

Heapsort

CLRS 6

Heap

Um vetor $A[1..m]$ é um (max-)heap se

$$A[\lfloor i/2 \rfloor] \geq A[i]$$

para todo $i = 2, 3, \dots, m$.

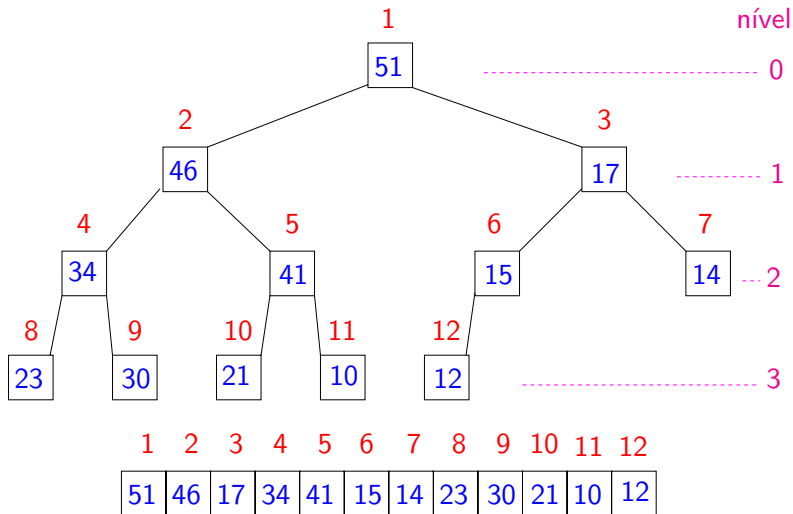
De uma forma mais geral, $A[j..m]$ é um heap se

$$A[\lfloor i/2 \rfloor] \geq A[i]$$

para todo $i = 2j, 2j + 1, 4j, \dots, 4j + 3, 8j, \dots, 8j + 7, \dots$

Neste caso também diremos que a subárvore com raiz j é um heap.

Exemplo



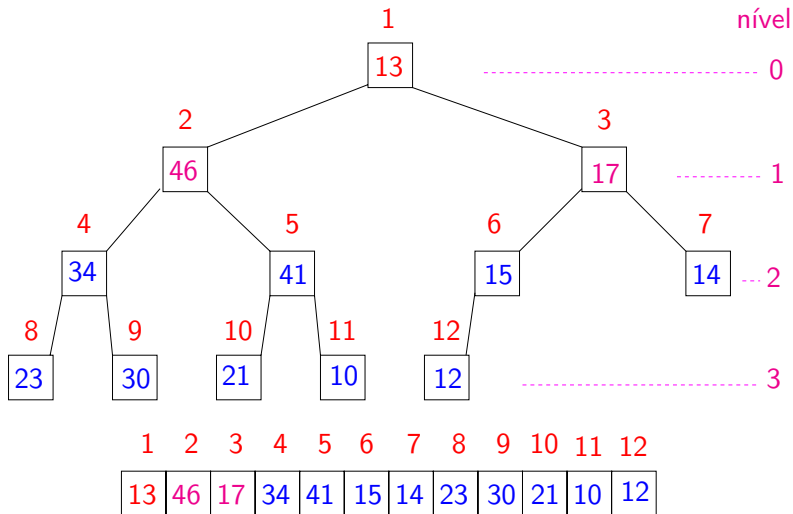
Desce-Heap

Recebe $A[1..m]$ e $i \geq 1$ tais que subárvores com raiz $2i$ e $2i+1$ são heaps e **rearranja** A de modo que subárvore com raiz i seja heap.

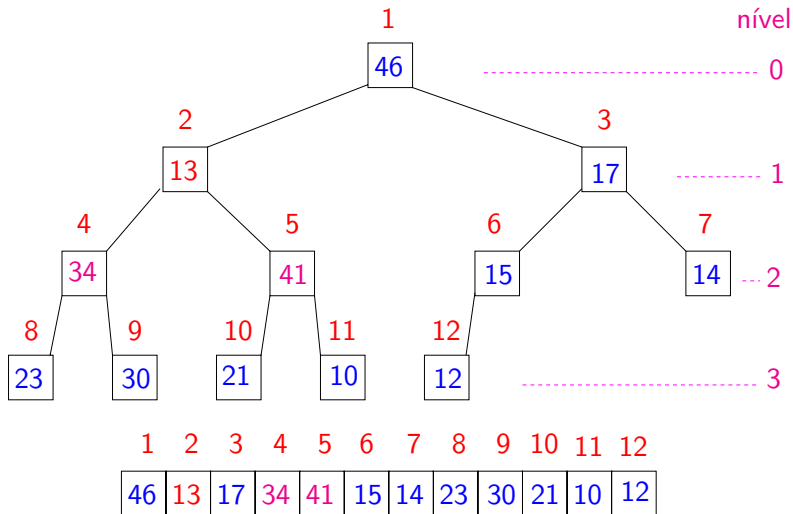
DESCE-HEAP (A, m, i)

```
1   $e \leftarrow 2i$ 
2   $d \leftarrow 2i + 1$ 
3  se  $e \leq m$  e  $A[e] > A[i]$ 
4      então  $maior \leftarrow e$ 
5      senão  $maior \leftarrow i$ 
6  se  $d \leq m$  e  $A[d] > A[maior]$ 
7      então  $maior \leftarrow d$ 
8  se  $maior \neq i$ 
9      então  $A[i] \leftrightarrow A[maior]$ 
10     DESCE-HEAP ( $A, m, maior$ )
```

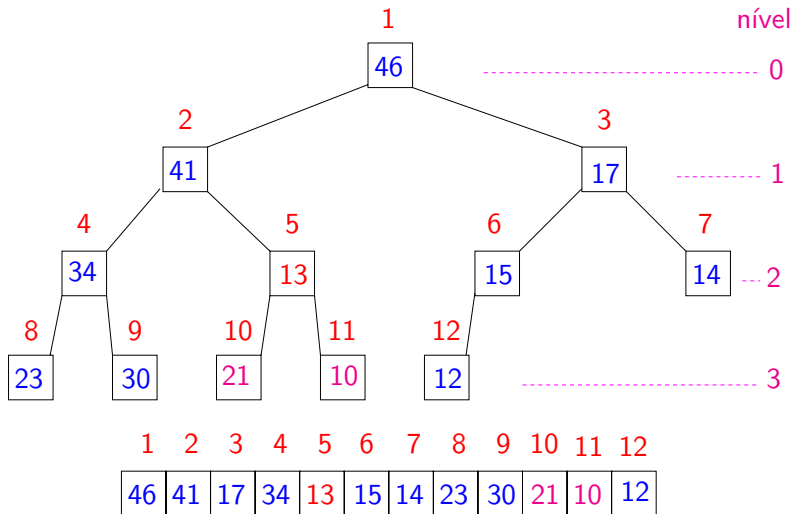
Simulação



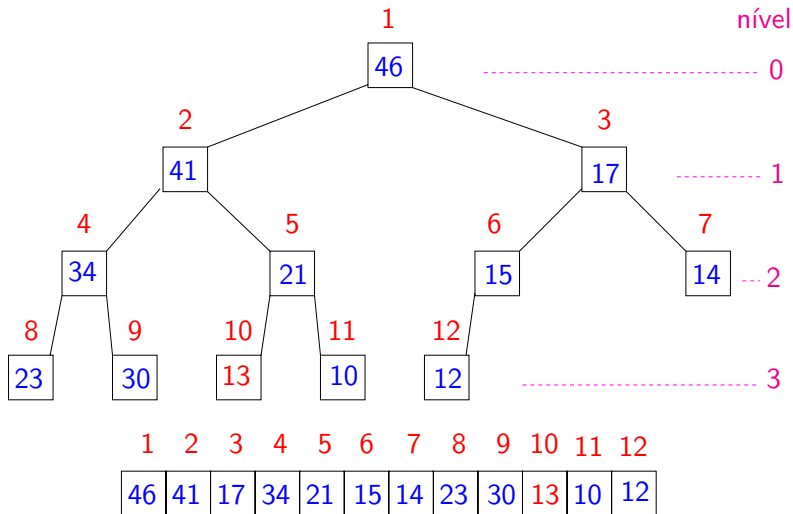
Simulação



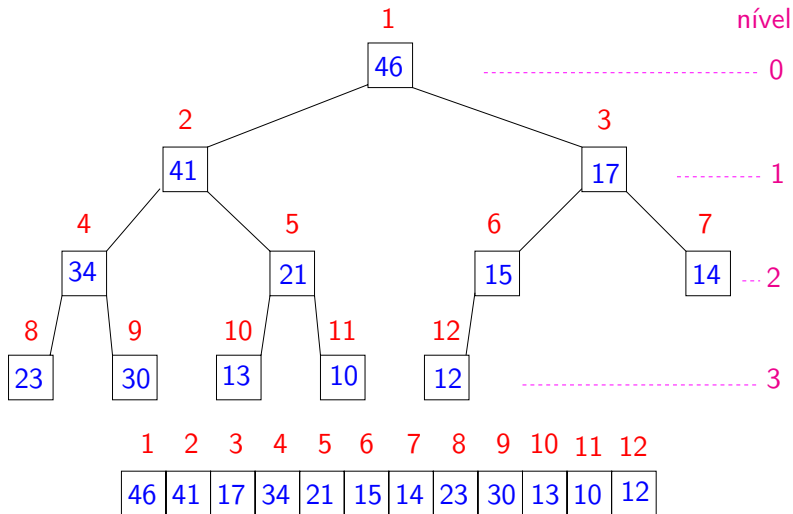
Simulação



Simulação



Simulação



Consumo de tempo

$h :=$ altura de $i = \lfloor \lg \frac{m}{i} \rfloor$

$T(h) :=$ consumo de tempo no pior caso

linha	todas as execuções da linha
1-3	$= \Theta(1)$
4-5	$= \Theta(1)$
6	$= \Theta(1)$
7	$= O(1)$
8	$= \Theta(1)$
9	$= O(1)$
10	$\leq T(h-1)$
total	$\leq T(h-1) + \Theta(1)$

Consumo de tempo

$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

$$T(h) \leq T(h-1) + \Theta(1),$$

pois altura de *maior* é $h-1$.

Consumo de tempo

$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

$$T(h) \leq T(h-1) + \Theta(1),$$

pois altura de *maior* é $h-1$.

Solução assintótica: $T(h)$ é ???.

Consumo de tempo

$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

$$T(h) \leq T(h-1) + \Theta(1),$$

pois altura de *maior* é $h-1$.

Solução assintótica: $T(h)$ é $O(h)$.

Consumo de tempo

$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

$$T(h) \leq T(h-1) + \Theta(1),$$

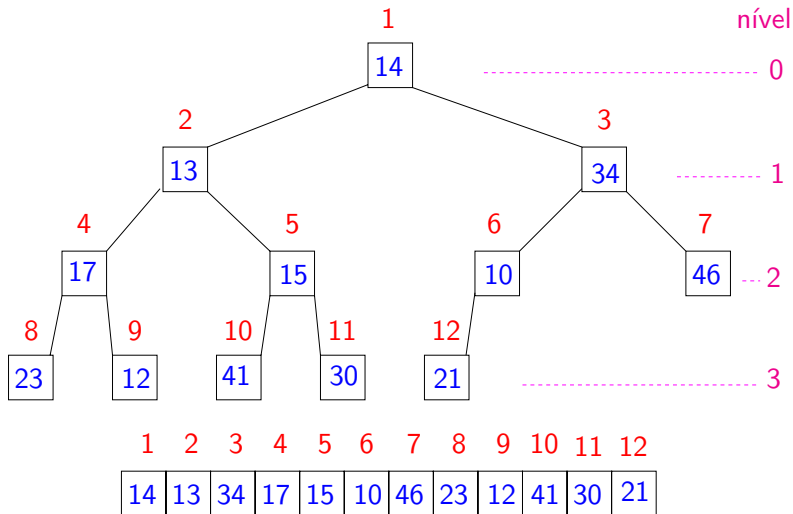
pois altura de *maior* é $h-1$.

Solução assintótica: $T(h)$ é $O(h)$.

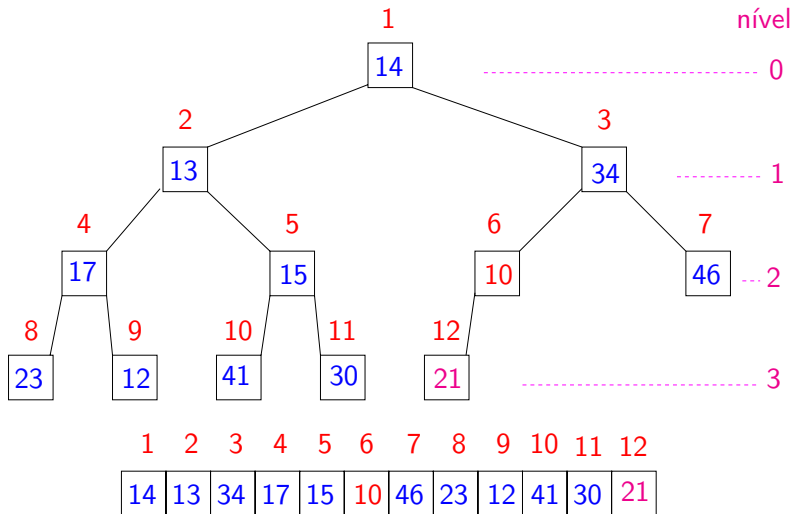
Como $h \leq \lg m$, podemos dizer que:

O consumo de tempo do algoritmo **DESCE-HEAP** é $O(\lg m)$
(ou melhor ainda, $O(h)$).

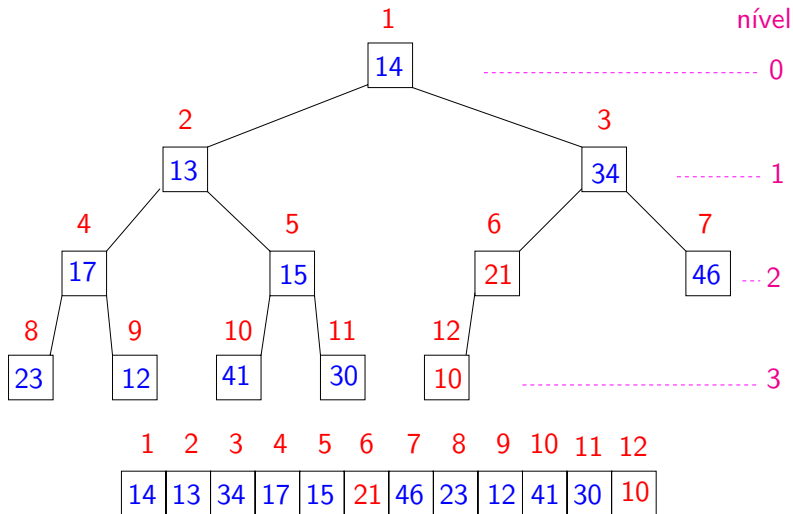
Construção de um heap



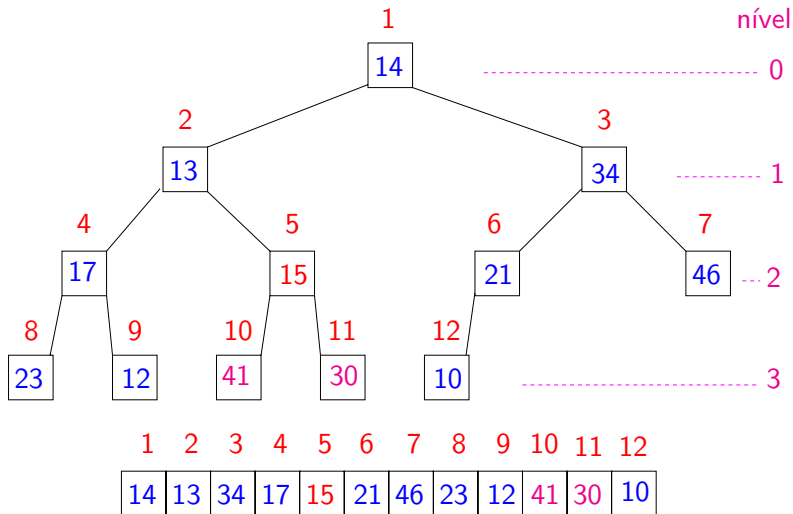
Construção de um heap



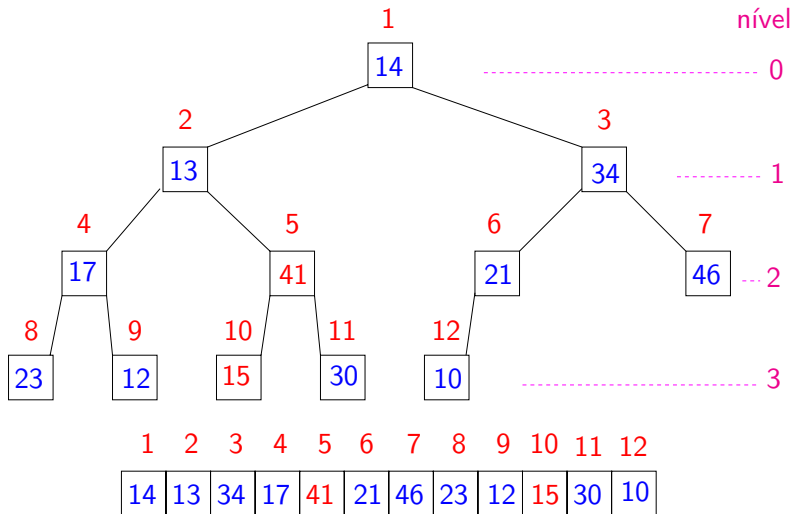
Construção de um heap



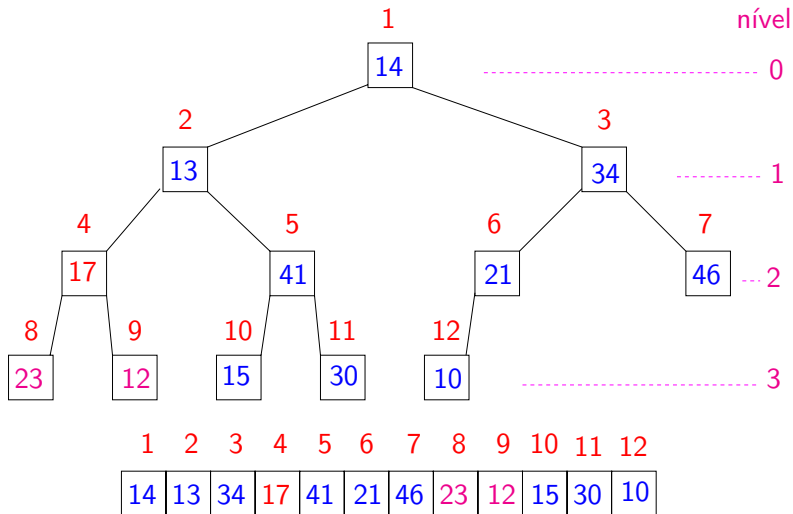
Construção de um heap



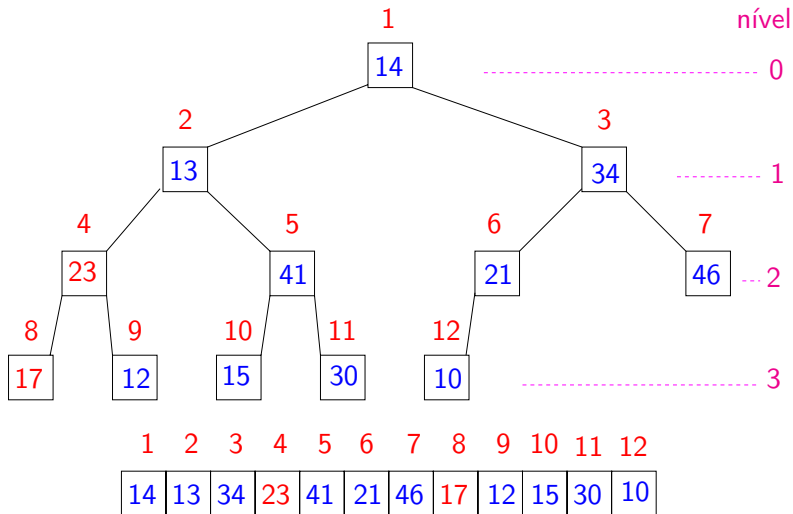
Construção de um heap



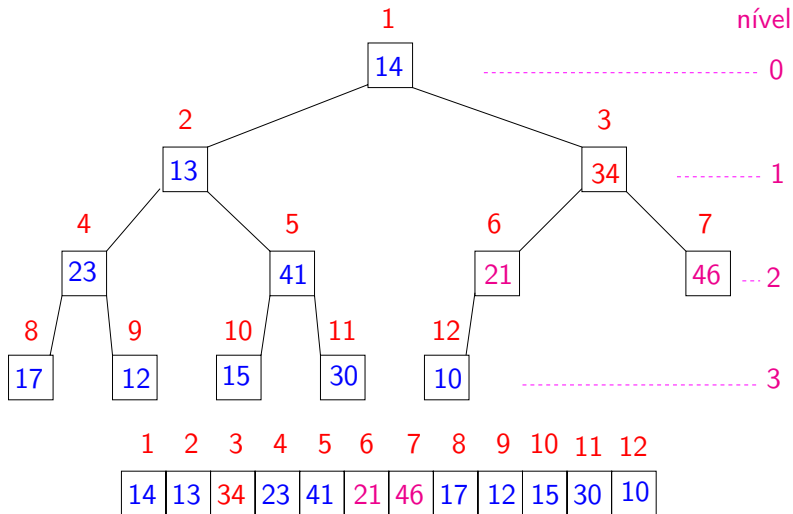
Construção de um heap



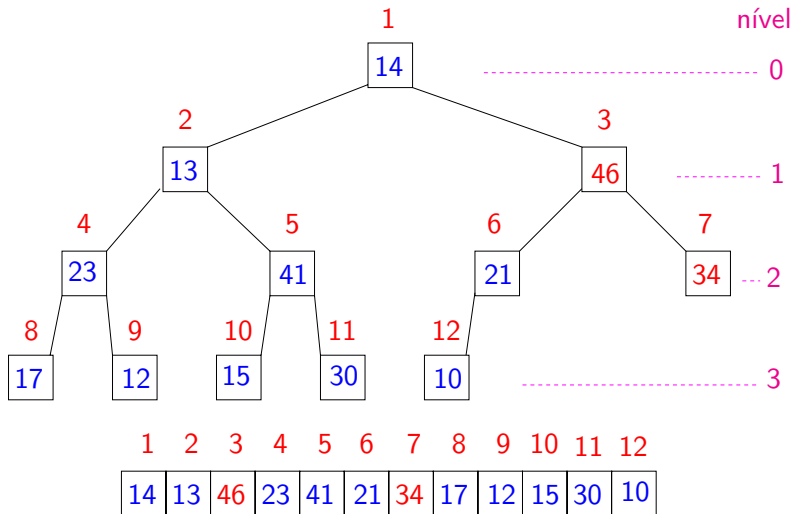
Construção de um heap



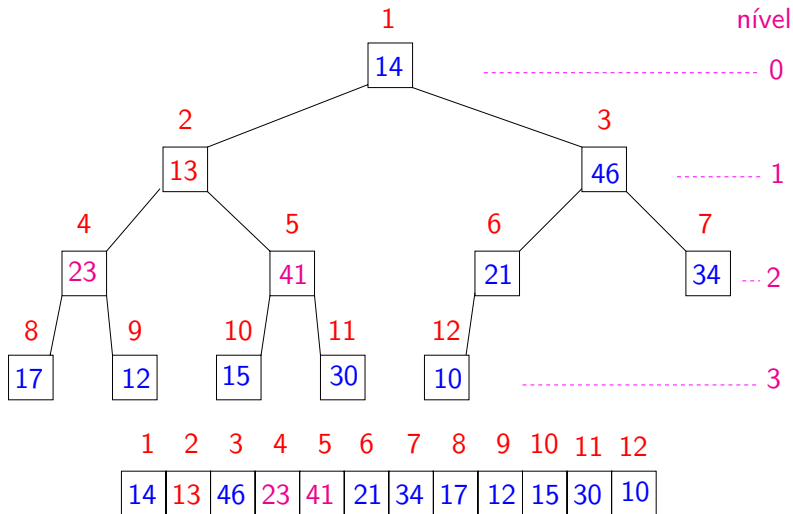
Construção de um heap



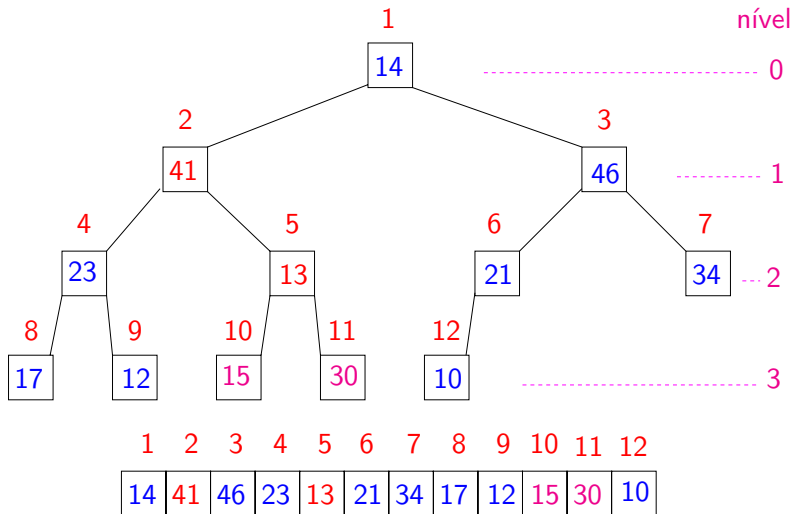
Construção de um heap



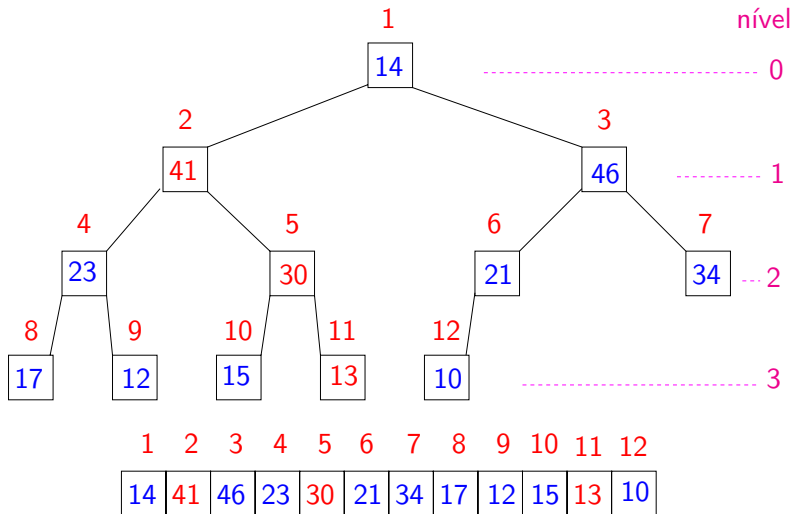
Construção de um heap



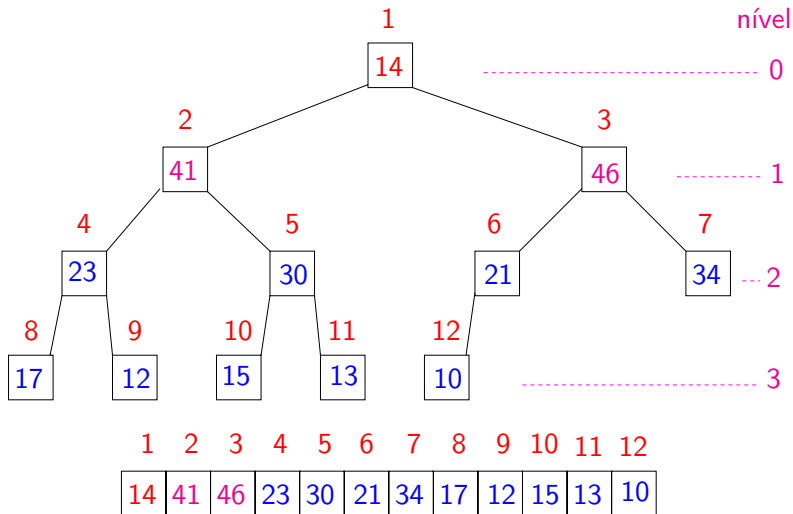
Construção de um heap



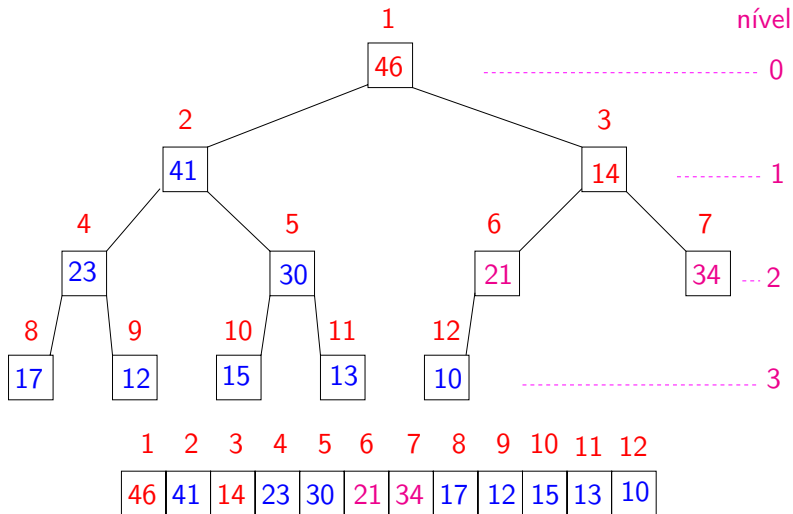
Construção de um heap



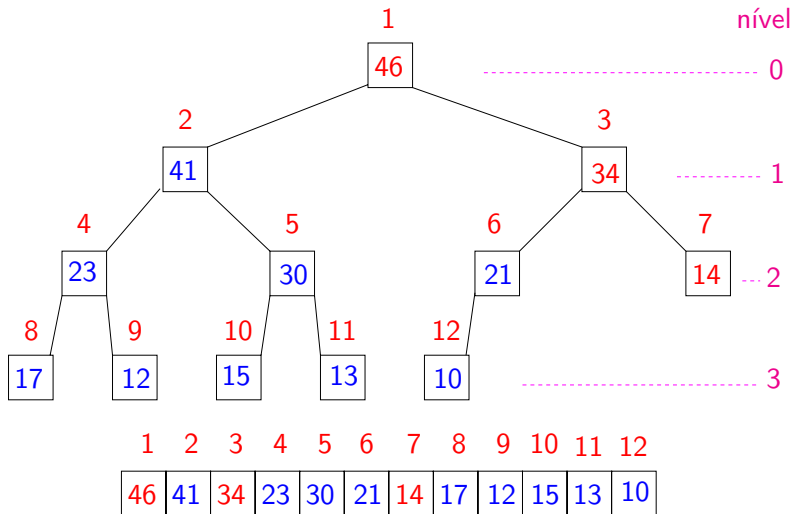
Construção de um heap



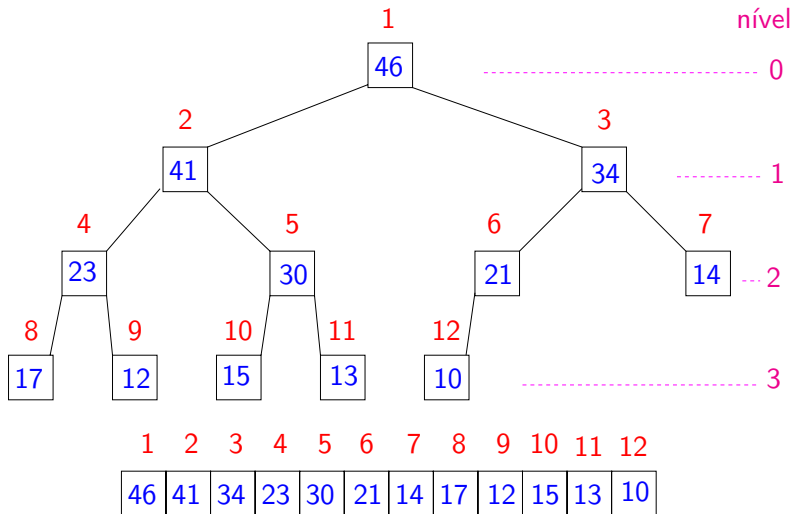
Construção de um heap



Construção de um heap



Construção de um heap



Construção de um heap

Recebe um vetor $A[1..n]$ e rearranja A para que seja heap.

CONSTRÓI-HEAP (A, n)

2 para $i \leftarrow \lfloor n/2 \rfloor$ decrescendo até 1 faça

3 DESCE-HEAP(A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i + 1, \dots, n$ são raízes de heaps.

$T(n) :=$ consumo de tempo no pior caso

Construção de um heap

Recebe um vetor $A[1..n]$ e rearranja A para que seja heap.

CONSTRÓI-HEAP (A, n)

2 para $i \leftarrow \lfloor n/2 \rfloor$ decrescendo até 1 faça

3 DESCE-HEAP(A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i + 1, \dots, n$ são raízes de heaps.

$T(n)$:= consumo de tempo no pior caso

Análise grosseira: $T(n)$ é $\frac{n}{2} O(\lg n) = O(n \lg n)$.

Análise mais cuidadosa: $T(n)$ é ????

$T(n)$ é $O(n)$

Prova:

Consumo de **DESCE-HEAP**(A, n, i) é proporcional a h , logo

$$T(n) = \sum_{h=1}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} h$$

$T(n)$ é $O(n)$

Prova:

Consumo de **DESCE-HEAP**(A, n, i) é proporcional a h , logo

$$\begin{aligned} T(n) &= \sum_{h=1}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} h \\ &\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h} h \end{aligned}$$

$T(n)$ é $O(n)$

Prova:

Consumo de **DESCE-HEAP**(A, n, i) é proporcional a h , logo

$$\begin{aligned} T(n) &= \sum_{h=1}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} h \\ &\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h} h \\ &\leq n \left(\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \dots + \frac{\lfloor \lg n \rfloor}{2^{\lfloor \lg n \rfloor}} \right) \end{aligned}$$

$T(n)$ é $O(n)$

Prova:

Consumo de **DESCE-HEAP**(A, n, i) é proporcional a h , logo

$$\begin{aligned}T(n) &= \sum_{h=1}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} h \\ &\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h} h \\ &\leq n \left(\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \dots + \frac{\lfloor \lg n \rfloor}{2^{\lfloor \lg n \rfloor}} \right) \\ &< n \frac{1/2}{(1 - 1/2)^2} = 2n.\end{aligned}$$

$T(n)$ é $O(n)$

Prova:

O consumo de tempo de **DESCE-HEAP**(A, n, i) é $O(h)$, onde h é a altura da árvore de raiz i . Logo,

$$T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} O(h)$$

$T(n)$ é $O(n)$

Prova:

O consumo de tempo de **DESCE-HEAP**(A, n, i) é $O(h)$, onde h é a altura da árvore de raiz i . Logo,

$$\begin{aligned} T(n) &= \sum_{h=0}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} O(h) \\ &= O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) \end{aligned}$$

$T(n)$ é $O(n)$

Prova:

O consumo de tempo de **DESCE-HEAP**(A, n, i) é $O(h)$, onde h é a altura da árvore de raiz i . Logo,

$$\begin{aligned}T(n) &= \sum_{h=0}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} O(h) \\&= O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) \\&= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right)\end{aligned}$$

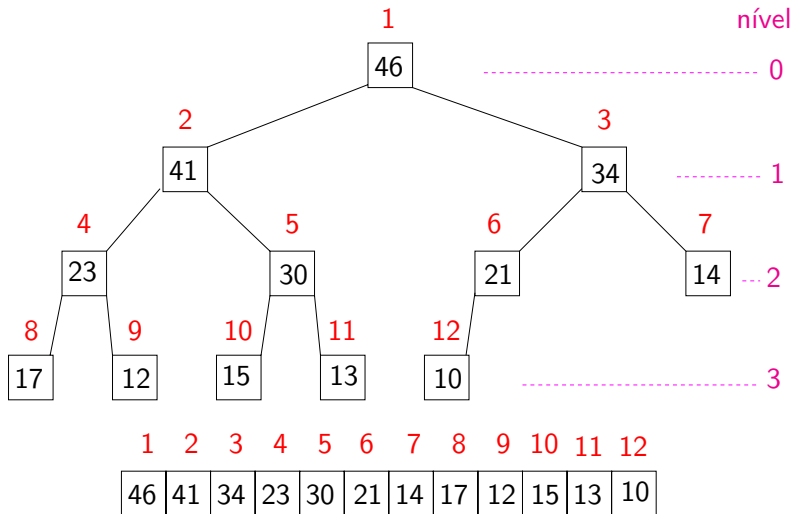
$T(n)$ é $O(n)$

Prova:

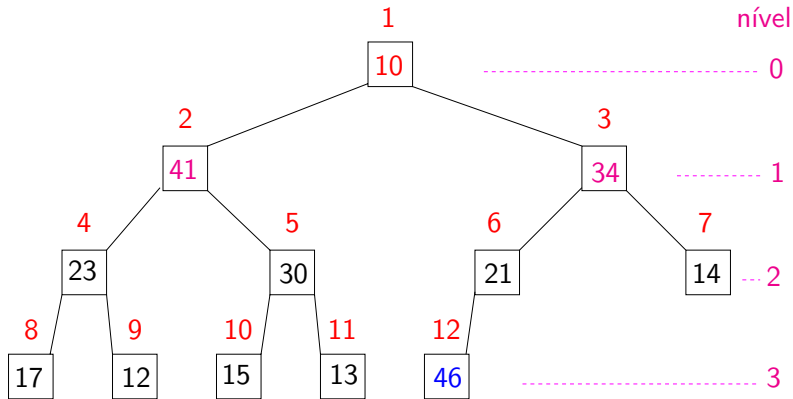
O consumo de tempo de **DESCE-HEAP**(A, n, i) é $O(h)$, onde h é a altura da árvore de raiz i . Logo,

$$\begin{aligned} T(n) &= \sum_{h=0}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} O(h) \\ &= O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) \\ &= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \\ &= O\left(n \frac{1/2}{(1 - 1/2)^2}\right) = O(2n) = O(n). \end{aligned}$$

Heap sort

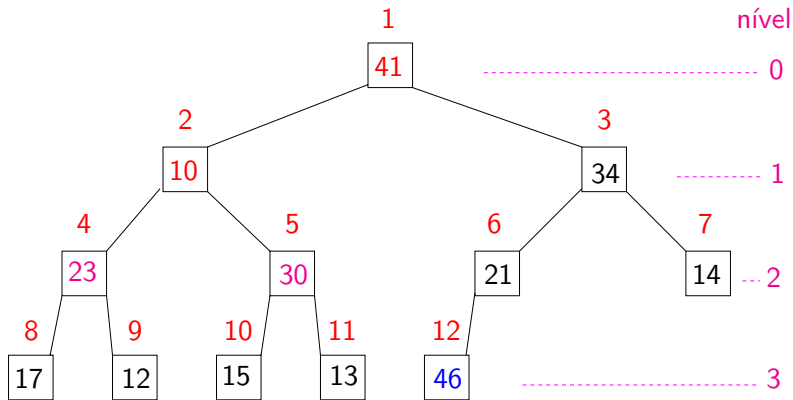


Heap sort



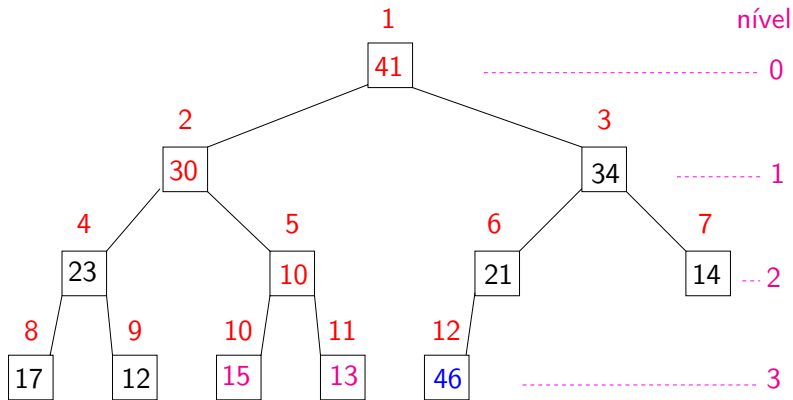
1	2	3	4	5	6	7	8	9	10	11	12
10	41	34	23	30	21	14	17	12	15	13	46

Heap sort



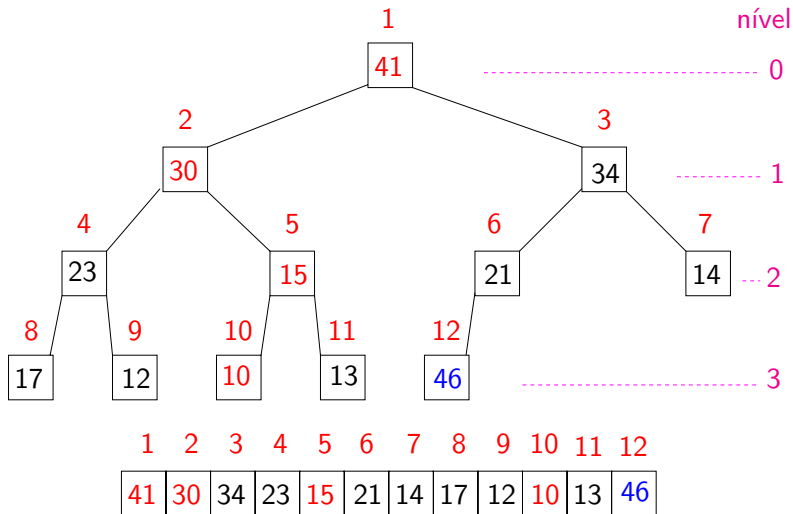
1	2	3	4	5	6	7	8	9	10	11	12
41	10	34	23	30	21	14	17	12	15	13	46

Heap sort

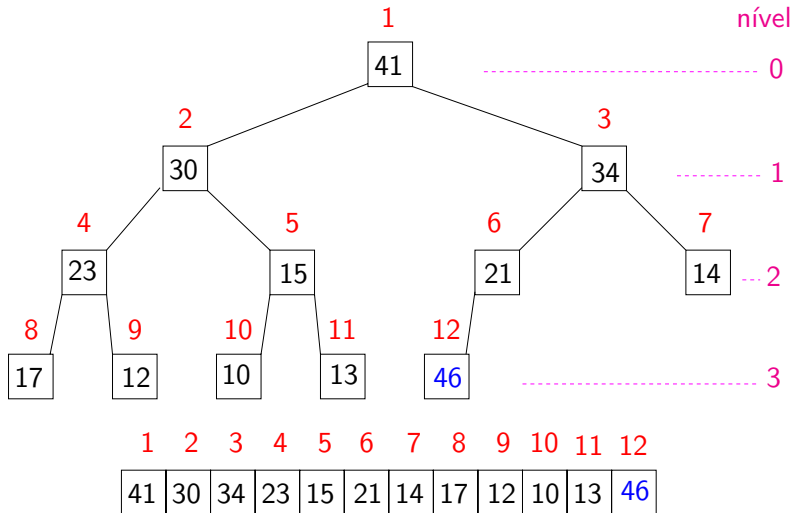


1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	10	21	14	17	12	15	13	46

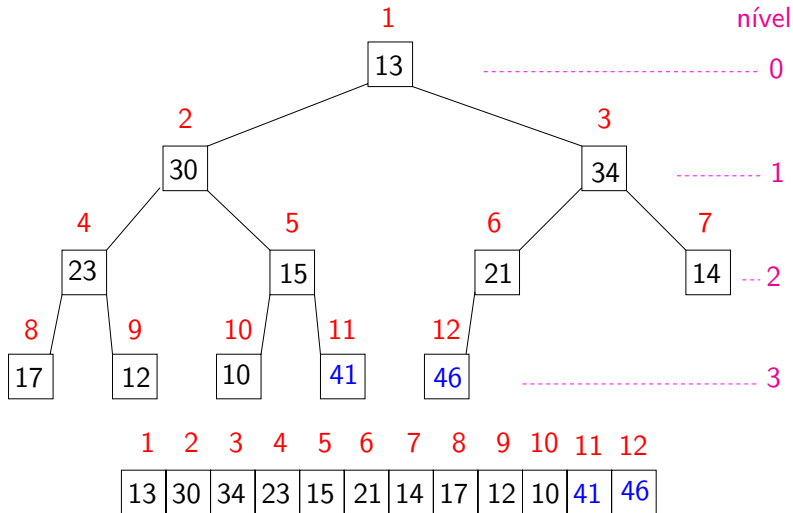
Heap sort



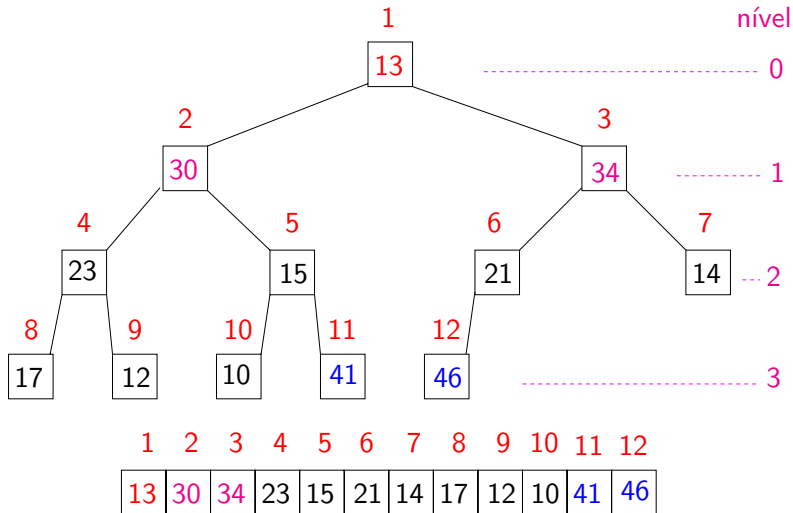
Heap sort



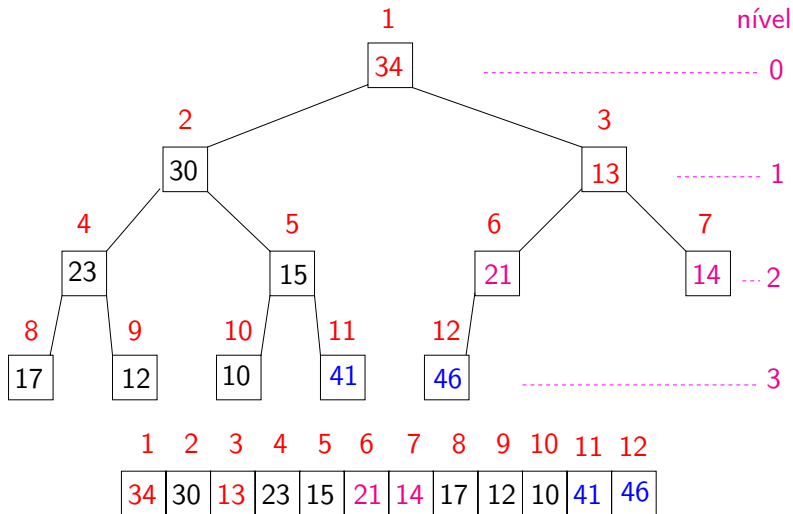
Heap sort



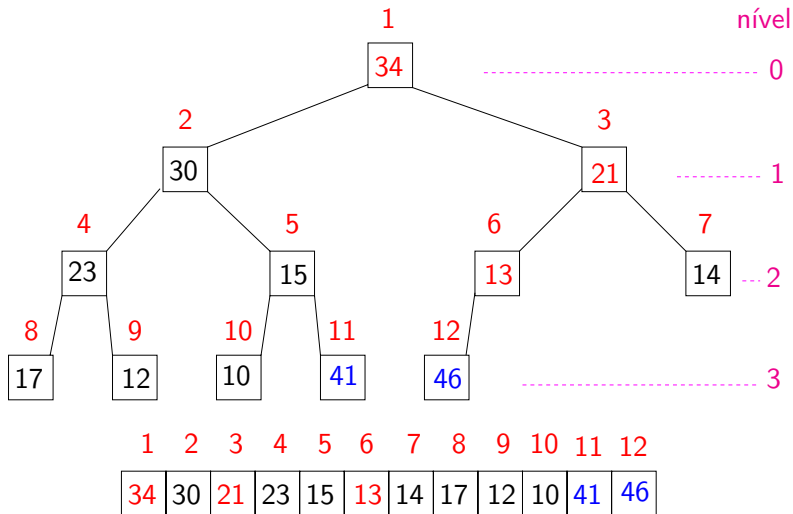
Heap sort



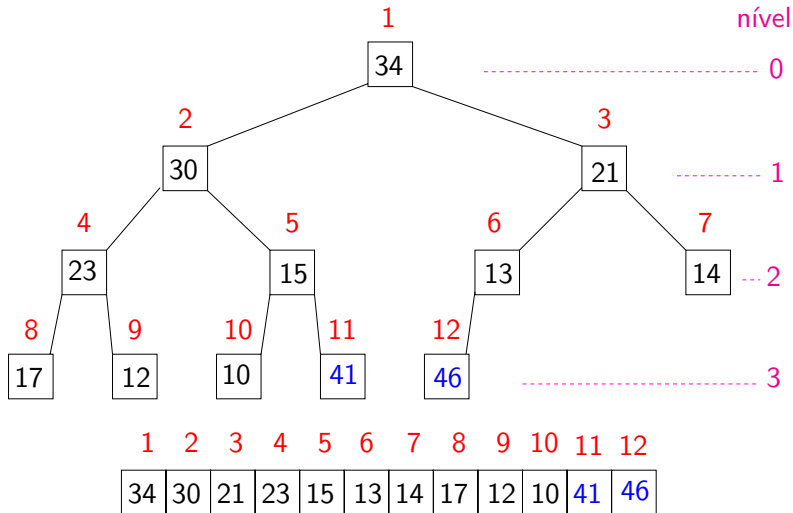
Heap sort



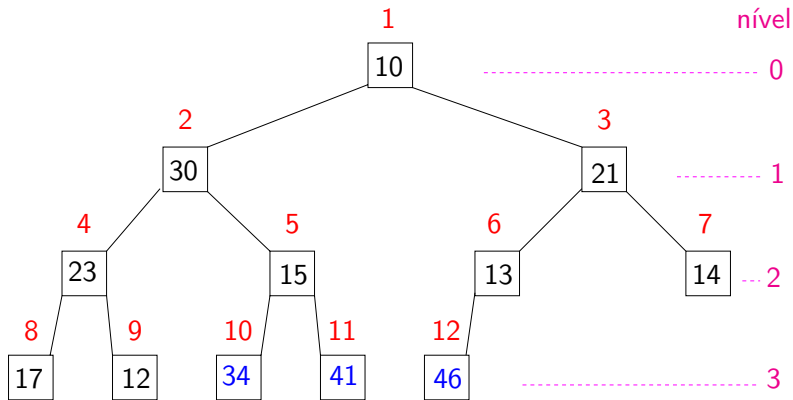
Heap sort



Heap sort



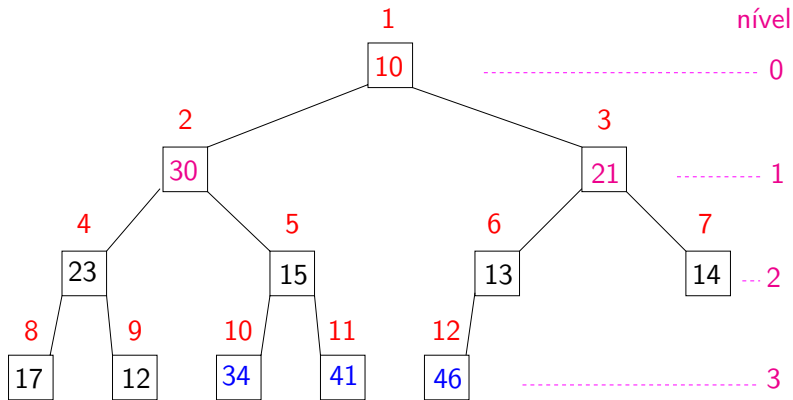
Heap sort



1 2 3 4 5 6 7 8 9 10 11 12

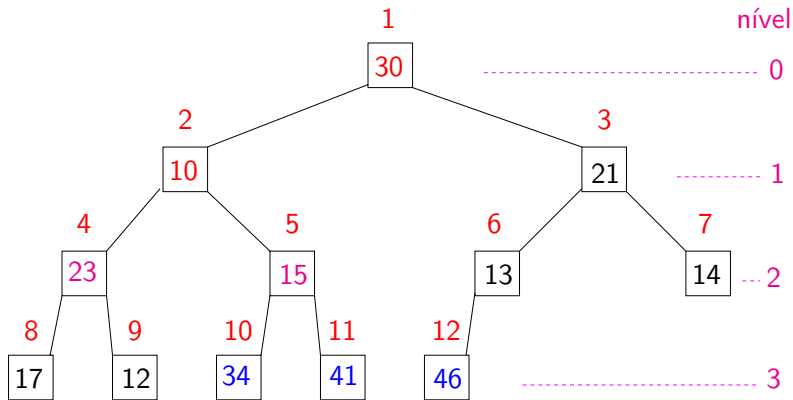
10	30	21	23	15	13	14	17	12	34	41	46
----	----	----	----	----	----	----	----	----	----	----	----

Heap sort



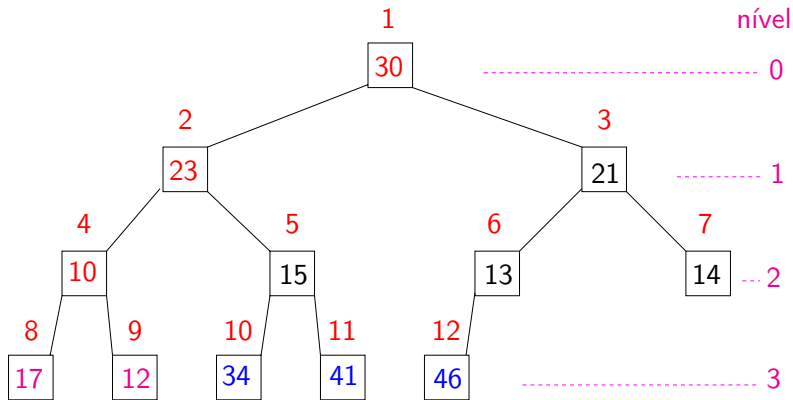
1	2	3	4	5	6	7	8	9	10	11	12
10	30	21	23	15	13	14	17	12	34	41	46

Heap sort



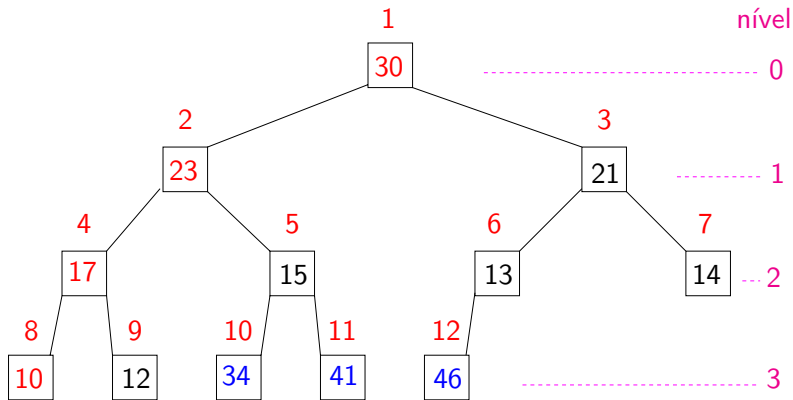
1	2	3	4	5	6	7	8	9	10	11	12
30	10	21	23	15	13	14	17	12	34	41	46

Heap sort



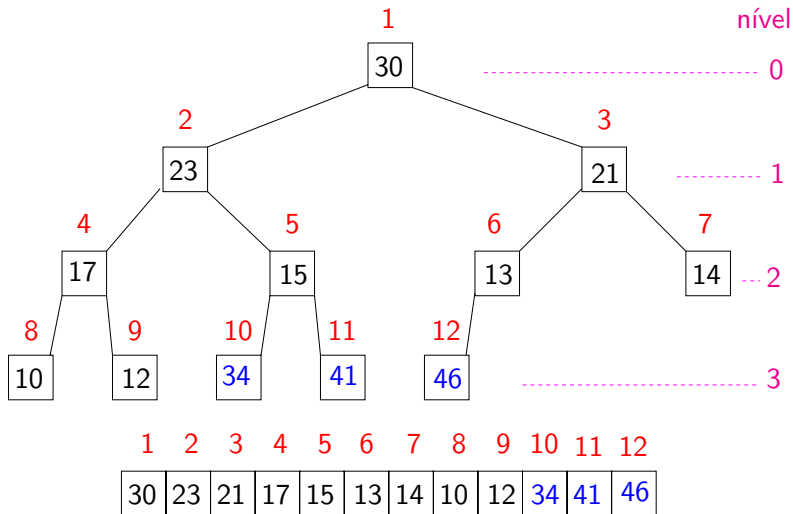
1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	10	15	13	14	17	12	34	41	46

Heap sort

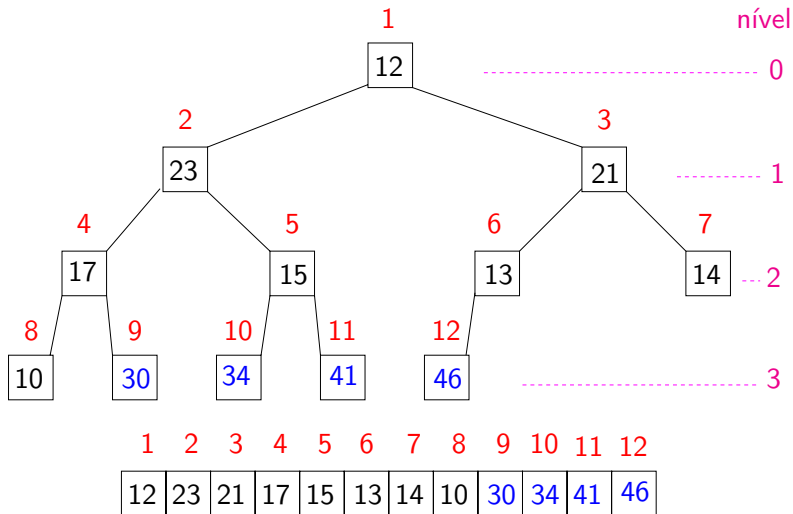


1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	17	15	13	14	10	12	34	41	46

Heap sort



Heap sort



Heap sort

Algoritmo rearranja $A[1..n]$ em ordem crescente.

```
HEAPSORT ( $A, n$ )
0  CONSTRÓI-HEAP ( $A, n$ )  ▷ pré-processamento
1   $m \leftarrow n$ 
2  para  $i \leftarrow n$  decrescendo até 2 faça
3       $A[1] \leftrightarrow A[i]$ 
4       $m \leftarrow m - 1$ 
5      DESCE-HEAP ( $A, m, 1$ )
```

Relações invariantes: Na linha 2 vale que:

(i0) $A[m + 1..n]$ é crescente;

(i1) $A[1..m] \leq A[m + 1]$;

(i2) $A[1..m]$ é um heap.

Consumo de tempo

linha	todas as execuções da linha
0	$= \Theta(n)$
1	$= \Theta(1)$
2	$= \Theta(n)$
3	$= \Theta(n)$
4	$= \Theta(n)$
5	$= nO(\lg n)$
total	$= nO(\lg n) + \Theta(n) = O(n \lg n)$

O consumo de tempo do algoritmo **HEAPSORT** é $O(n \lg n)$.

Limites inferiores

CLRS 8.1

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1..n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $O(n \lg n)$.

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1..n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $O(n \lg n)$.

Existe algoritmo **assintoticamente** melhor?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1..n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $O(n \lg n)$.

Existe algoritmo **assintoticamente** melhor?

NÃO, se o algoritmo é baseado em **comparações**.

Prova?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1..n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $O(n \lg n)$.

Existe algoritmo **assintoticamente** melhor?

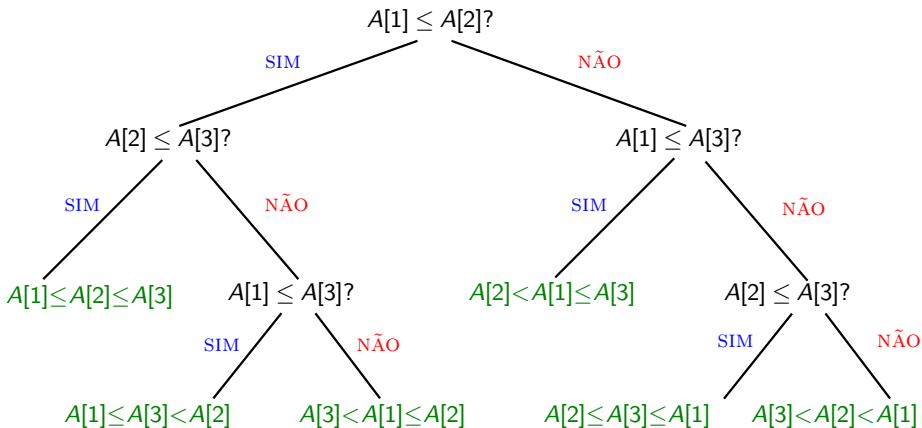
NÃO, se o algoritmo é baseado em **comparações**.

Prova?

Qualquer algoritmo baseado em comparações é uma “**árvore de decisão**”.

Exemplo

ORDENA-POR-INSERÇÃO ($A[1..3]$):



Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Resposta: **altura**, h , da árvore de decisão.

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Resposta: **altura**, h , da árvore de decisão.

Todas as $n!$ permutações de $1, \dots, n$ devem ser folhas.

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Resposta: **altura**, h , da árvore de decisão.

Todas as $n!$ permutações de $1, \dots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Resposta: **altura**, h , da árvore de decisão.

Todas as $n!$ permutações de $1, \dots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h . A afirmação vale para $h = 0$.

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Resposta: **altura**, h , da árvore de decisão.

Todas as $n!$ permutações de $1, \dots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h . A afirmação vale para $h = 0$.

Suponha que a afirmação vale para toda árvore binária de altura menor que h , para $h \geq 1$.

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Resposta: **altura**, h , da árvore de decisão.

Todas as $n!$ permutações de $1, \dots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h . A afirmação vale para $h = 0$.

Suponha que a afirmação vale para toda árvore binária de altura menor que h , para $h \geq 1$.

Número de folhas de árvore de altura h é a soma do número de folhas das subárvores, que têm altura $\leq h - 1$.

Limite inferior

Considere uma **árvore de decisão** para $A[1..n]$.

Número de comparações, no pior caso?

Resposta: **altura**, h , da árvore de decisão.

Todas as $n!$ permutações de $1, \dots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h . A afirmação vale para $h = 0$.

Suponha que a afirmação vale para toda árvore binária de altura menor que h , para $h \geq 1$.

Número de folhas de árvore de altura h é a soma do número de folhas das subárvores, que têm altura $\leq h - 1$.

Logo, o número de folhas de uma árvore de altura h é

$$\leq 2 \times 2^{h-1} = 2^h.$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$n! \geq \prod_{i=\frac{n}{2}}^n i \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$n! \geq \prod_{i=\frac{n}{2}}^n i \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

Portanto,

$$h \geq \lg(n!) \geq \frac{1}{2} n(\lg n - 1).$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$n! \geq \prod_{i=\frac{n}{2}}^n i \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

Portanto,

$$h \geq \lg(n!) \geq \frac{1}{2} n(\lg n - 1).$$

Mais precisamente, a fórmula de Stirling diz que

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$n! \geq \prod_{i=\frac{n}{2}}^n i \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

Portanto,

$$h \geq \lg(n!) \geq \frac{1}{2} n(\lg n - 1).$$

Mais precisamente, a fórmula de Stirling diz que

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Disso, temos que $h \geq \lg(n!) \geq \lg\left(\frac{n}{e}\right)^n = n(\lg n - \lg e)$.

Conclusão

Todo algoritmo de ordenação
baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso.

Portanto...

MERGESORT e HEAPSORT são assintoticamente ótimos!