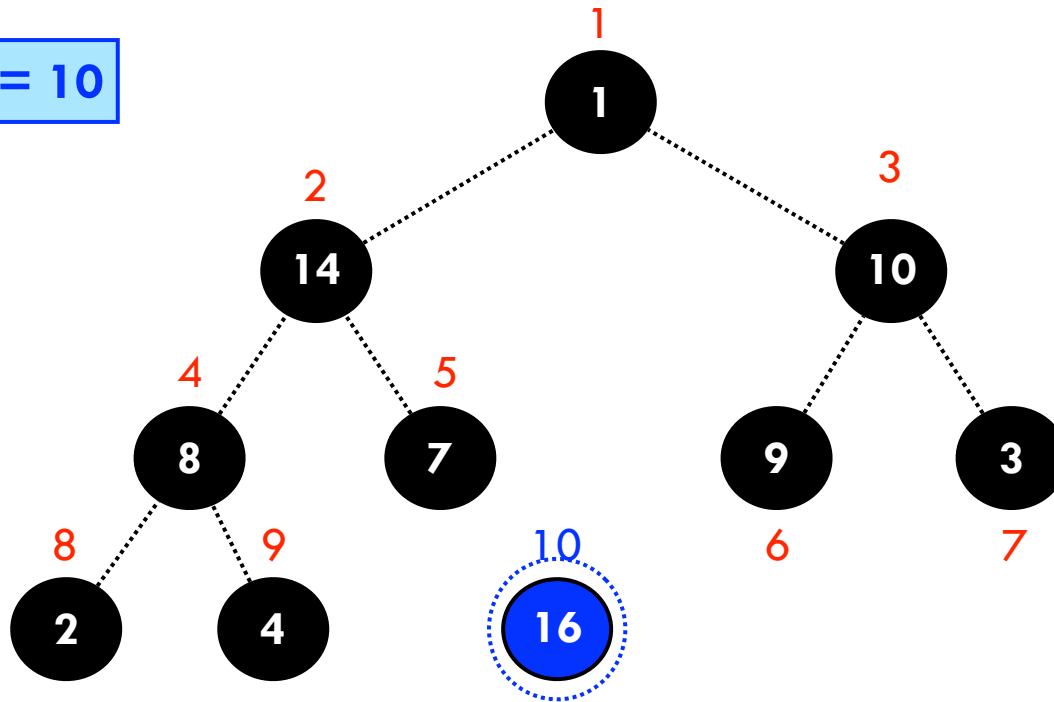
V =

1	2	3	4	5	6	7	8	9	10
1	14	10	8	7	9	3	2	4	16

MAX-HEAPIFY



□ iteração i = 10



HEAP SORT

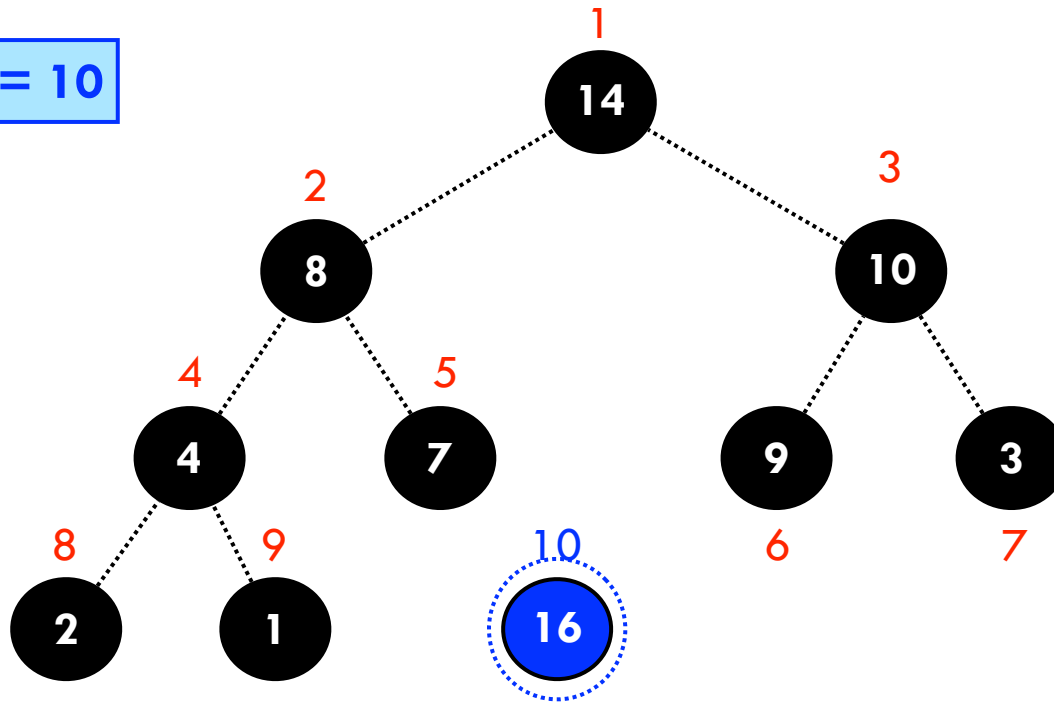
V =

1	2	3	4	5	6	7	8	9	10
14	8	10	4	7	9	3	2	1	16

MAX-HEAPIFY



□ iteração i = 10

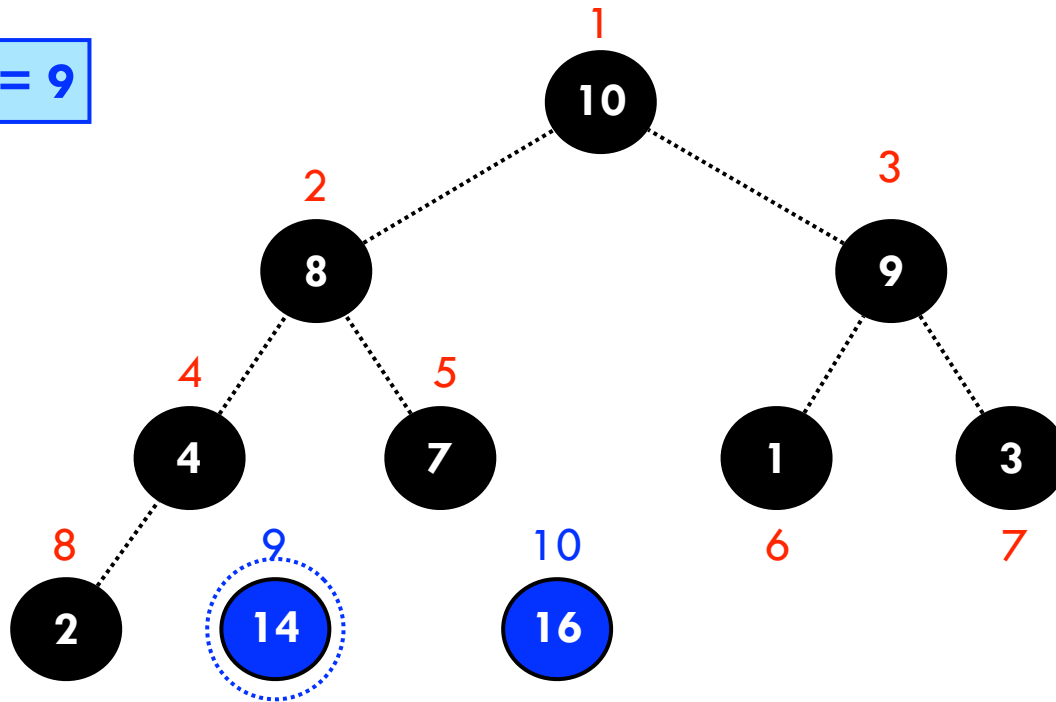


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
10	8	9	4	7	1	3	2	14	16

□ iteração $i = 9$

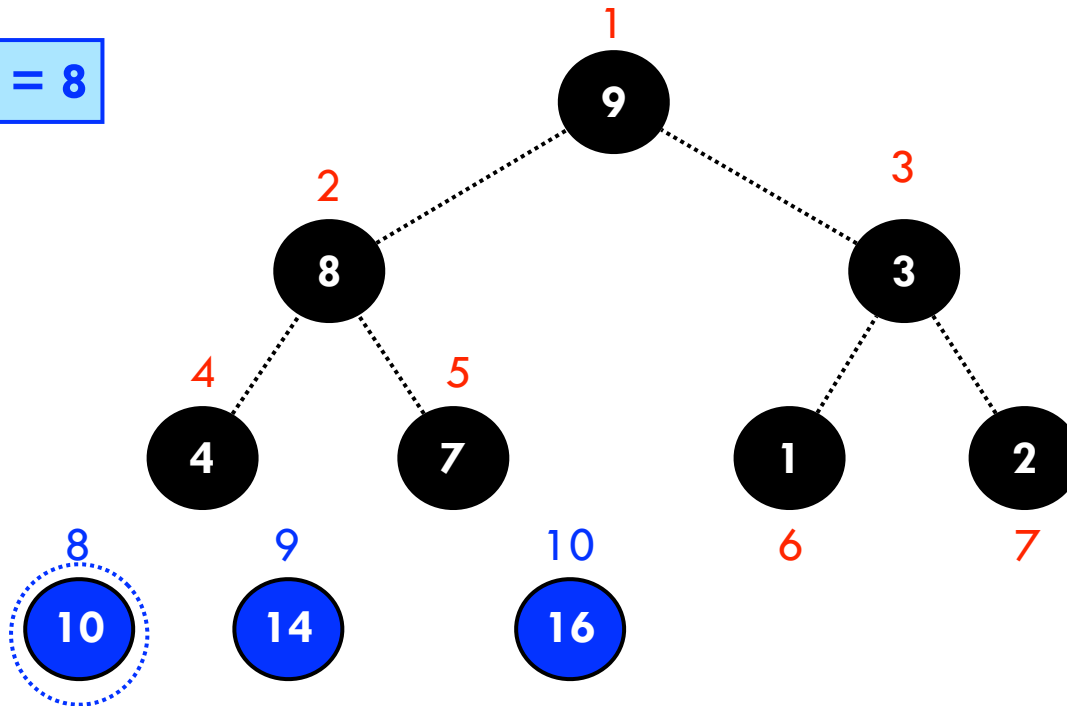


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
9	8	3	4	7	1	2	10	14	16

□ iteração $i = 8$

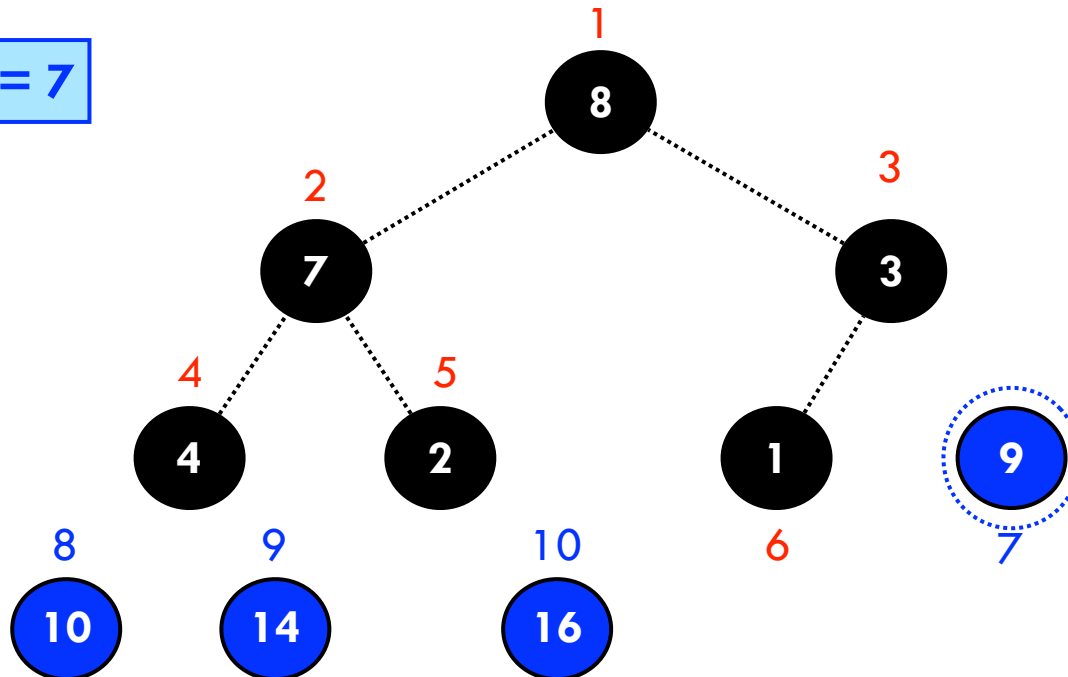


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
9	8	3	4	7	1	9	10	14	16

□ iteração $i = 7$

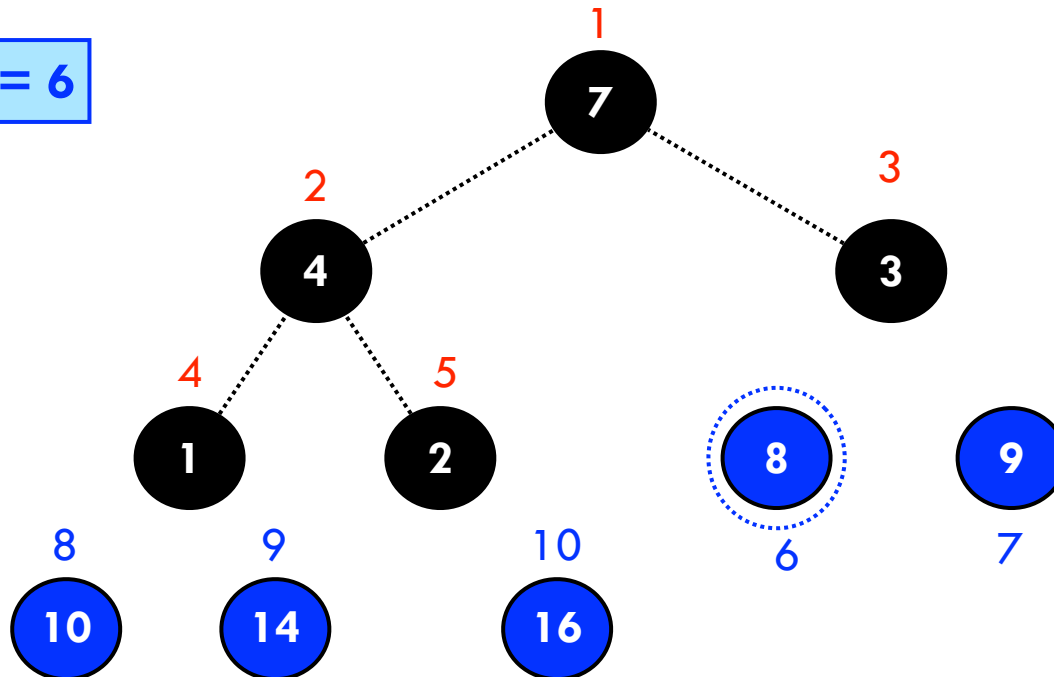


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
7	4	3	1	2	8	9	10	14	16

□ iteração $i = 6$

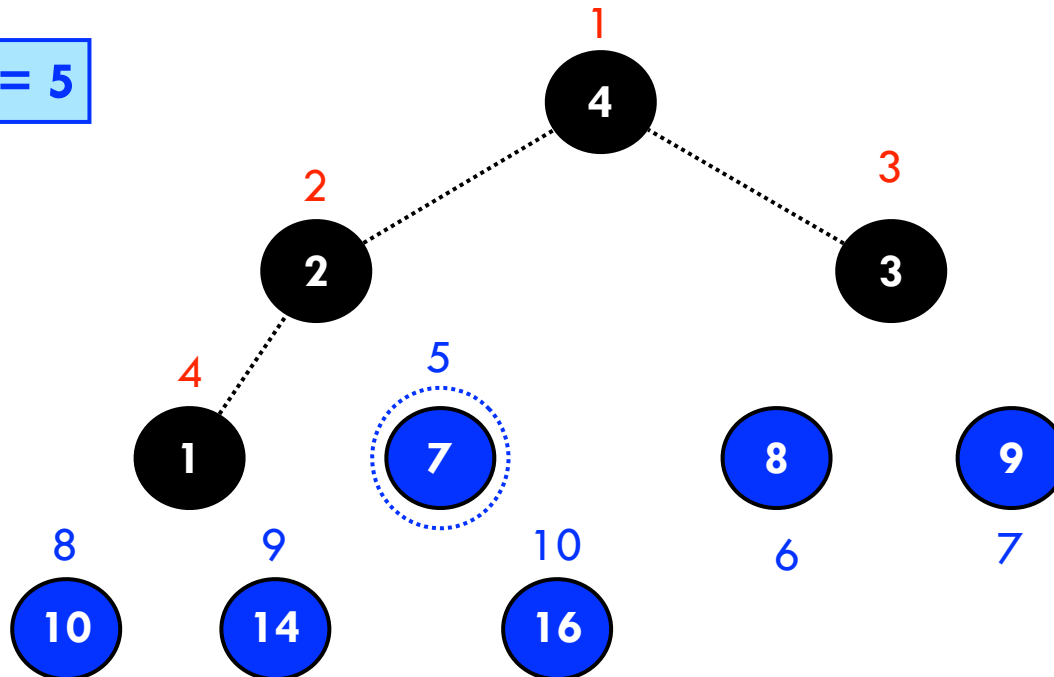


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
4	2	3	1	7	8	9	10	14	16

□ iteração $i = 5$

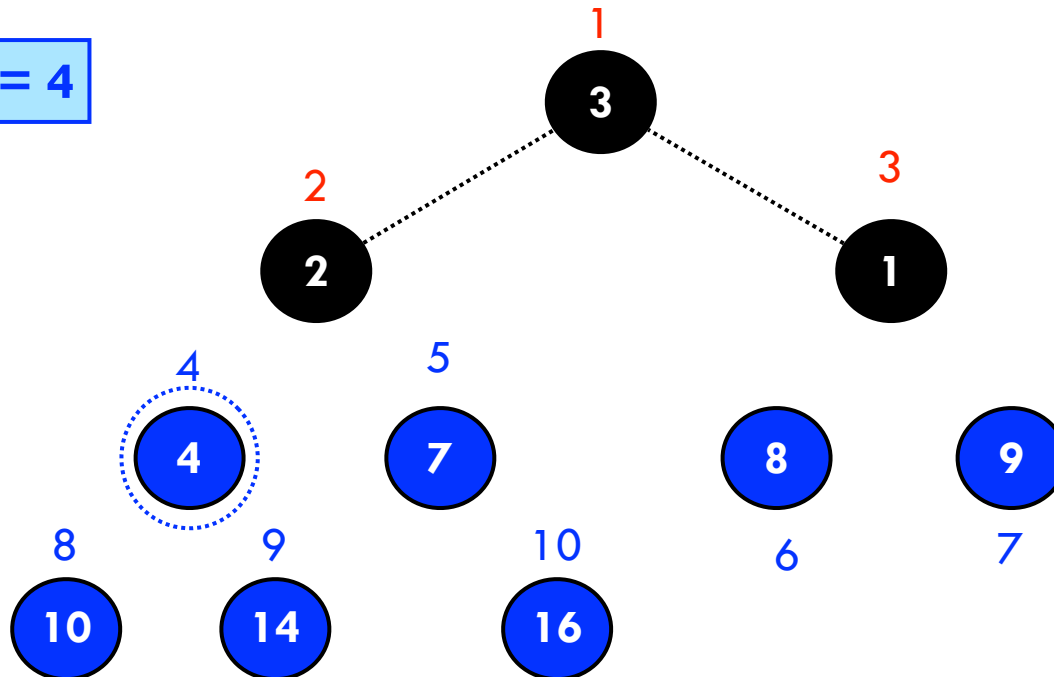


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
3	2	1	4	7	8	9	10	14	16

□ iteração $i = 4$

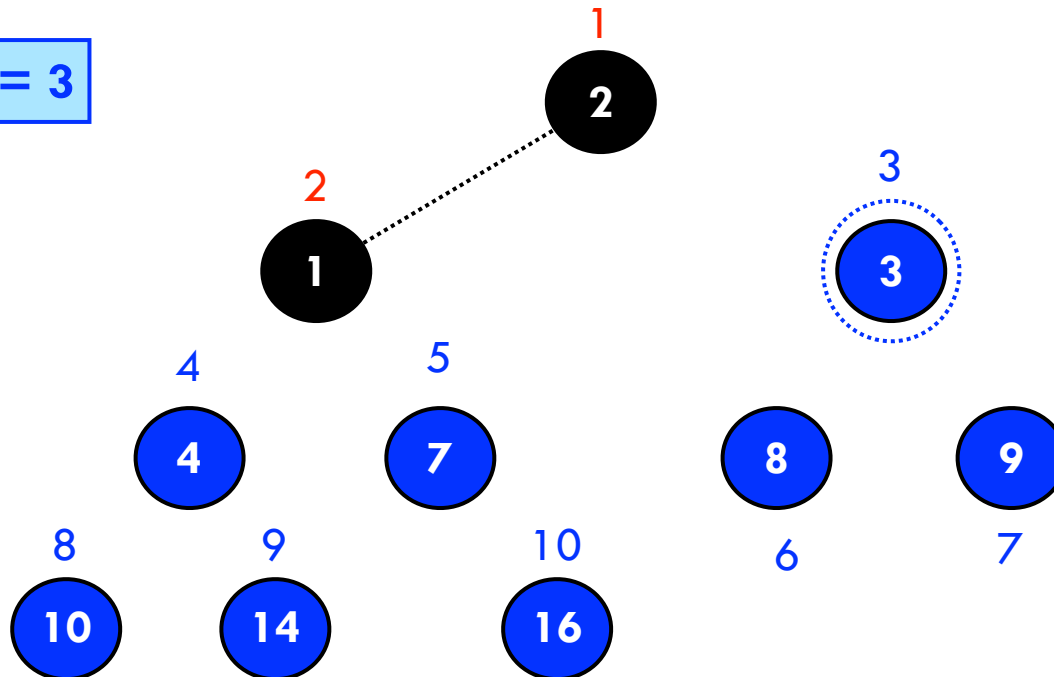


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
2	1	3	4	7	8	9	10	14	16

□ iteração $i = 3$

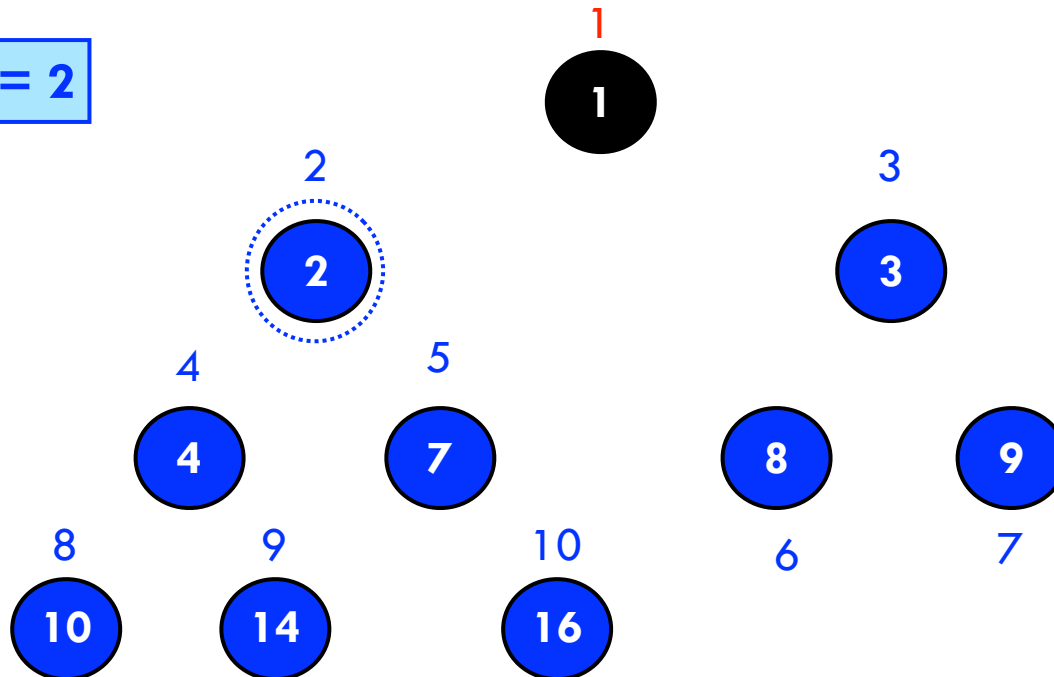


HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
1	2	3	4	7	8	9	10	14	16

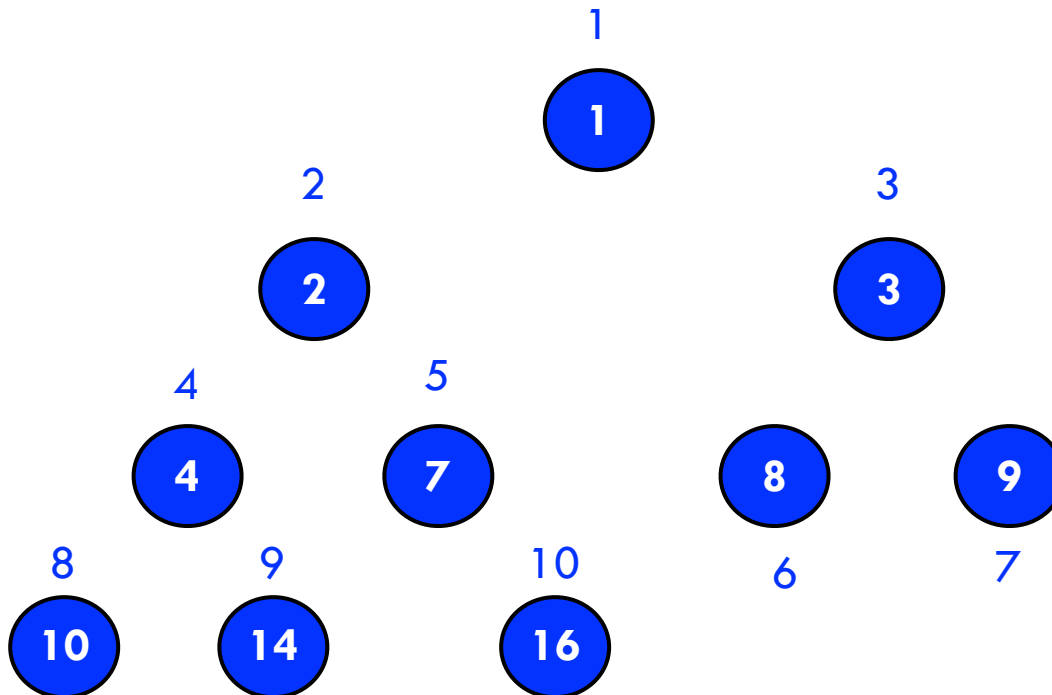
□ iteração $i = 2$



HEAP SORT

V =

1	2	3	4	5	6	7	8	9	10
1	2	3	4	7	8	9	10	14	16



Roteiro

- 1 Introdução
- 2 Heaps
- 3 Heap Sort
- 4 Exemplo
- 5 Exercícios
- 6 Referências

Exemplo

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

vetor não ordenado

Roteiro



- 1 Introdução
- 2 Heaps
- 3 Heap Sort
- 4 Exemplo
- 5 Exercícios
- 6 Referências

Exercícios



HANDS ON :)))

Exercícios



1) Reuna-se com seu grupo e execute o teste de mesa (simulação) do algoritmo **Heap Sort** para a sua sequência de números aleatórios, definida na planilha de grupos da disciplina.

Exercícios

2) Implemente o **heapSort** em Python considerando as seguintes funções

```
/* função principal */
```

```
def heapSort(array);
```

```
/* constrói heap de máximo a partir do vetor */
```

```
def buildMaxHeap(array);
```

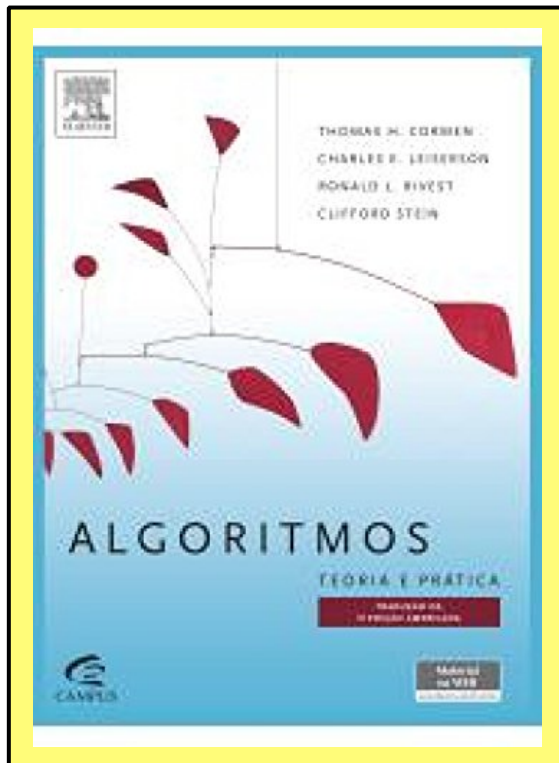
```
/* reconstrói o heap, desconsiderando o elemento já ordenado */
```

```
def maxHeapify(array, i, heapsize);
```

Roteiro

- 1 Introdução
- 2 Heaps
- 3 Heap Sort
- 4 Exemplo
- 5 Exercícios
- 6 Referências

Referências sugeridas

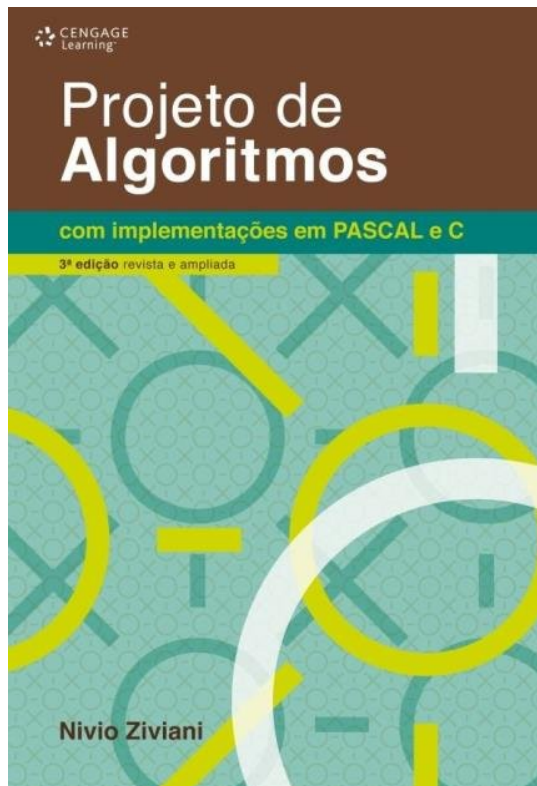


[Cormen et al, 2018]

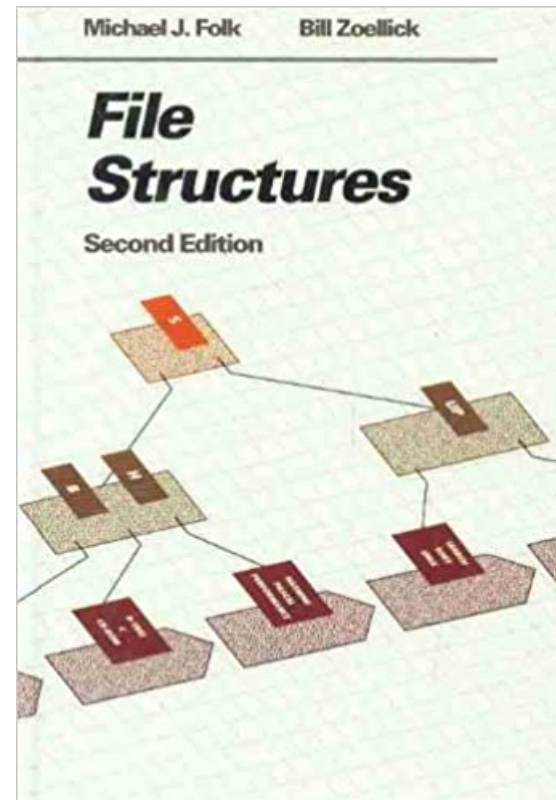


[Drozdek, 2017]

Referências sugeridas



[Ziviani, 2010]



[Folk & Zoellick, 1992]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br