

Projeto e Análise de Algoritmos

Fila de prioridade

Lehilton Pedrosa

Primeiro Semestre de 2023

Selection Sort

Selecionando o próximo menor

Vamos ordenar usando ideia diferente

- ▶ suponha que o subvetor $A[1 \dots i - 1]$ esteja ordenado
- ▶ também, suponha que $\max A[1 \dots i - 1] \leq \min A[i \dots n]$.
- ▶ substituimos a posição $A[i]$ pelo mínimo em $A[i \dots n]$

Antes de substituir:

1							i				n	
20	25	35	40	44	55	70	80	99	65	85		

Após substituir:

1							i				n	
20	25	35	40	44	55	65	80	99	70	85		

Selecionando o próximo menor

Vamos ordenar usando ideia diferente

- ▶ suponha que o subvetor $A[1 \dots i - 1]$ esteja ordenado
- ▶ também, suponha que $\max A[1 \dots i - 1] \leq \min A[i \dots n]$.
- ▶ substituímos a posição $A[i]$ pelo mínimo em $A[i \dots n]$

Antes de substituir:

1						i			n	
20	25	35	40	44	55	70	80	99	65	85

Após substituir:

1						i			n	
20	25	35	40	44	55	65	80	99	70	85

Selecionando o próximo menor

Vamos ordenar usando ideia diferente

- ▶ suponha que o subvetor $A[1 \dots i - 1]$ esteja ordenado
- ▶ também, suponha que $\max A[1 \dots i - 1] \leq \min A[i \dots n]$.
- ▶ substituímos a posição $A[i]$ pelo mínimo em $A[i \dots n]$

Antes de substituir:

1						i			n	
20	25	35	40	44	55	70	80	99	65	85

Após substituir:

1						i			n	
20	25	35	40	44	55	65	80	99	70	85

Selecionando o próximo menor

Vamos ordenar usando ideia diferente

- ▶ suponha que o subvetor $A[1 \dots i - 1]$ esteja ordenado
- ▶ também, suponha que $\max A[1 \dots i - 1] \leq \min A[i \dots n]$.
- ▶ substituimos a posição $A[i]$ pelo mínimo em $A[i \dots n]$

Antes de substituir:

1						i			n	
20	25	35	40	44	55	70	80	99	65	85

Após substituir:

1						i			n	
20	25	35	40	44	55	65	80	99	70	85

Selecionando o próximo menor

Vamos ordenar usando ideia diferente

- ▶ suponha que o subvetor $A[1 \dots i - 1]$ esteja ordenado
- ▶ também, suponha que $\max A[1 \dots i - 1] \leq \min A[i \dots n]$.
- ▶ substituímos a posição $A[i]$ pelo mínimo em $A[i \dots n]$

Antes de substituir:

1							i				n	
20	25	35	40	44	55	70	80	99	65	85		

Após substituir:

1							i				n	
20	25	35	40	44	55	65	80	99	70	85		

Selecionando o próximo menor

Vamos ordenar usando ideia diferente

- ▶ suponha que o subvetor $A[1 \dots i - 1]$ esteja ordenado
- ▶ também, suponha que $\max A[1 \dots i - 1] \leq \min A[i \dots n]$.
- ▶ substituímos a posição $A[i]$ pelo mínimo em $A[i \dots n]$

Antes de substituir:

1							i				n	
20	25	35	40	44	55	70	80	99	65	85		

Após substituir:

1							i				n	
20	25	35	40	44	55	65	80	99	70	85		

Pseudocódigo de SELECTION-SORT

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow 1$  até  $n - 1$ 
2       $min \leftarrow i$ 
3      para  $j \leftarrow i + 1$  até  $n$ 
4          se  $A[j] < A[min]$ 
5               $min \leftarrow j$ 
6       $A[i] \leftrightarrow A[min]$ 
```

Invariante

Ao início de cada iteração:

1. $A[1 \dots i-1]$ está ordenado,
2. $A[1 \dots i-1] \leq A[i \dots n]$.

Pseudocódigo de SELECTION-SORT

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow 1$  até  $n - 1$ 
2       $min \leftarrow i$ 
3      para  $j \leftarrow i + 1$  até  $n$ 
4          se  $A[j] < A[min]$ 
5               $min \leftarrow j$ 
6       $A[i] \leftrightarrow A[min]$ 
```

Invariante

Ao início de cada iteração:

1. $A[1 \dots i - 1]$ está ordenado,
2. $A[1 \dots i - 1] \leq A[i \dots n]$.

Pseudocódigo de SELECTION-SORT

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow 1$  até  $n - 1$ 
2       $min \leftarrow i$ 
3      para  $j \leftarrow i + 1$  até  $n$ 
4          se  $A[j] < A[min]$ 
5               $min \leftarrow j$ 
6       $A[i] \leftrightarrow A[min]$ 
```

Invariante

Ao início de cada iteração:

1. $A[1 \dots i - 1]$ está ordenado,
2. $A[1 \dots i - 1] \leq A[i \dots n]$.

Pseudocódigo de SELECTION-SORT

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow 1$  até  $n - 1$ 
2       $min \leftarrow i$ 
3      para  $j \leftarrow i + 1$  até  $n$ 
4          se  $A[j] < A[min]$ 
5               $min \leftarrow j$ 
6       $A[i] \leftrightarrow A[min]$ 
```

Invariante

Ao início de cada iteração:

1. $A[1 \dots i - 1]$ está ordenado,
2. $A[1 \dots i - 1] \leq A[i \dots n]$.

Complexidade de SELECTION-SORT

	SELECTION-SORT(A, n)	Tempo
1	para $i \leftarrow 1$ até $n - 1$	?
2	$min \leftarrow i$	?
3	para $j \leftarrow i + 1$ até n	?
4	se $A[j] < A[min]$	?
5	$min \leftarrow j$	?
6	$A[i] \leftrightarrow A[min]$	?

Consumo de tempo no pior caso? $\Theta(n^2)$

E no melhor caso?

Complexidade de SELECTION-SORT

	SELECTION-SORT(A, n)	Tempo
1	para $i \leftarrow 1$ até $n - 1$	$\Theta(n)$
2	$min \leftarrow i$	$\Theta(n)$
3	para $j \leftarrow i + 1$ até n	$\Theta(n^2)$
4	se $A[j] < A[min]$	$\Theta(n^2)$
5	$min \leftarrow j$	$\Theta(n^2)$
6	$A[i] \leftrightarrow A[min]$	$\Theta(n)$

Consumo de tempo no pior caso? $\Theta(n^2)$

E no melhor caso?

Complexidade de SELECTION-SORT

	SELECTION-SORT (A, n)	Tempo
1	para $i \leftarrow 1$ até $n - 1$	$\Theta(n)$
2	$min \leftarrow i$	$\Theta(n)$
3	para $j \leftarrow i + 1$ até n	$\Theta(n^2)$
4	se $A[j] < A[min]$	$\Theta(n^2)$
5	$min \leftarrow j$	$\Theta(n^2)$
6	$A[i] \leftrightarrow A[min]$	$\Theta(n)$

Consumo de tempo no pior caso? $\Theta(n^2)$

E no melhor caso?

Complexidade de SELECTION-SORT

SELECTION-SORT(A, n)		Tempo
1	para $i \leftarrow 1$ até $n - 1$	$\Theta(n)$
2	$min \leftarrow i$	$\Theta(n)$
3	para $j \leftarrow i + 1$ até n	$\Theta(n^2)$
4	se $A[j] < A[min]$	$\Theta(n^2)$
5	$min \leftarrow j$	$\Theta(n^2)$
6	$A[i] \leftrightarrow A[min]$	$\Theta(n)$

Consumo de tempo no pior caso? $\Theta(n^2)$

E no melhor caso?

Complexidade de SELECTION-SORT

SELECTION-SORT(A, n)		Tempo
1	para $i \leftarrow 1$ até $n - 1$	$\Theta(n)$
2	$min \leftarrow i$	$\Theta(n)$
3	para $j \leftarrow i + 1$ até n	$\Theta(n^2)$
4	se $A[j] < A[min]$	$\Theta(n^2)$
5	$min \leftarrow j$	$\Theta(n^2)$
6	$A[i] \leftrightarrow A[min]$	$\Theta(n)$

Consumo de tempo no pior caso? $\Theta(n^2)$

E no melhor caso?

Uma versão alternativa

Podemos reescrever esse algoritmo

- ▶ ordenamos a partir do final
- ▶ selecionamos o maior remanescente
- ▶ refatoramos com uma sub-rotina MAXIMUM

Uma versão alternativa

Podemos reescrever esse algoritmo

- ▶ ordenamos a partir do final
- ▶ selecionamos o maior remanescente
- ▶ refatoramos com uma sub-rotina MAXIMUM

Uma versão alternativa

Podemos reescrever esse algoritmo

- ▶ ordenamos a partir do final
- ▶ selecionamos o **maior** remanescente
- ▶ refatoramos com uma sub-rotina `MAXIMUM`

Uma versão alternativa

Podemos reescrever esse algoritmo

- ▶ ordenamos a partir do final
- ▶ selecionamos o **maior** remanescente
- ▶ refatoramos com uma sub-rotina **MAXIMUM**

Uma versão alternativa

Podemos reescrever esse algoritmo

- ▶ ordenamos a partir do final
- ▶ selecionamos o **maior** remanescente
- ▶ refatoramos com uma sub-rotina **MAXIMUM**

```
SELECTION-SORT( $A, n$ )  
1  para  $i \leftarrow n$  decrescendo até 2  
2       $max \leftarrow \text{MAXIMUM}(A, i)$   
3       $A[i] \leftrightarrow A[max]$ 
```

Reverendo a complexidade

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow n$  decrescendo até 2  
2       $max \leftarrow \text{MAXIMUM}(A, i)$   
3       $A[i] \leftrightarrow A[max]$ 
```

- ▶ suponha que $\text{MAXIMUM}(A, i)$ leva tempo $O(t(i))$
- ▶ então o tempo total é

$$T(n) = \sum_{i=2}^n O(t(i)) \leq \sum_{i=2}^n O(t(n)) = O(n \cdot t(n))$$

- ▶ vamos tentar otimizar a rotina $\text{MAXIMUM}(A, i)$

Reverendo a complexidade

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow n$  decrescendo até 2  
2       $max \leftarrow \text{MAXIMUM}(A, i)$   
3       $A[i] \leftrightarrow A[max]$ 
```

- ▶ suponha que $\text{MAXIMUM}(A, i)$ leva tempo $O(t(i))$
- ▶ então o tempo total é

$$T(n) = \sum_{i=2}^n O(t(i)) \leq \sum_{i=2}^n O(t(n)) = O(n \cdot t(n))$$

- ▶ vamos tentar otimizar a rotina $\text{MAXIMUM}(A, i)$

Reverendo a complexidade

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow n$  decrescendo até 2  
2       $max \leftarrow \text{MAXIMUM}(A, i)$   
3       $A[i] \leftrightarrow A[max]$ 
```

- ▶ suponha que $\text{MAXIMUM}(A, i)$ leva tempo $O(t(i))$
- ▶ então o tempo total é

$$T(n) = \sum_{i=2}^n O(t(i)) \leq \sum_{i=2}^n O(t(n)) = O(n \cdot t(n))$$

- ▶ vamos tentar otimizar a rotina $\text{MAXIMUM}(A, i)$

Reverendo a complexidade

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow n$  decrescendo até 2  
2       $max \leftarrow \text{MAXIMUM}(A, i)$   
3       $A[i] \leftrightarrow A[max]$ 
```

- ▶ suponha que $\text{MAXIMUM}(A, i)$ leva tempo $O(t(i))$
- ▶ então o tempo total é

$$T(n) = \sum_{i=2}^n O(t(i)) \leq \sum_{i=2}^n O(t(n)) = O(n \cdot t(n))$$

- ▶ vamos tentar otimizar a rotina $\text{MAXIMUM}(A, i)$

Reverendo a complexidade

SELECTION-SORT(A, n)

```
1  para  $i \leftarrow n$  decrescendo até 2  
2       $max \leftarrow \text{MAXIMUM}(A, i)$   
3       $A[i] \leftrightarrow A[max]$ 
```

- ▶ suponha que $\text{MAXIMUM}(A, i)$ leva tempo $O(t(i))$
- ▶ então o tempo total é

$$T(n) = \sum_{i=2}^n O(t(i)) \leq \sum_{i=2}^n O(t(n)) = O(n \cdot t(n))$$

- ▶ vamos tentar otimizar a rotina $\text{MAXIMUM}(A, i)$

Heapsort

O algoritmo HEAP-SORT

Vamos estudar a **fila de prioridade**

- ▶ é uma estrutura de dados também chamada de max-heap
- ▶ implementa **MAXIMUM** com tempo $O(\log n)$
- ▶ usando um heap, podemos ordenar em $O(n \log n)$
- ▶ esse algoritmo de ordenação é chamado de **heapsort**

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i

- ▶ o filho esquerdo é $2i$
- ▶ o filho direito é $2i + 1$

- ▶ Pais

- ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
- ▶ o nó 1 não tem pai.

- ▶ Folhas

- ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
- ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

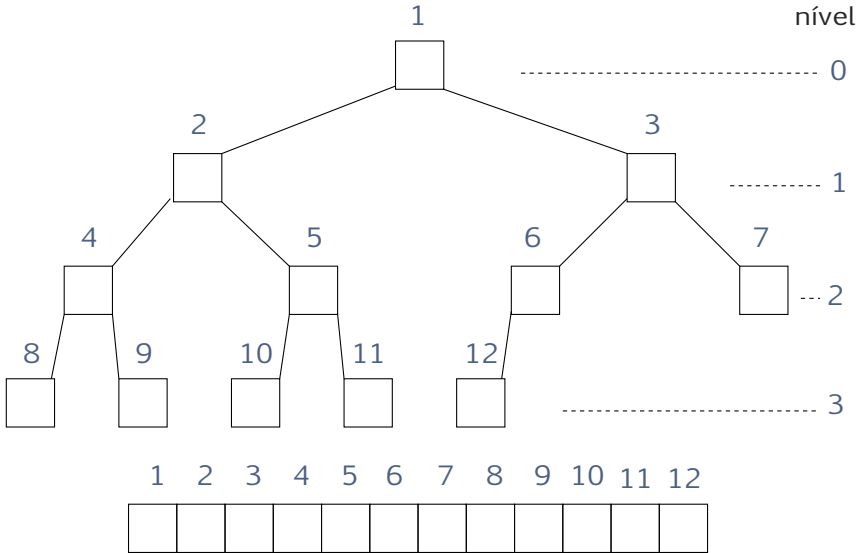
- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Representando um heap

Um heap é uma **árvore binária** armazenada em vetor $A[1 \dots n]$

- ▶ Filhos de um nó i
 - ▶ o filho esquerdo é $2i$
 - ▶ o filho direito é $2i + 1$
- ▶ Pais
 - ▶ o pai de um nó i é $\lfloor i/2 \rfloor$
 - ▶ o nó 1 não tem pai.
- ▶ Folhas
 - ▶ um nó i é folha se não tiver filhos, i.e., se $2i > n$.
 - ▶ as folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Exemplo de um heap



Árvore completa

Um heap é uma árvore **completa**

- ▶ cada nível $\ell = 0, 1, 2, \dots$ tem 2^ℓ nós (a não ser o último)
- ▶ se i está no nível ℓ , o número nós nos níveis anteriores é

$$2^0 + 2^1 + \dots + 2^{\ell-1} = 2^\ell - 1$$

- ▶ então os nós do nível ℓ são índices

$$2^\ell, 2^\ell + 1, 2^\ell + 2, \dots, 2^{\ell+1} - 1$$

Árvore completa

Um heap é uma árvore **completa**

- ▶ cada nível $\ell = 0, 1, 2, \dots$ tem 2^ℓ nós (a não ser o último)
- ▶ se i está no nível ℓ , o número nós nos níveis anteriores é

$$2^0 + 2^1 + \dots + 2^{\ell-1} = 2^\ell - 1$$

- ▶ então os nós do nível ℓ são índices

$$2^\ell, 2^\ell + 1, 2^\ell + 2, \dots, 2^{\ell+1} - 1$$

Árvore completa

Um heap é uma árvore **completa**

- ▶ cada nível $\ell = 0, 1, 2, \dots$ tem 2^ℓ nós (a não ser o último)
- ▶ se i está no nível ℓ , o número nós nos níveis anteriores é

$$2^0 + 2^1 + \dots + 2^{\ell-1} = 2^\ell - 1$$

- ▶ então os nós do nível ℓ são índices

$$2^\ell, 2^\ell + 1, 2^\ell + 2, \dots, 2^{\ell+1} - 1$$

Árvore completa

Um heap é uma árvore **completa**

- ▶ cada nível $\ell = 0, 1, 2, \dots$ tem 2^ℓ nós (a não ser o último)
- ▶ se i está no nível ℓ , o número nós nos níveis anteriores é

$$2^0 + 2^1 + \dots + 2^{\ell-1} = 2^\ell - 1$$

- ▶ então os nós do nível ℓ são índices

$$2^\ell, 2^\ell + 1, 2^\ell + 2, \dots, 2^{\ell+1} - 1$$

Árvore completa

Um heap é uma árvore **completa**

- ▶ cada nível $\ell = 0, 1, 2, \dots$ tem 2^ℓ nós (a não ser o último)
- ▶ se i está no nível ℓ , o número nós nos níveis anteriores é

$$2^0 + 2^1 + \dots + 2^{\ell-1} = 2^\ell - 1$$

- ▶ então os nós do nível ℓ são índices

$$2^\ell, 2^\ell + 1, 2^\ell + 2, \dots, 2^{\ell+1} - 1$$

Árvore completa

Um heap é uma árvore **completa**

- ▶ cada nível $\ell = 0, 1, 2, \dots$ tem 2^ℓ nós (a não ser o último)
- ▶ se i está no nível ℓ , o número nós nos níveis anteriores é

$$2^0 + 2^1 + \dots + 2^{\ell-1} = 2^\ell - 1$$

- ▶ então os nós do nível ℓ são índices

$$2^\ell, 2^\ell + 1, 2^\ell + 2, \dots, 2^{\ell+1} - 1$$

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

- ▶ se i está no nível ℓ , então

$$2^\ell \leq i < 2^{\ell+1} \Rightarrow$$

- ▶ assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

- ▶ se i está no nível ℓ , então

$$\begin{aligned} 2^\ell &\leq i < 2^{\ell+1} && \Rightarrow \\ \log_2 2^\ell &\leq \log_2 i < \log_2 2^{\ell+1} && \Rightarrow \\ \ell &\leq \log_2 i < \ell + 1 \end{aligned}$$

- ▶ assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

► se i está no nível ℓ , então

$$\begin{aligned} 2^\ell &\leq i < 2^{\ell+1} && \Rightarrow \\ \log_2 2^\ell &\leq \log_2 i < \log_2 2^{\ell+1} && \Rightarrow \\ \ell &\leq \log_2 i < \ell + 1 \end{aligned}$$

► assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

► se i está no nível ℓ , então

$$\begin{aligned} 2^\ell &\leq i < 2^{\ell+1} && \Rightarrow \\ \log_2 2^\ell &\leq \log_2 i < \log_2 2^{\ell+1} && \Rightarrow \\ \ell &\leq \log_2 i < \ell + 1 \end{aligned}$$

► assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

► se i está no nível ℓ , então

$$\begin{aligned} 2^\ell &\leq i < 2^{\ell+1} && \Rightarrow \\ \log_2 2^\ell &\leq \log_2 i < \log_2 2^{\ell+1} && \Rightarrow \\ \ell &\leq \log_2 i < \ell + 1 \end{aligned}$$

► assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

► se i está no nível ℓ , então

$$\begin{aligned} 2^\ell &\leq i < 2^{\ell+1} && \Rightarrow \\ \log_2 2^\ell &\leq \log_2 i < \log_2 2^{\ell+1} && \Rightarrow \\ \ell &\leq \log_2 i < \ell + 1 \end{aligned}$$

► assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

- ▶ se i está no nível ℓ , então

$$\begin{aligned} 2^\ell &\leq i < 2^{\ell+1} && \Rightarrow \\ \log_2 2^\ell &\leq \log_2 i < \log_2 2^{\ell+1} && \Rightarrow \\ \ell &\leq \log_2 i < \ell + 1 \end{aligned}$$

- ▶ assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Árvore completa

Nível de um nó

O nível de um nó de índice i é $\lfloor \log_2 i \rfloor$.

► se i está no nível ℓ , então

$$\begin{aligned} 2^\ell &\leq i < 2^{\ell+1} && \Rightarrow \\ \log_2 2^\ell &\leq \log_2 i < \log_2 2^{\ell+1} && \Rightarrow \\ \ell &\leq \log_2 i < \ell + 1 \end{aligned}$$

► assim, $\ell = \lfloor \log_2 i \rfloor$

Portanto, a altura da árvore é $\lfloor \log_2 n \rfloor$.

Definição

Um heap $A[1 \dots n]$ é chamado de **max-heap** se cada nó tiver valor maior que seus filhos, i.e., para cada i ,

$$\triangleright A[i] \geq A[2i] \quad \text{e} \quad A[i] \geq A[2i+1]$$

- ▶ essa restrição é a **propriedade de max-heap**
- ▶ o valor da raiz é um máximo do heap
- ▶ cada subárvore também é um max-heap

Definição

Um heap $A[1 \dots n]$ é chamado de **max-heap** se cada nó tiver valor maior que seus filhos, i.e., para cada i ,

$$\triangleright A[i] \geq A[2i] \quad \text{e} \quad A[i] \geq A[2i+1]$$

- ▶ essa restrição é a **propriedade de max-heap**
- ▶ o valor da raiz é um máximo do heap
- ▶ cada subárvore também é um max-heap

Definição

Um heap $A[1 \dots n]$ é chamado de **max-heap** se cada nó tiver valor maior que seus filhos, i.e., para cada i ,

▶ $A[i] \geq A[2i]$ e $A[i] \geq A[2i + 1]$

- ▶ essa restrição é a **propriedade de max-heap**
- ▶ o valor da raiz é um máximo do heap
- ▶ cada subárvore também é um max-heap

Definição

Um heap $A[1 \dots n]$ é chamado de **max-heap** se cada nó tiver valor maior que seus filhos, i.e., para cada i ,

▶ $A[i] \geq A[2i]$ e $A[i] \geq A[2i + 1]$

- ▶ essa restrição é a **propriedade de max-heap**
- ▶ o valor da raiz é um máximo do heap
- ▶ cada subárvore também é um max-heap

Definição

Um heap $A[1 \dots n]$ é chamado de **max-heap** se cada nó tiver valor maior que seus filhos, i.e., para cada i ,

▶ $A[i] \geq A[2i]$ e $A[i] \geq A[2i + 1]$

- ▶ essa restrição é a **propriedade de max-heap**
- ▶ o valor da raiz é um máximo do heap
- ▶ cada subárvore também é um max-heap

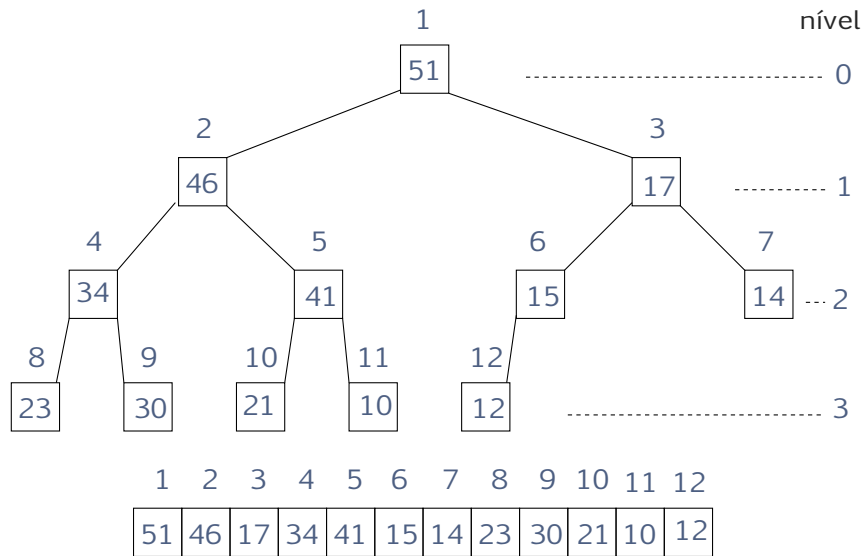
Definição

Um heap $A[1 \dots n]$ é chamado de **max-heap** se cada nó tiver valor maior que seus filhos, i.e., para cada i ,

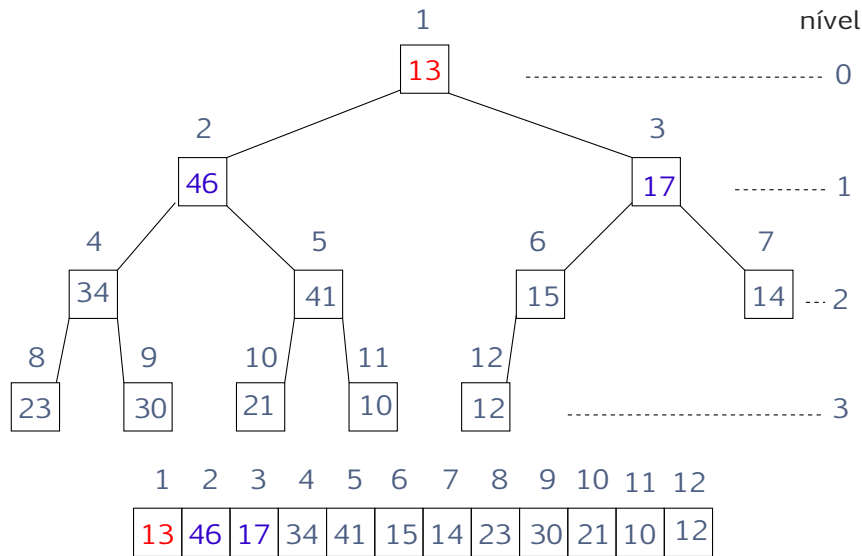
▶ $A[i] \geq A[2i]$ e $A[i] \geq A[2i + 1]$

- ▶ essa restrição é a **propriedade de max-heap**
- ▶ o valor da raiz é um máximo do heap
- ▶ cada subárvore também é um max-heap

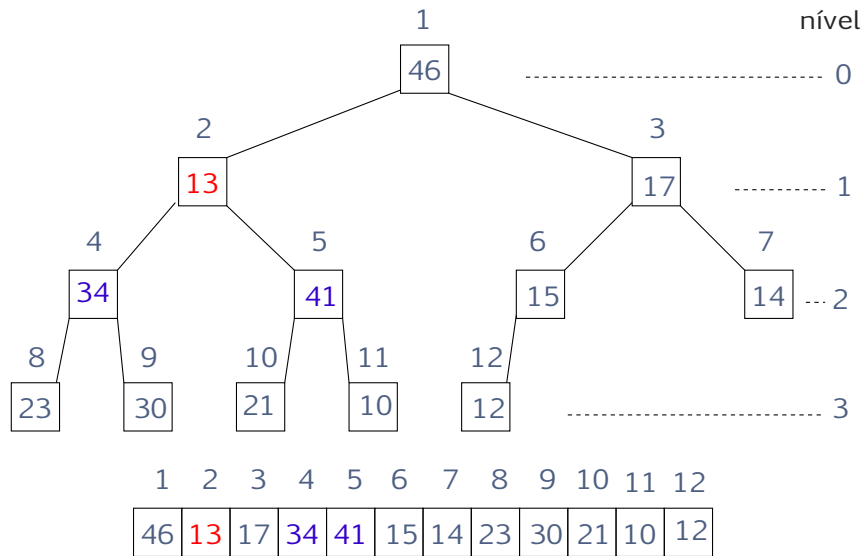
Exemplo de max-heap



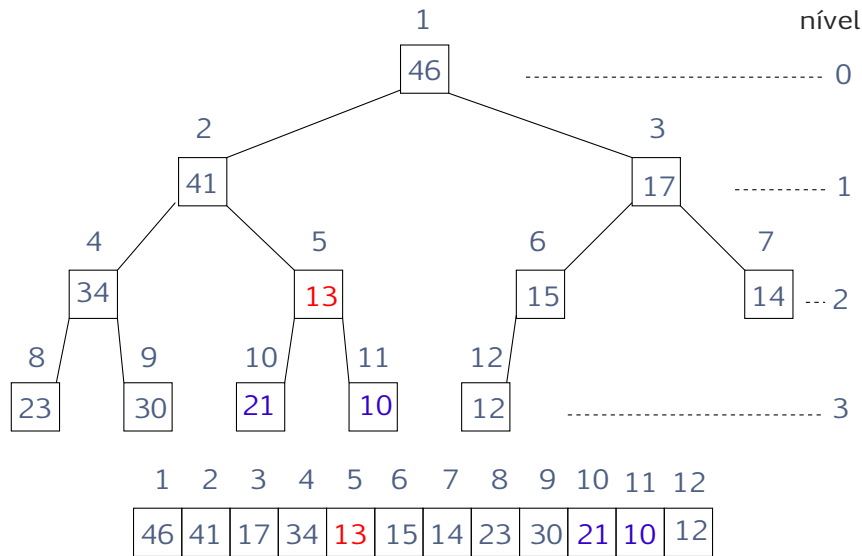
Manipulação de max-heap



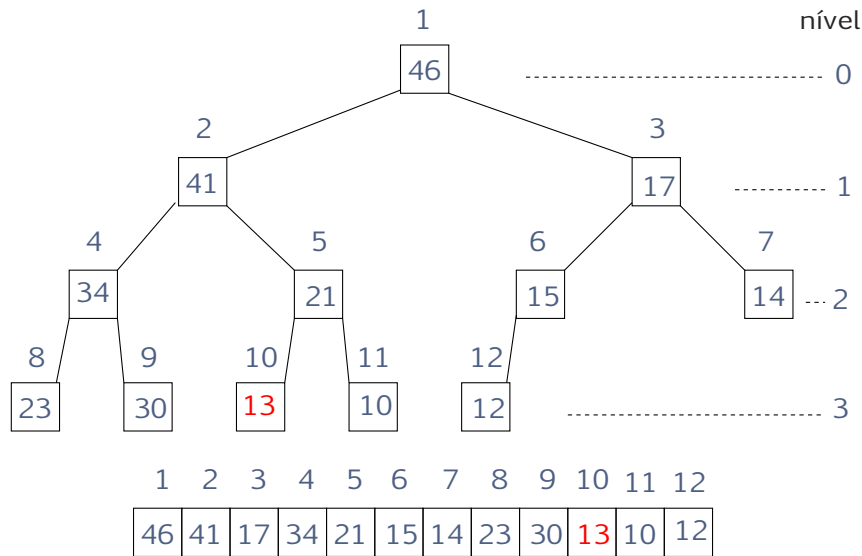
Manipulação de max-heap



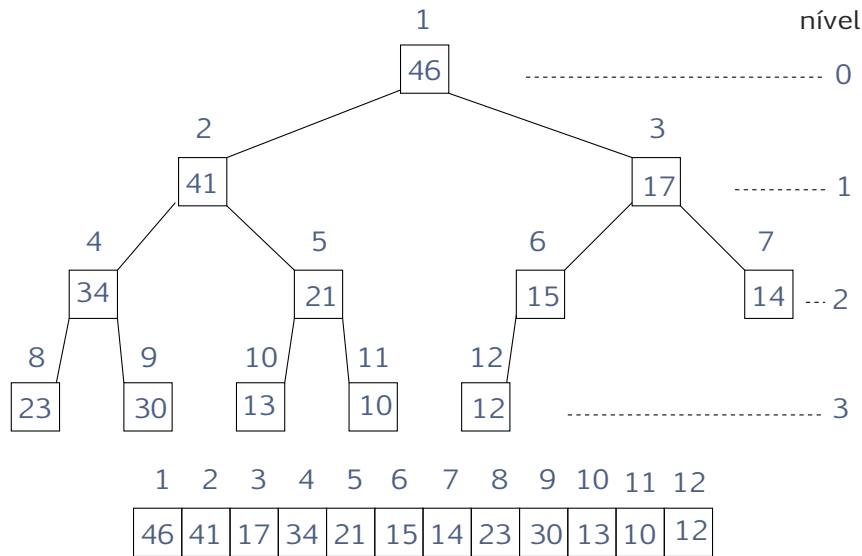
Manipulação de max-heap



Manipulação de max-heap



Manipulação de max-heap



Consertando um max-heap

- ▶ suponha que as subárvores $2i$ e $2i + 1$ são max-heaps
- ▶ como transformar a subárvore i em um max-heap?

MAX-HEAPIFY(A, n, i)

```
1   $e \leftarrow 2i$ 
2   $d \leftarrow 2i + 1$ 
3   $maior \leftarrow i$ 
4  se  $e \leq n$  e  $A[e] > A[maior]$ 
5       $maior \leftarrow e$ 
6  se  $d \leq n$  e  $A[d] > A[maior]$ 
7       $maior \leftarrow d$ 
8  se  $maior \neq i$ 
9       $A[i] \leftrightarrow A[maior]$ 
10     MAX-HEAPIFY( $A, n, maior$ )
```

Consertando um max-heap

- ▶ suponha que as subárvores $2i$ e $2i+1$ são max-heaps
- ▶ como transformar a subárvore i em um max-heap?

MAX-HEAPIFY(A, n, i)

```
1   $e \leftarrow 2i$ 
2   $d \leftarrow 2i + 1$ 
3   $maior \leftarrow i$ 
4  se  $e \leq n$  e  $A[e] > A[maior]$ 
5       $maior \leftarrow e$ 
6  se  $d \leq n$  e  $A[d] > A[maior]$ 
7       $maior \leftarrow d$ 
8  se  $maior \neq i$ 
9       $A[i] \leftrightarrow A[maior]$ 
10     MAX-HEAPIFY( $A, n, maior$ )
```

Consertando um max-heap

- ▶ suponha que as subárvores $2i$ e $2i + 1$ são max-heaps
- ▶ como transformar a subárvore i em um max-heap?

MAX-HEAPIFY(A, n, i)

```
1   $e \leftarrow 2i$ 
2   $d \leftarrow 2i + 1$ 
3   $maior \leftarrow i$ 
4  se  $e \leq n$  e  $A[e] > A[maior]$ 
5       $maior \leftarrow e$ 
6  se  $d \leq n$  e  $A[d] > A[maior]$ 
7       $maior \leftarrow d$ 
8  se  $maior \neq i$ 
9       $A[i] \leftrightarrow A[maior]$ 
10     MAX-HEAPIFY( $A, n, maior$ )
```

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i+1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i+1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i+1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i+1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i+1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i+1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i+1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i+1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i + 1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i + 1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i + 1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i + 1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i + 1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i + 1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i + 1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i + 1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i + 1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i + 1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Correção de MAX-HEAPIFY

Lema

MAX-HEAPIFY transforma a subárvore i em max-heap.

Ideia para demonstração: indução na altura h do nó i .

- ▶ se $h = 0$, então i é folha e o algoritmo está correto
- ▶ considere um nó i com altura $h > 0$
- ▶ suponha que o algoritmo funciona para árvores menores
 - ▶ antes da linha 8, temos $A[\text{maior}] \geq A[i], A[2i], A[2i + 1]$
 - ▶ após a linha 9, temos $A[i] \geq A[2i], A[2i + 1]$
 - ▶ segue que $A[i]$ é máximo no vetor
 - ▶ pela hipótese de indução, MAX-HEAPIFY transforma a subárvore com raiz maior em max-heap
 - ▶ segue que i é max-heap

Complexidade de MAX-HEAPIFY

MAX-HEAPIFY(A, n, i)		Tempo
1	$e \leftarrow 2i$	?
2	$d \leftarrow 2i + 1$	?
3	$maior \leftarrow i$	?
4	se $e \leq n$ e $A[e] > A[i]$	?
4	$maior \leftarrow e$	?
6	se $d \leq n$ e $A[d] > A[maior]$	?
7	$maior \leftarrow d$	?
8	se $maior \neq i$	?
9	$A[i] \leftrightarrow A[maior]$	?
10	MAX-HEAPIFY($A, n, maior$)	?

O tempo de execução é $T(h) = T(h - 1) + \Theta(1) = O(h)$

Complexidade de MAX-HEAPIFY

MAX-HEAPIFY(A, n, i)		Tempo
1	$e \leftarrow 2i$	$\Theta(1)$
2	$d \leftarrow 2i + 1$	$\Theta(1)$
3	$maior \leftarrow i$	$\Theta(1)$
4	se $e \leq n$ e $A[e] > A[i]$	$\Theta(1)$
4	$maior \leftarrow e$	$O(1)$
6	se $d \leq n$ e $A[d] > A[maior]$	$\Theta(1)$
7	$maior \leftarrow d$	$O(1)$
8	se $maior \neq i$	$\Theta(1)$
9	$A[i] \leftrightarrow A[maior]$	$O(1)$
10	MAX-HEAPIFY($A, n, maior$)	$T(h - 1)$

O tempo de execução é $T(h) = T(h - 1) + \Theta(1) = O(h)$

Complexidade de MAX-HEAPIFY

MAX-HEAPIFY(A, n, i)		Tempo
1	$e \leftarrow 2i$	$\Theta(1)$
2	$d \leftarrow 2i + 1$	$\Theta(1)$
3	$maior \leftarrow i$	$\Theta(1)$
4	se $e \leq n$ e $A[e] > A[i]$	$\Theta(1)$
4	$maior \leftarrow e$	$O(1)$
6	se $d \leq n$ e $A[d] > A[maior]$	$\Theta(1)$
7	$maior \leftarrow d$	$O(1)$
8	se $maior \neq i$	$\Theta(1)$
9	$A[i] \leftrightarrow A[maior]$	$O(1)$
10	MAX-HEAPIFY($A, n, maior$)	$T(h - 1)$

O tempo de execução é $T(h) = T(h - 1) + \Theta(1) = O(h)$

Complexidade de MAX-HEAPIFY

MAX-HEAPIFY(A, n, i)		Tempo
1	$e \leftarrow 2i$	$\Theta(1)$
2	$d \leftarrow 2i + 1$	$\Theta(1)$
3	$maior \leftarrow i$	$\Theta(1)$
4	se $e \leq n$ e $A[e] > A[i]$	$\Theta(1)$
4	$maior \leftarrow e$	$O(1)$
6	se $d \leq n$ e $A[d] > A[maior]$	$\Theta(1)$
7	$maior \leftarrow d$	$O(1)$
8	se $maior \neq i$	$\Theta(1)$
9	$A[i] \leftrightarrow A[maior]$	$O(1)$
10	MAX-HEAPIFY($A, n, maior$)	$T(h - 1)$

O tempo de execução é $T(h) = T(h - 1) + \Theta(1) = O(h)$

Construindo um max-heap

Podemos consertar um vetor inteiro

- ▶ recebemos um vetor $A[1 \dots n]$ desorganizado
- ▶ mas cada folha já é um heap
- ▶ consertamos o penúltimo nível
- ▶ depois o antepenúltimo
- ▶ e assim por diante

Construindo um max-heap

Podemos consertar um vetor inteiro

- ▶ recebemos um vetor $A[1 \dots n]$ desorganizado
- ▶ mas cada folha já é um heap
- ▶ consertamos o penúltimo nível
- ▶ depois o antepenúltimo
- ▶ e assim por diante

Construindo um max-heap

Podemos consertar um vetor inteiro

- ▶ recebemos um vetor $A[1 \dots n]$ desorganizado
- ▶ mas cada folha já é um heap
- ▶ consertamos o penúltimo nível
- ▶ depois o antepenúltimo
- ▶ e assim por diante

Construindo um max-heap

Podemos consertar um vetor inteiro

- ▶ recebemos um vetor $A[1 \dots n]$ desorganizado
- ▶ mas cada folha já é um heap
- ▶ consertamos o penúltimo nível
- ▶ depois o antepenúltimo
- ▶ e assim por diante

Construindo um max-heap

Podemos consertar um vetor inteiro

- ▶ recebemos um vetor $A[1 \dots n]$ desorganizado
- ▶ mas cada folha já é um heap
- ▶ consertamos o penúltimo nível
- ▶ depois o antepenúltimo
- ▶ e assim por diante

Construindo um max-heap

Podemos consertar um vetor inteiro

- ▶ recebemos um vetor $A[1 \dots n]$ desorganizado
- ▶ mas cada folha já é um heap
- ▶ consertamos o penúltimo nível
- ▶ depois o antepenúltimo
- ▶ e assim por diante

Construindo um max-heap

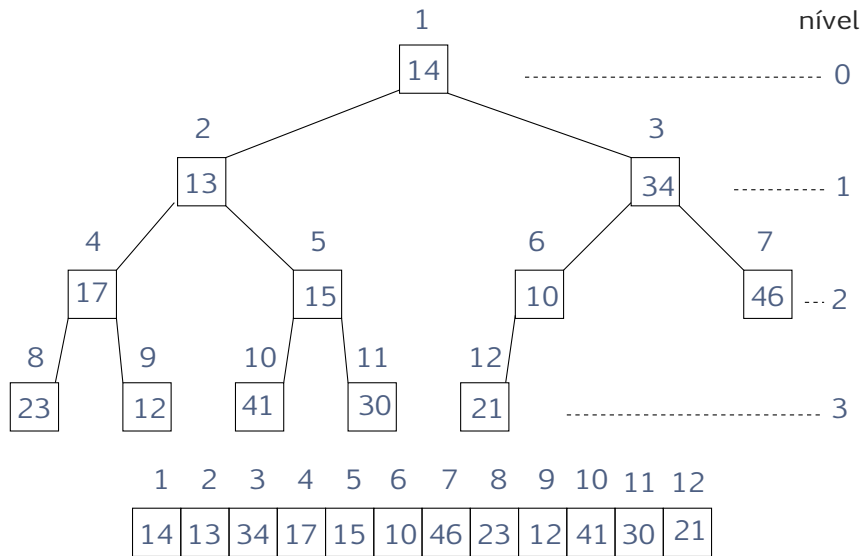
Podemos consertar um vetor inteiro

- ▶ recebemos um vetor $A[1 \dots n]$ desorganizado
- ▶ mas cada folha já é um heap
- ▶ consertamos o penúltimo nível
- ▶ depois o antepenúltimo
- ▶ e assim por diante

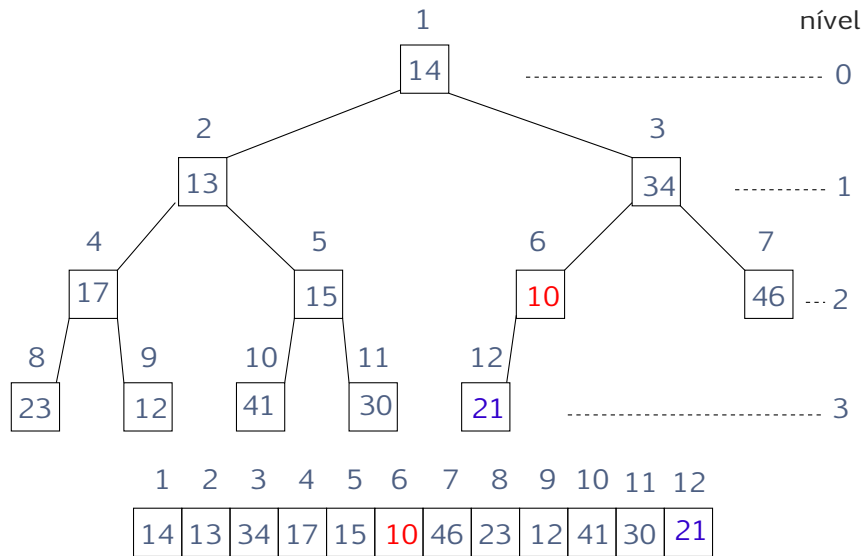
BUILD-MAX-HEAP(A, n)

```
1  para  $i \leftarrow \lfloor n/2 \rfloor$  decrescendo até 1
2      MAX-HEAPIFY( $A, n, i$ )
```

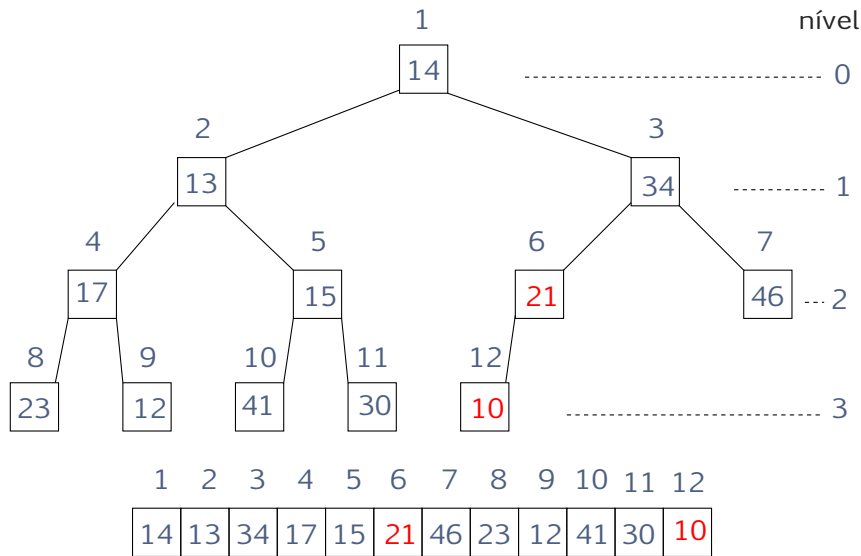
Construção de um max-heap



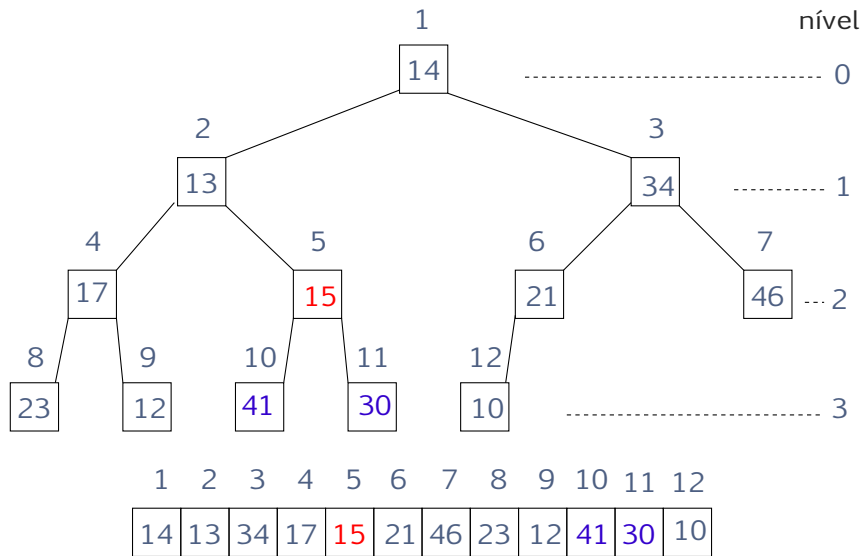
Construção de um max-heap



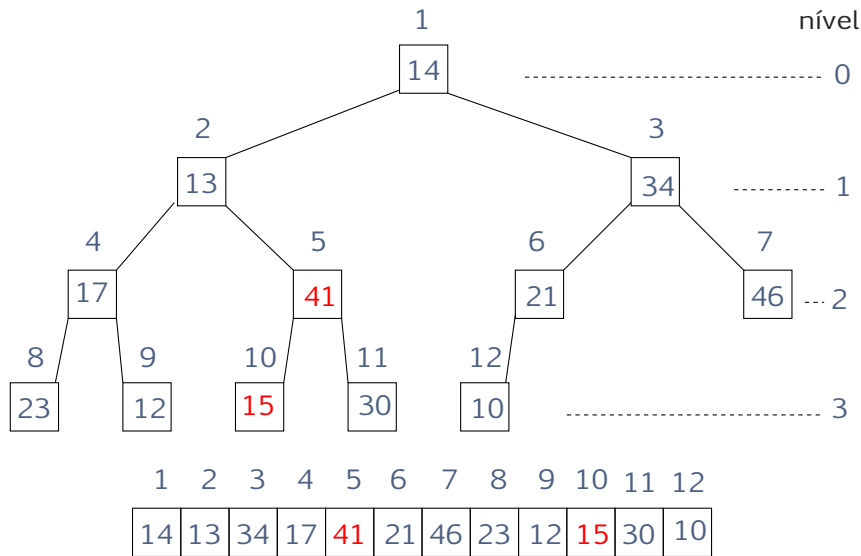
Construção de um max-heap



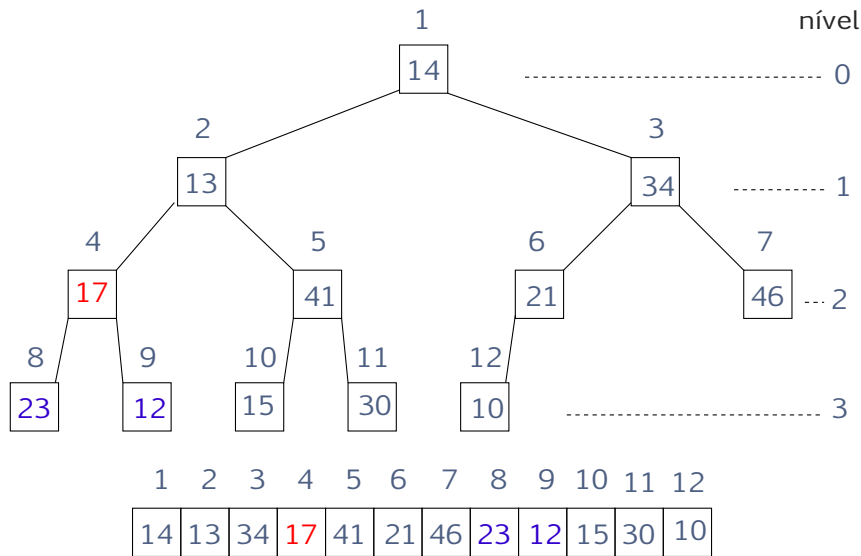
Construção de um max-heap



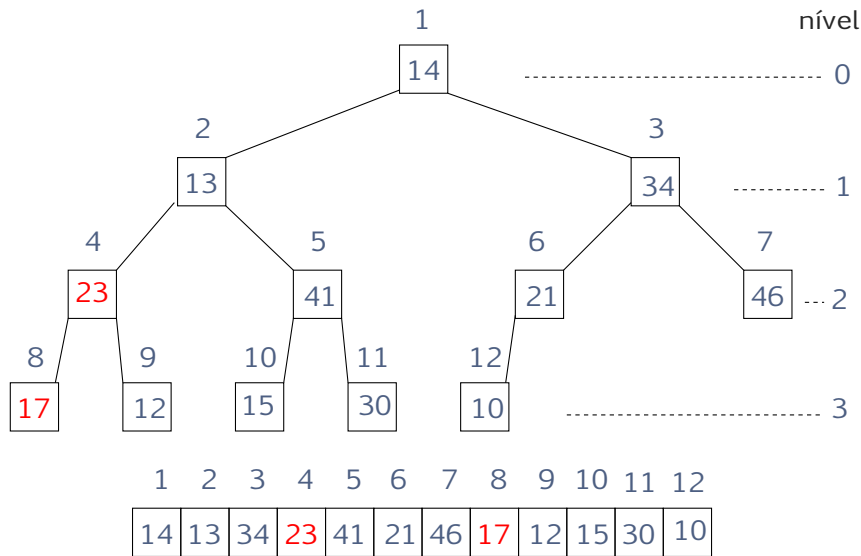
Construção de um max-heap



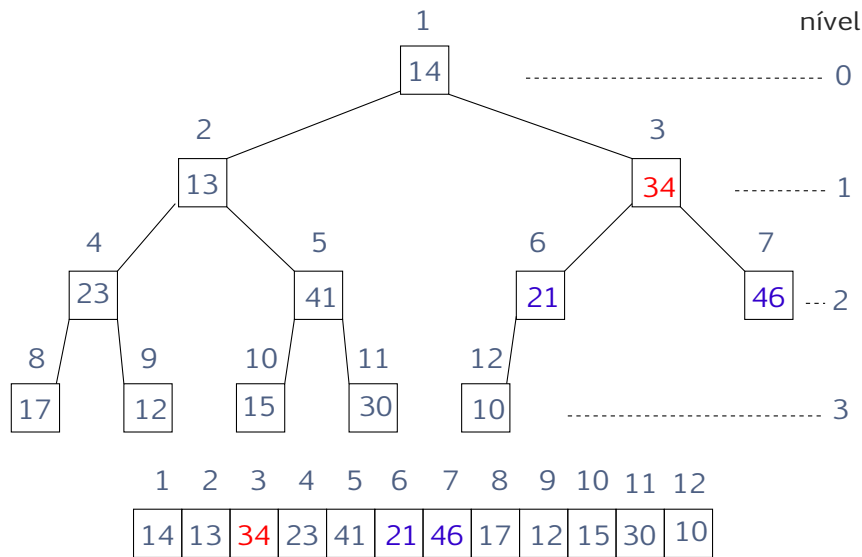
Construção de um max-heap



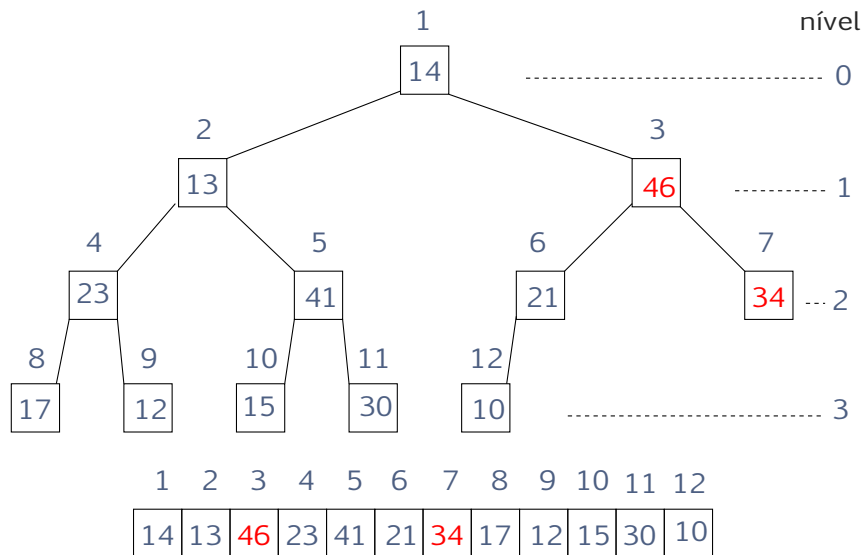
Construção de um max-heap



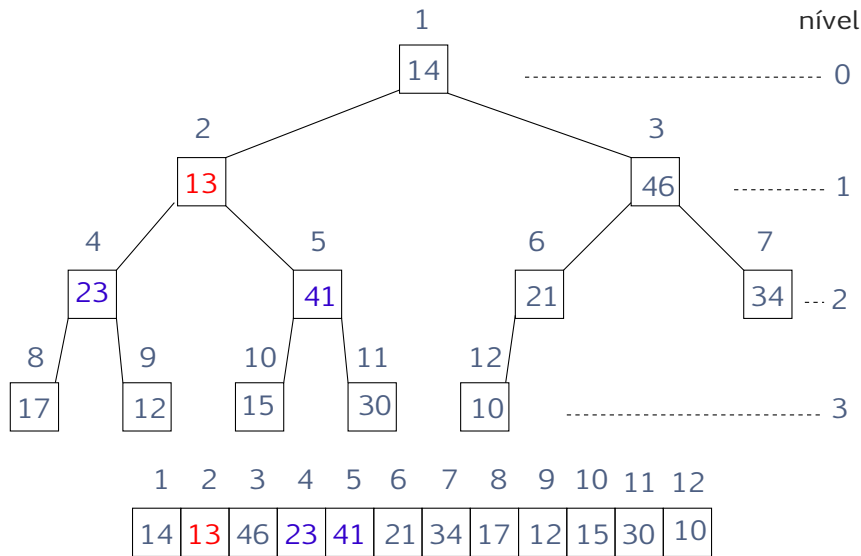
Construção de um max-heap



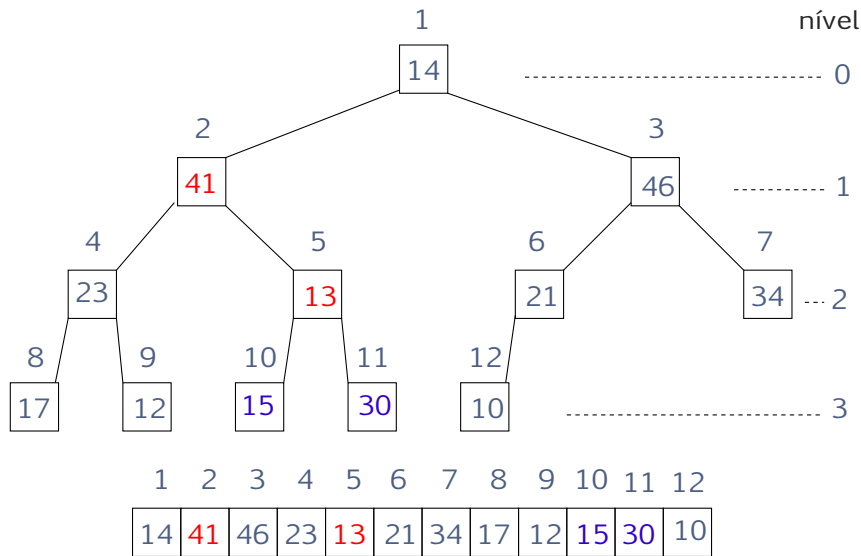
Construção de um max-heap



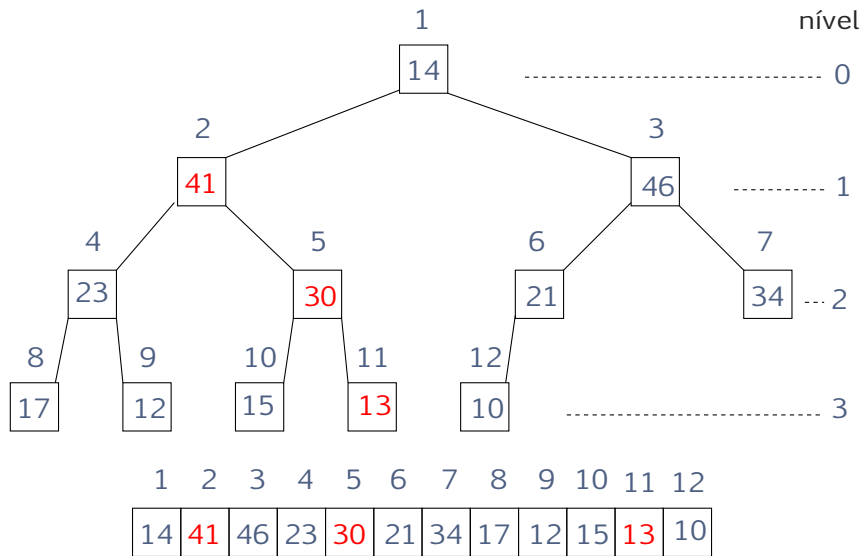
Construção de um max-heap



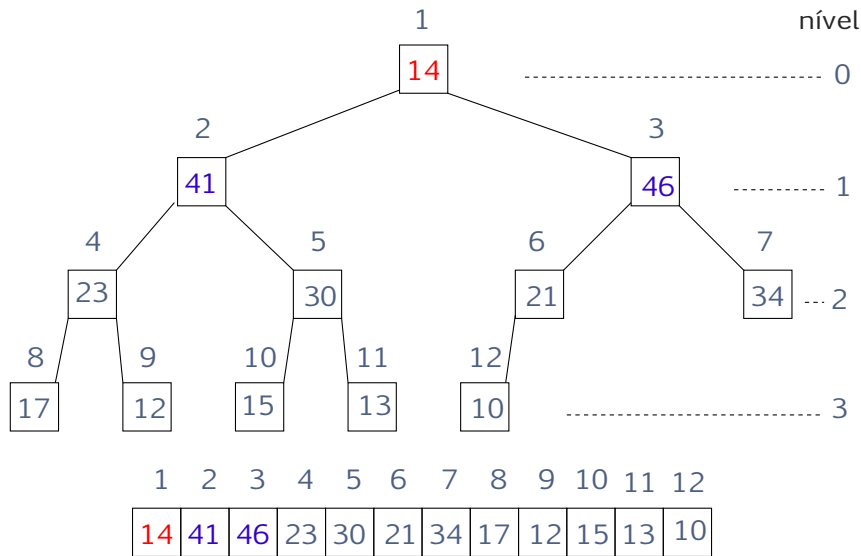
Construção de um max-heap



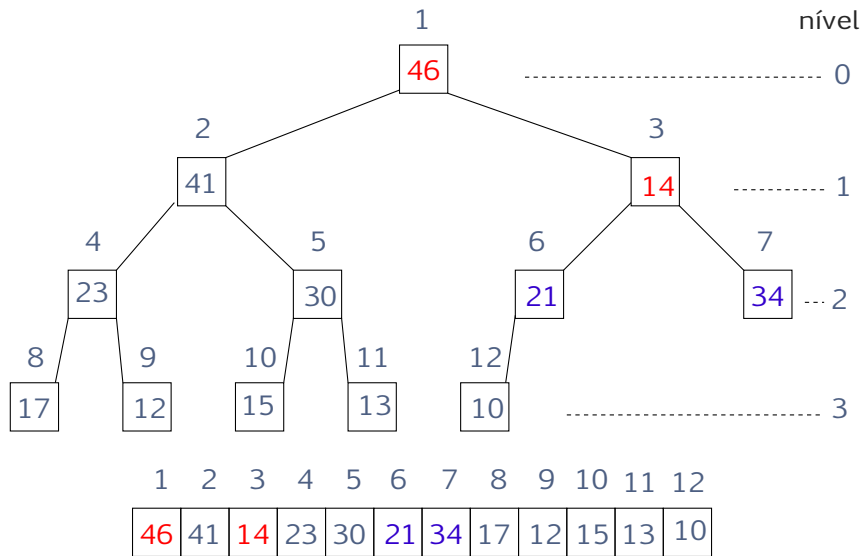
Construção de um max-heap



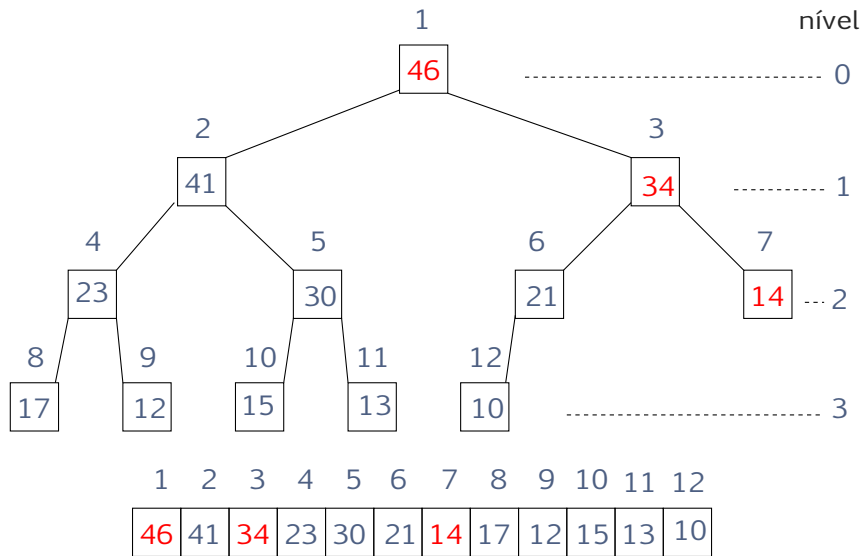
Construção de um max-heap



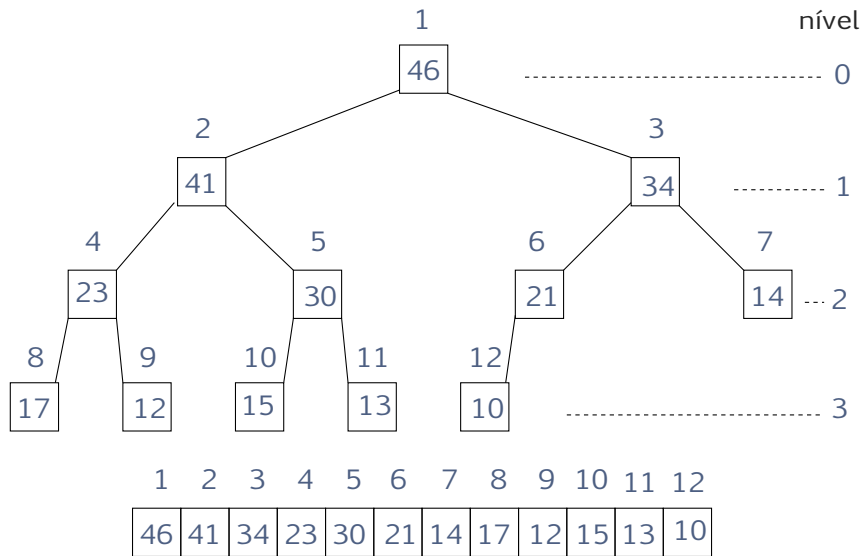
Construção de um max-heap



Construção de um max-heap



Construção de um max-heap



Análise de BUILD-MAX-HEAP

BUILD-MAX-HEAP(A, n)

```
1  para  $i \leftarrow \lfloor n/2 \rfloor$  decrescendo até 1  
2      MAX-HEAPIFY( $A, n, i$ )
```

Invariante

No início de cada iteração, $i + 1, \dots, n$ são raízes de max-heaps.

Complexidade

- ▶ uma análise rápida leva a $T(n) = n \cdot O(\log n) = O(n \log n)$
- ▶ mas na verdade mostraremos que $T(n)$ é **linear**!

Análise de BUILD-MAX-HEAP

BUILD-MAX-HEAP(A, n)

```
1  para  $i \leftarrow \lfloor n/2 \rfloor$  decrescendo até 1  
2      MAX-HEAPIFY( $A, n, i$ )
```

Invariante

No início de cada iteração, $i + 1, \dots, n$ são raízes de max-heaps.

Complexidade

- ▶ uma análise rápida leva a $T(n) = n \cdot O(\log n) = O(n \log n)$
- ▶ mas na verdade mostraremos que $T(n)$ é **linear**!

Análise de BUILD-MAX-HEAP

BUILD-MAX-HEAP(A, n)

```
1  para  $i \leftarrow \lfloor n/2 \rfloor$  decrescendo até 1  
2      MAX-HEAPIFY( $A, n, i$ )
```

Invariante

No início de cada iteração, $i + 1, \dots, n$ são raízes de max-heaps.

Complexidade

- ▶ uma análise rápida leva a $T(n) = n \cdot O(\log n) = O(n \log n)$
- ▶ mas na verdade mostraremos que $T(n)$ é **linear**!

Análise de BUILD-MAX-HEAP

BUILD-MAX-HEAP(A, n)

```
1  para  $i \leftarrow \lfloor n/2 \rfloor$  decrescendo até 1  
2      MAX-HEAPIFY( $A, n, i$ )
```

Invariante

No início de cada iteração, $i + 1, \dots, n$ são raízes de max-heaps.

Complexidade

- ▶ uma análise rápida leva a $T(n) = n \cdot O(\log n) = O(n \log n)$
- ▶ mas na verdade mostraremos que $T(n)$ é **linear**!

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP

Análise mais cuidadosa:

- ▶ seja k a altura da árvore inteira
 - ▶ temos 1 nó de altura k
 - ▶ temos 2 nós de altura $k - 1$
 - ▶ temos 4 nós de altura $k - 2$
 - ▶ e assim por diante
- ▶ uma chamada de MAX-HEAPIFY leva tempo $O(h)$
- ▶ vamos somar os tempos de todas as chamadas

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left\{ \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ 1 - \frac{1}{2^k} < 1 \right. \\ &\quad \left. + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \right. \\ &\quad \left. + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \right. \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left\{ \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ 1 - \frac{1}{2^k} < 1 \right. \\ &\quad \left. + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \right. \\ &\quad \left. + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \right. \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left\{ \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ 1 - \frac{1}{2^k} < 1 \right. \\ &\quad \left. + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \right. \\ &\quad \left. + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \right. \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left\{ \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ 1 - \frac{1}{2^k} < 1 \right. \\ &\quad \left. + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \right. \\ &\quad \left. + \frac{1}{2^3} + \dots + \frac{1}{2^k} \right\} \left\{ \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \right. \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left\{ \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \left\{ 1 - \frac{1}{2^k} < 1 \right. \\ &\quad \left. + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \left\{ \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \right. \\ &\quad \left. + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \left\{ \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \right. \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \left\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad \quad + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \left\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \left\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \left\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \left\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \left\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

Complexidade de BUILD-MAX-HEAP (cont)

Seja $k = \lfloor \log_2 n \rfloor$ a altura da árvore inteira, então

$$T(n) = \sum_{h=1}^k 2^{k-h} \cdot O(h) = O(2^k) \sum_{h=1}^k \frac{h}{2^h}$$

Note que

$$\begin{aligned} \sum_{h=1}^k \frac{h}{2^h} &= \left. \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} 1 - \frac{1}{2^k} < 1 \\ &\quad + \left. \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{2} - \frac{1}{2^k} < \frac{1}{2} \\ &\quad + \left. \frac{1}{2^3} + \cdots + \frac{1}{2^k} \right\} \frac{1}{4} - \frac{1}{2^k} < \frac{1}{4} \end{aligned}$$

Ou seja,

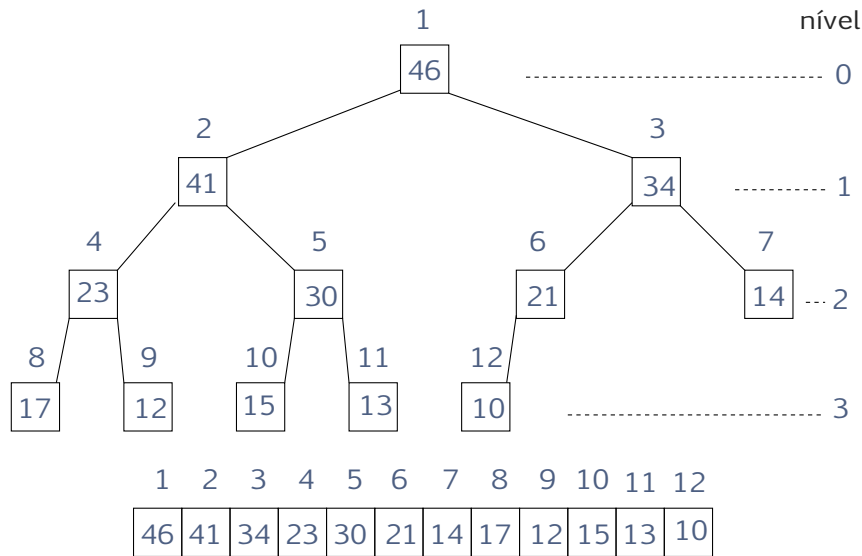
$$T(n) = O(2^k) \sum_{h=0}^{k-1} \frac{1}{2^h} \leq O(2^k) \cdot \sum_{h=0}^{\infty} \frac{1}{2^h} = O(2^k) \cdot 2 = O(n)$$

O algoritmo HEAP-SORT

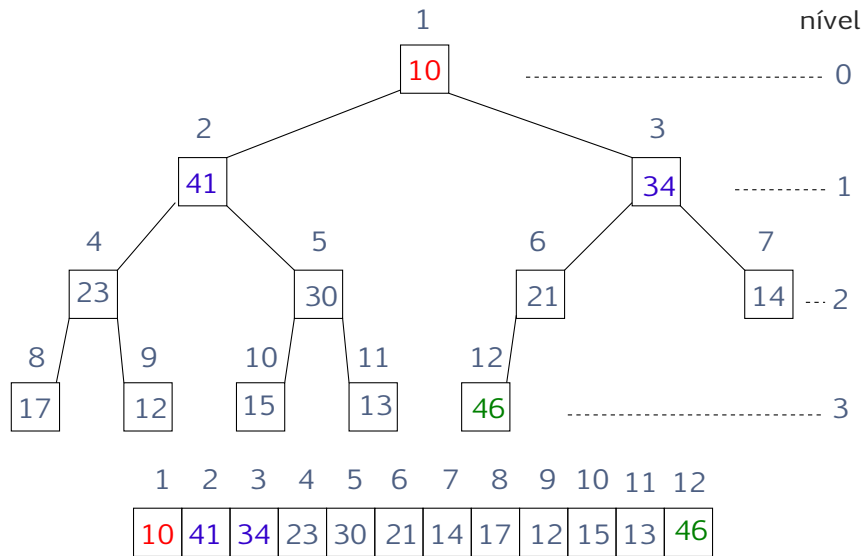
HEAP-SORT(A, n)

- 1 **BUILD-MAX-HEAP**(A, n)
- 2 para $m \leftarrow n$ decrescendo até 2
- 3 $A[1] \leftrightarrow A[m]$
- 4 **MAX-HEAPIFY**($A, m - 1, 1$)

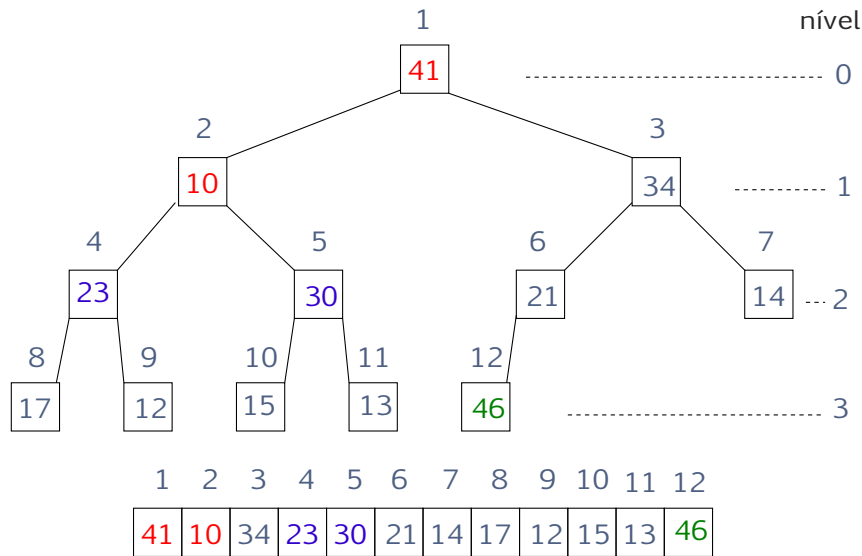
Execução de HEAP-SORT



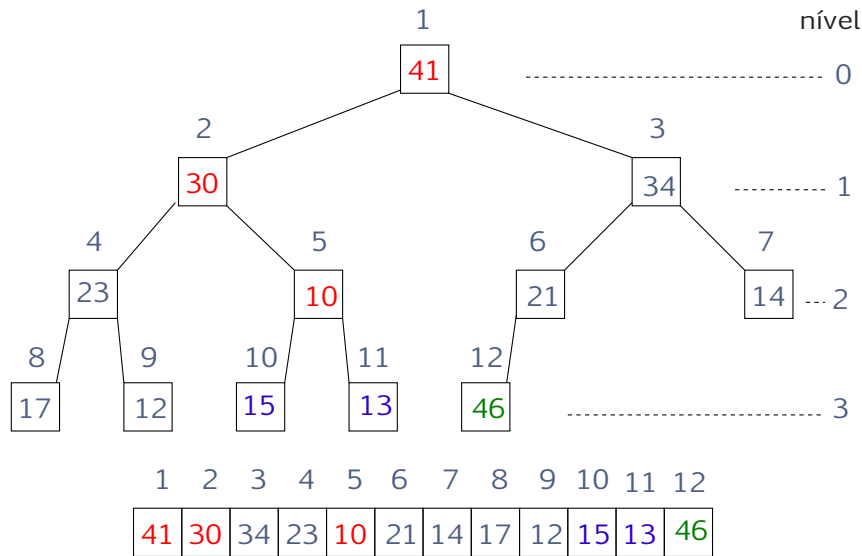
Execução de HEAP-SORT



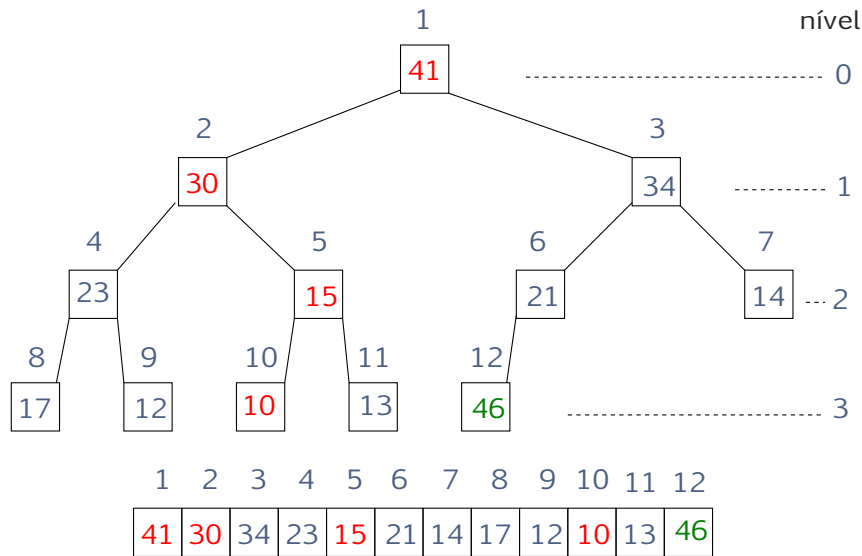
Execução de HEAP-SORT



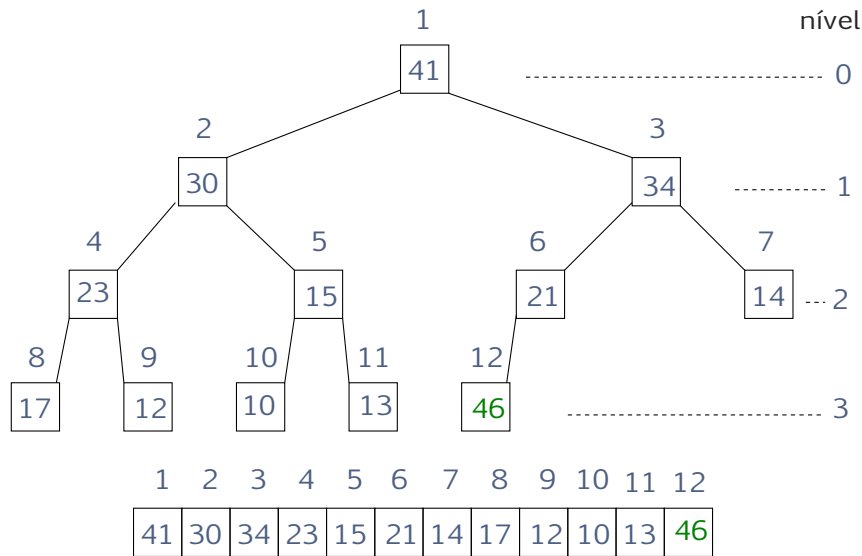
Execução de HEAP-SORT



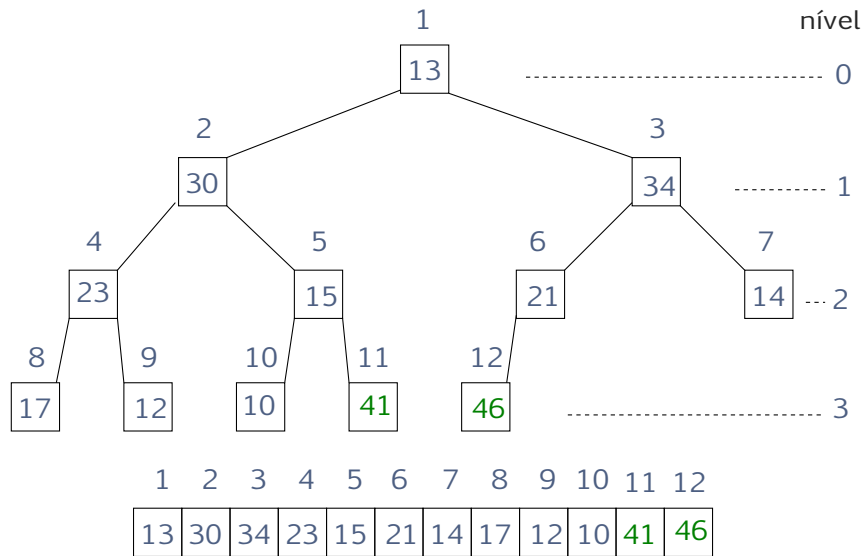
Execução de HEAP-SORT



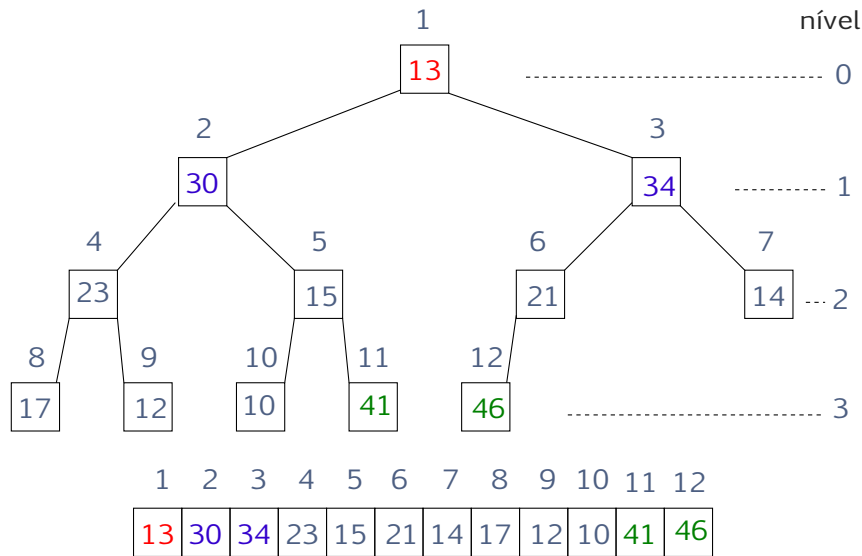
Execução de HEAP-SORT



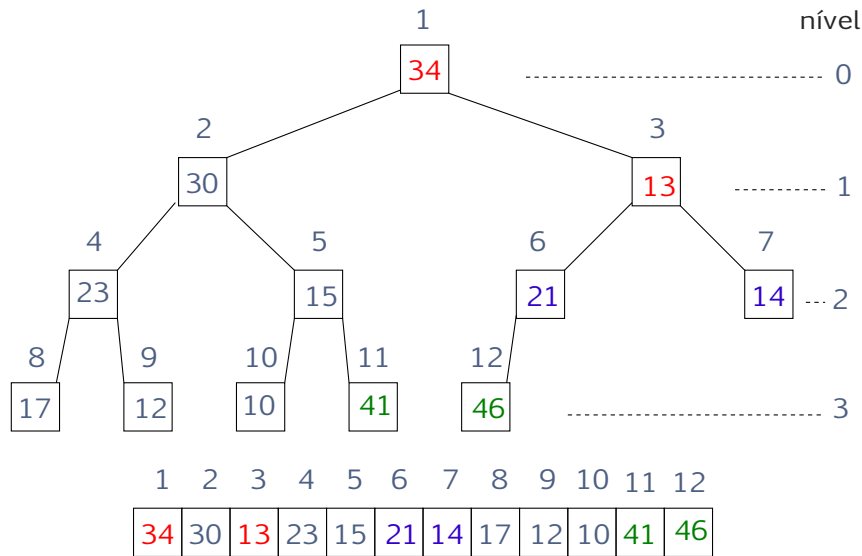
Execução de HEAP-SORT



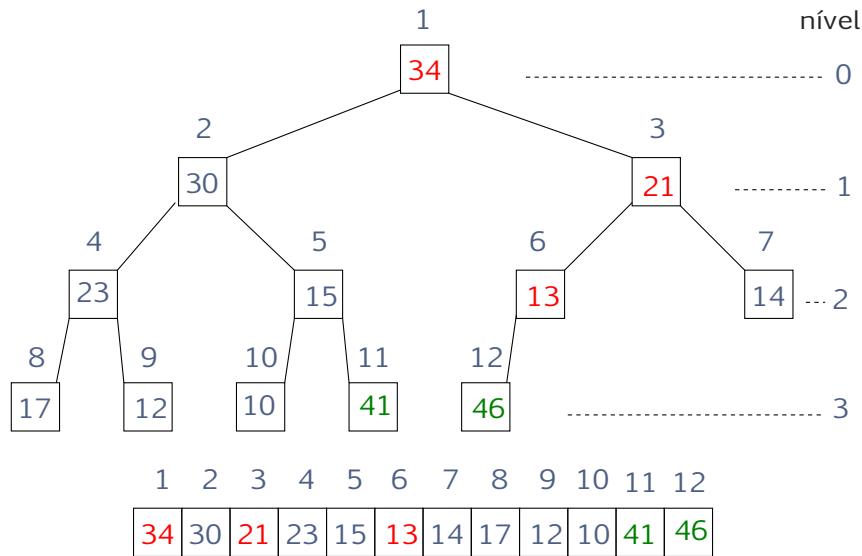
Execução de HEAP-SORT



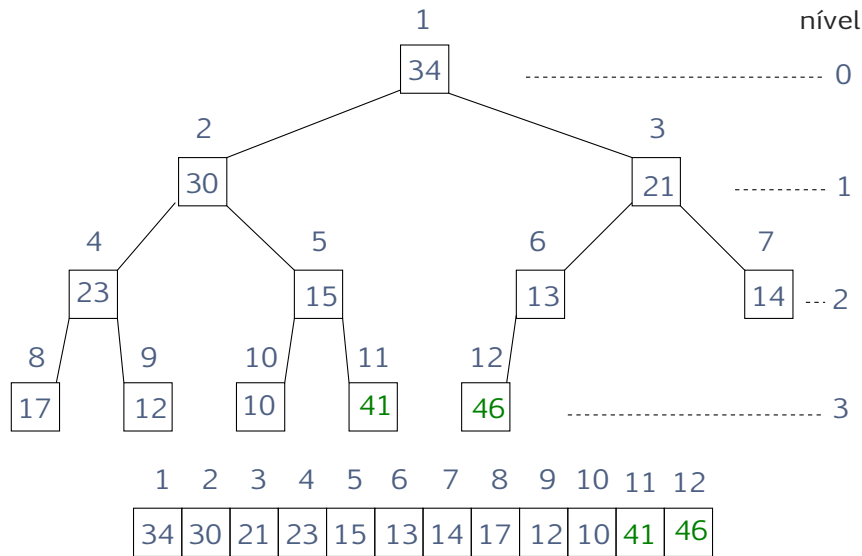
Execução de HEAP-SORT



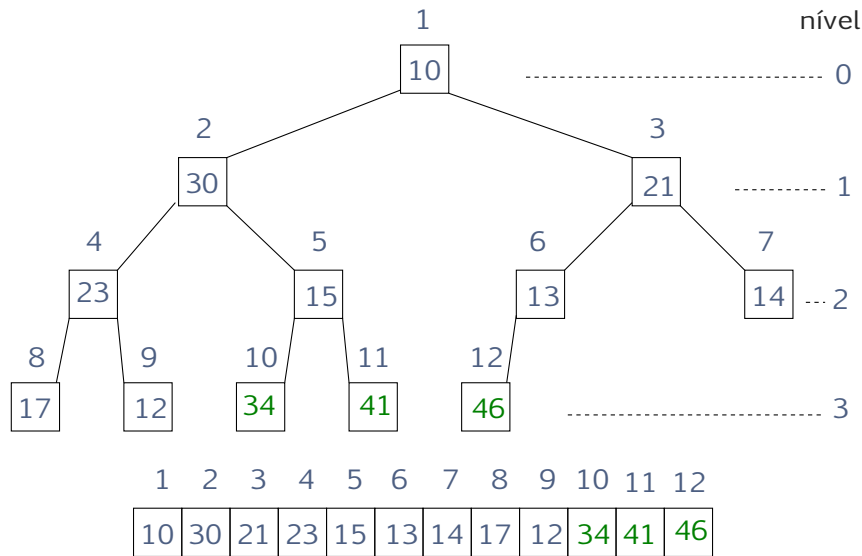
Execução de HEAP-SORT



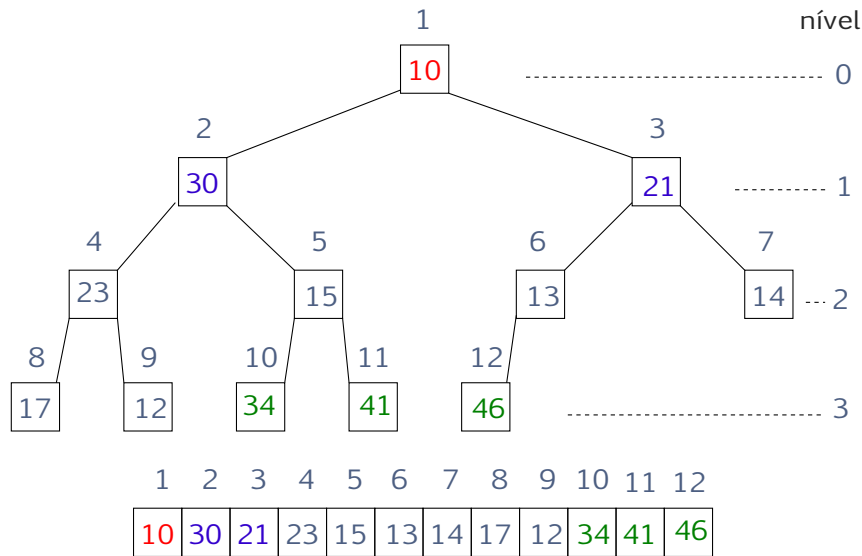
Execução de HEAP-SORT



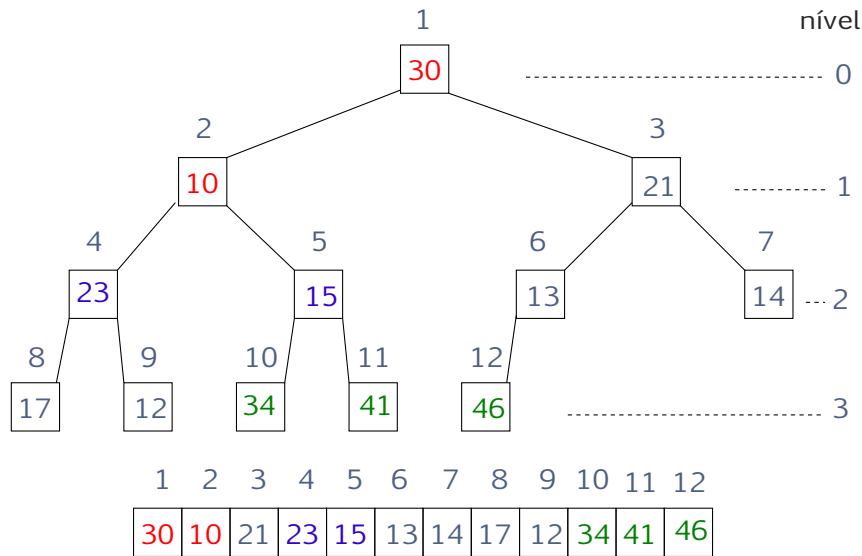
Execução de HEAP-SORT



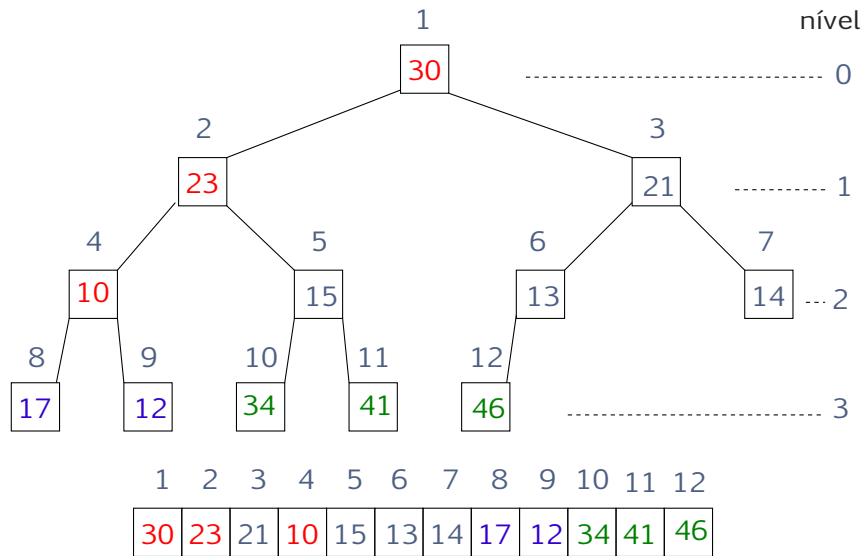
Execução de HEAP-SORT



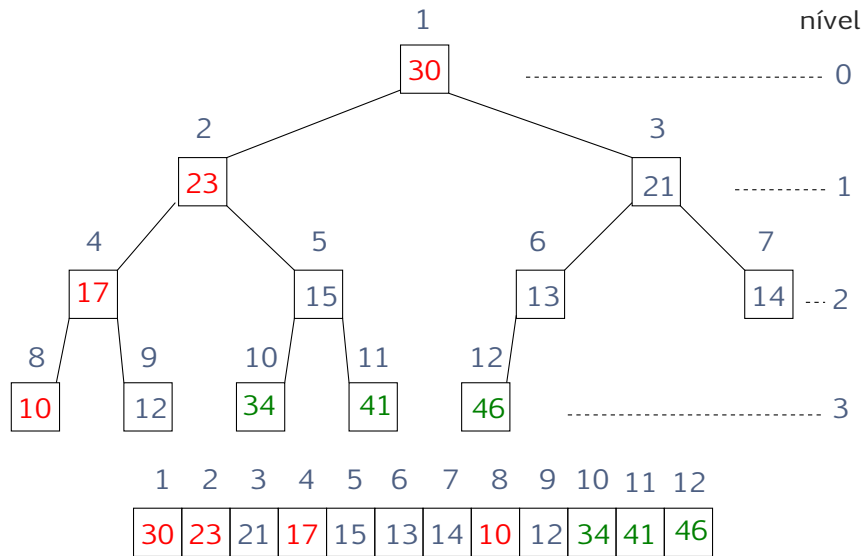
Execução de HEAP-SORT



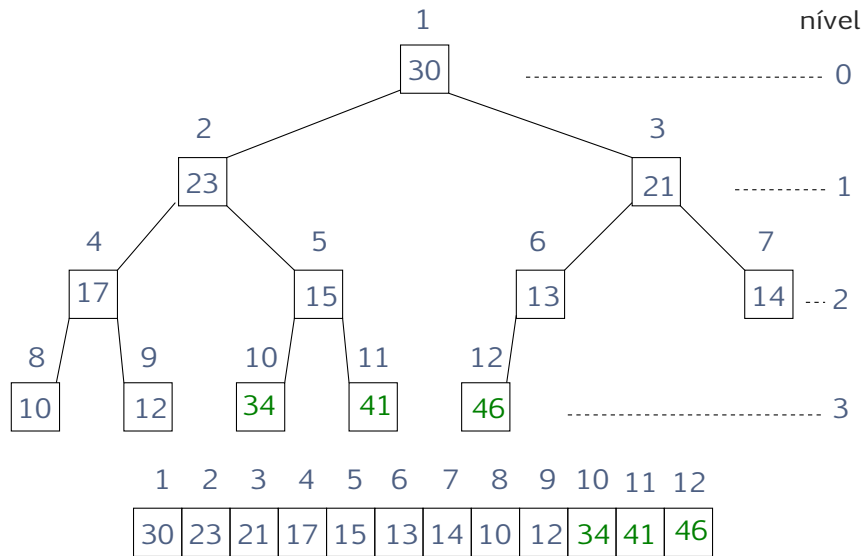
Execução de HEAP-SORT



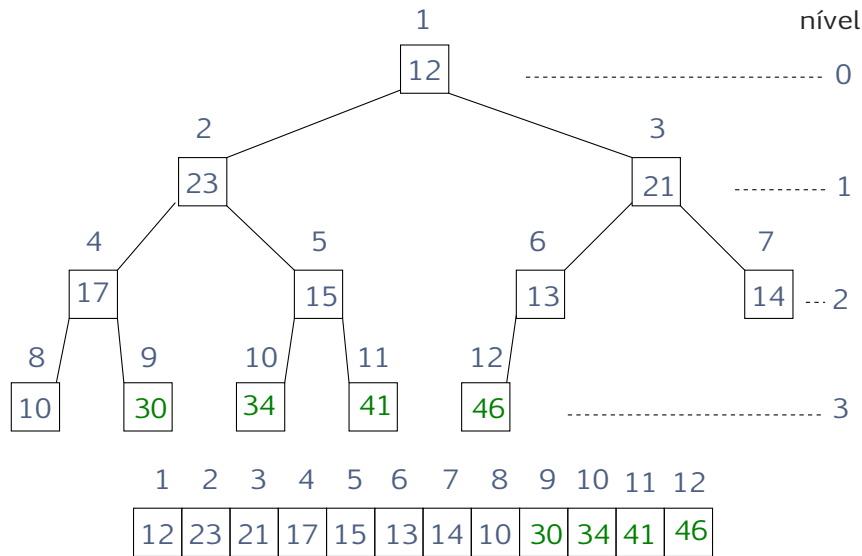
Execução de HEAP-SORT



Execução de HEAP-SORT



Execução de HEAP-SORT



Análise de HEAP-SORT

HEAP-SORT(A, n)

```
1  BUILD-MAX-HEAP( $A, n$ )
2  para  $m \leftarrow n$  decrescendo até 2
3       $A[1] \leftrightarrow A[m]$ 
4      MAX-HEAPIFY( $A, m - 1, 1$ )
```

Invariantes

No início de cada iteração vale

1. $A[1 \dots m]$ é um max-heap
2. $A[m + 1 \dots n]$ é crescente
3. $A[1 \dots m] \leq A[m + 1]$

Análise de HEAP-SORT

HEAP-SORT(A, n)

```
1  BUILD-MAX-HEAP( $A, n$ )
2  para  $m \leftarrow n$  decrescendo até 2
3       $A[1] \leftrightarrow A[m]$ 
4      MAX-HEAPIFY( $A, m - 1, 1$ )
```

Invariantes

No início de cada iteração vale

1. $A[1 \dots m]$ é um max-heap
2. $A[m + 1 \dots n]$ é crescente
3. $A[1 \dots m] \leq A[m + 1]$

Análise de HEAP-SORT

HEAP-SORT(A, n)

```
1  BUILD-MAX-HEAP( $A, n$ )
2  para  $m \leftarrow n$  decrescendo até 2
3       $A[1] \leftrightarrow A[m]$ 
4      MAX-HEAPIFY( $A, m - 1, 1$ )
```

Invariantes

No início de cada iteração vale

1. $A[1 \dots m]$ é um max-heap
2. $A[m + 1 \dots n]$ é crescente
3. $A[1 \dots m] \leq A[m + 1]$

Análise de HEAP-SORT

HEAP-SORT(A, n)

```
1  BUILD-MAX-HEAP( $A, n$ )
2  para  $m \leftarrow n$  decrescendo até 2
3       $A[1] \leftrightarrow A[m]$ 
4      MAX-HEAPIFY( $A, m - 1, 1$ )
```

Invariantes

No início de cada iteração vale

1. $A[1 \dots m]$ é um max-heap
2. $A[m + 1 \dots n]$ é crescente
3. $A[1 \dots m] \leq A[m + 1]$

Complexidade de HEAP-SORT

HEAP-SORT(A, n)		Tempo
1	BUILD-MAX-HEAP(A, n)	?
2	para $m \leftarrow n$ decrescendo até 2	?
3	$A[1] \leftrightarrow A[m]$	?
4	MAX-HEAPIFY($A, m - 1, 1$)	?

Consumo de tempo no pior caso? $O(n \log n)$

Complexidade de HEAP-SORT

HEAP-SORT(A, n)		Tempo
1	BUILD-MAX-HEAP(A, n)	$\Theta(n)$
2	para $m \leftarrow n$ decrescendo até 2	$\Theta(n)$
3	$A[1] \leftrightarrow A[m]$	$\Theta(n)$
4	MAX-HEAPIFY($A, m - 1, 1$)	$n \cdot O(\log n)$

Consumo de tempo no pior caso? $O(n \log n)$

Complexidade de HEAP-SORT

HEAP-SORT(A, n)		Tempo
1	BUILD-MAX-HEAP(A, n)	$\Theta(n)$
2	para $m \leftarrow n$ decrescendo até 2	$\Theta(n)$
3	$A[1] \leftrightarrow A[m]$	$\Theta(n)$
4	MAX-HEAPIFY($A, m - 1, 1$)	$n \cdot O(\log n)$

Consumo de tempo no pior caso? $O(n \log n)$

Complexidade de HEAP-SORT

HEAP-SORT(A, n)		Tempo
1	BUILD-MAX-HEAP(A, n)	$\Theta(n)$
2	para $m \leftarrow n$ decrescendo até 2	$\Theta(n)$
3	$A[1] \leftrightarrow A[m]$	$\Theta(n)$
4	MAX-HEAPIFY($A, m - 1, 1$)	$n \cdot O(\log n)$

Consumo de tempo no pior caso? $O(n \log n)$

Fila de prioridades

Filas de prioridades

Definição

Uma **fila de prioridades** é um tipo abstrato de dados que consiste de uma coleção S de itens com prioridades associadas e permite as operações

- ▶ $\text{MAXIMUM}(S)$ devolve um elemento de maior prioridade
- ▶ $\text{EXTRACT-MAX}(S)$ remove um elemento de maior prioridade
- ▶ $\text{INCREASE-KEY}(S, x, p)$ **aumenta** a prioridade de x para p
- ▶ $\text{INSERT}(S, x, p)$ insere um elemento x com prioridade p

Filas de prioridades

Definição

Uma **fila de prioridades** é um tipo abstrato de dados que consiste de uma coleção S de itens com prioridades associadas e permite as operações

- ▶ $\text{MAXIMUM}(S)$ devolve um elemento de maior prioridade
- ▶ $\text{EXTRACT-MAX}(S)$ remove um elemento de maior prioridade
- ▶ $\text{INCREASE-KEY}(S, x, p)$ **aumenta** a prioridade de x para p
- ▶ $\text{INSERT}(S, x, p)$ insere um elemento x com prioridade p

Filas de prioridades

Definição

Uma **fila de prioridades** é um tipo abstrato de dados que consiste de uma coleção S de itens com prioridades associadas e permite as operações

- ▶ $\text{MAXIMUM}(S)$ devolve um elemento de maior prioridade
- ▶ $\text{EXTRACT-MAX}(S)$ remove um elemento de maior prioridade
- ▶ $\text{INCREASE-KEY}(S, x, p)$ **aumenta** a prioridade de x para p
- ▶ $\text{INSERT}(S, x, p)$ insere um elemento x com prioridade p

Filas de prioridades

Definição

Uma **fila de prioridades** é um tipo abstrato de dados que consiste de uma coleção S de itens com prioridades associadas e permite as operações

- ▶ $\text{MAXIMUM}(S)$ devolve um elemento de maior prioridade
- ▶ $\text{EXTRACT-MAX}(S)$ remove um elemento de maior prioridade
- ▶ $\text{INCREASE-KEY}(S, x, p)$ **aumenta** a prioridade de x para p
- ▶ $\text{INSERT}(S, x, p)$ insere um elemento x com prioridade p

Filas de prioridades

Definição

Uma **fila de prioridades** é um tipo abstrato de dados que consiste de uma coleção S de itens com prioridades associadas e permite as operações

- ▶ $\text{MAXIMUM}(S)$ devolve um elemento de maior prioridade
- ▶ $\text{EXTRACT-MAX}(S)$ remove um elemento de maior prioridade
- ▶ $\text{INCREASE-KEY}(S, x, p)$ **aumenta** a prioridade de x para p
- ▶ $\text{INSERT}(S, x, p)$ insere um elemento x com prioridade p

Implementação com max-heap

```
HEAP-MAX( $A, n$ )  
1  devolva  $A[1]$ 
```

Complexidade de tempo: $\Theta(1)$

```
HEAP-EXTRACT-MAX( $A, n$ )  
1   $A[1] \leftarrow A[n]$   
2   $n \leftarrow n - 1$   
3  MAX-HEAPIFY( $A, n, 1$ )
```

Complexidade de tempo: $O(\log n)$

Implementação com max-heap

```
HEAP-MAX( $A, n$ )  
1  devolva  $A[1]$ 
```

Complexidade de tempo: $\Theta(1)$

```
HEAP-EXTRACT-MAX( $A, n$ )  
1   $A[1] \leftarrow A[n]$   
2   $n \leftarrow n - 1$   
3  MAX-HEAPIFY( $A, n, 1$ )
```

Complexidade de tempo: $O(\log n)$

Implementação com max-heap

```
HEAP-MAX( $A, n$ )  
1  devolva  $A[1]$ 
```

Complexidade de tempo: $\Theta(1)$

```
HEAP-EXTRACT-MAX( $A, n$ )  
1   $A[1] \leftarrow A[n]$   
2   $n \leftarrow n - 1$   
3  MAX-HEAPIFY( $A, n, 1$ )
```

Complexidade de tempo: $O(\log n)$

Implementação com max-heap

```
HEAP-MAX( $A, n$ )  
1  devolva  $A[1]$ 
```

Complexidade de tempo: $\Theta(1)$

```
HEAP-EXTRACT-MAX( $A, n$ )  
1   $A[1] \leftarrow A[n]$   
2   $n \leftarrow n - 1$   
3  MAX-HEAPIFY( $A, n, 1$ )
```

Complexidade de tempo: $O(\log n)$

Implementação com max-heap

HEAP-INCREASE-KEY($A, i, chave$)

```
1   $A[i] \leftarrow chave$   
2  enquanto  $i > 1$  e  $A[\lfloor i/2 \rfloor] < A[i]$   
3       $A[i] \leftrightarrow A[\lfloor i/2 \rfloor]$   
4       $i \leftarrow \lfloor i/2 \rfloor$ 
```

Complexidade de tempo: $O(\log n)$

MAX-HEAP-INSERT($A, n, chave$)

```
1   $n \leftarrow n + 1$   
2   $A[n] \leftarrow -\infty$   
3  HEAP-INCREASE-KEY( $A, n, chave$ )
```

Complexidade de tempo: $O(\log n)$

Implementação com max-heap

HEAP-INCREASE-KEY($A, i, chave$)

```
1   $A[i] \leftarrow chave$   
2  enquanto  $i > 1$  e  $A[\lfloor i/2 \rfloor] < A[i]$   
3       $A[i] \leftrightarrow A[\lfloor i/2 \rfloor]$   
4       $i \leftarrow \lfloor i/2 \rfloor$ 
```

Complexidade de tempo: $O(\log n)$

MAX-HEAP-INSERT($A, n, chave$)

```
1   $n \leftarrow n + 1$   
2   $A[n] \leftarrow -\infty$   
3  HEAP-INCREASE-KEY( $A, n, chave$ )
```

Complexidade de tempo: $O(\log n)$

Implementação com max-heap

HEAP-INCREASE-KEY($A, i, chave$)

```
1   $A[i] \leftarrow chave$   
2  enquanto  $i > 1$  e  $A[\lfloor i/2 \rfloor] < A[i]$   
3       $A[i] \leftrightarrow A[\lfloor i/2 \rfloor]$   
4       $i \leftarrow \lfloor i/2 \rfloor$ 
```

Complexidade de tempo: $O(\log n)$

MAX-HEAP-INSERT($A, n, chave$)

```
1   $n \leftarrow n + 1$   
2   $A[n] \leftarrow -\infty$   
3  HEAP-INCREASE-KEY( $A, n, chave$ )
```

Complexidade de tempo: $O(\log n)$

Implementação com max-heap

HEAP-INCREASE-KEY($A, i, chave$)

```
1   $A[i] \leftarrow chave$   
2  enquanto  $i > 1$  e  $A[\lfloor i/2 \rfloor] < A[i]$   
3       $A[i] \leftrightarrow A[\lfloor i/2 \rfloor]$   
4       $i \leftarrow \lfloor i/2 \rfloor$ 
```

Complexidade de tempo: $O(\log n)$

MAX-HEAP-INSERT($A, n, chave$)

```
1   $n \leftarrow n + 1$   
2   $A[n] \leftarrow -\infty$   
3  HEAP-INCREASE-KEY( $A, n, chave$ )
```

Complexidade de tempo: $O(\log n)$