Heapsort

CLRS 6

Heap

Um vetor $A[1 \mathrel{..} m]$ é um (max-)heap se

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

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

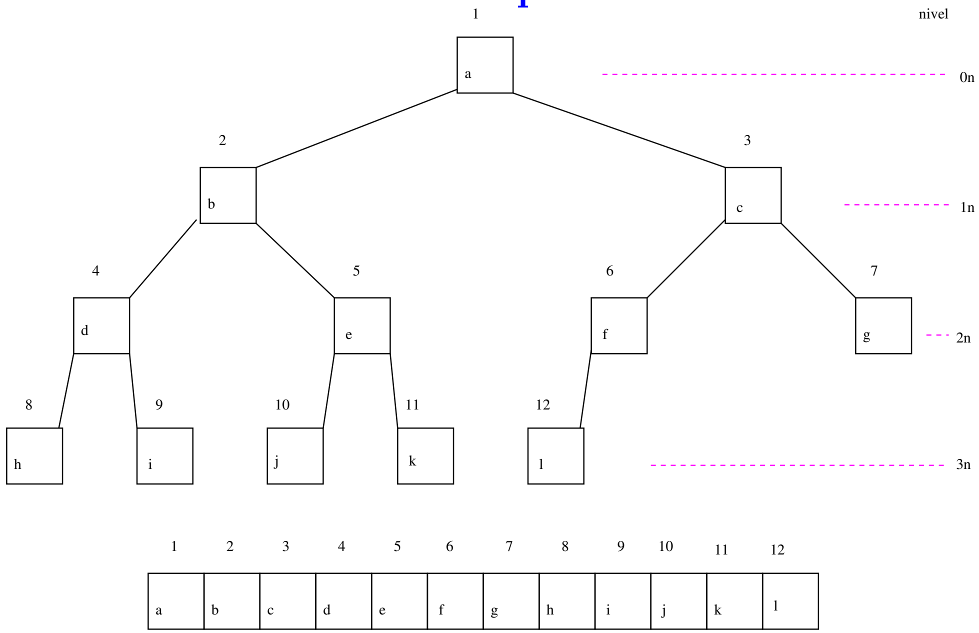
De uma forma mais geral, $A[j \mathrel{..} m]$ é um heap se

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

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

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

Exemplo

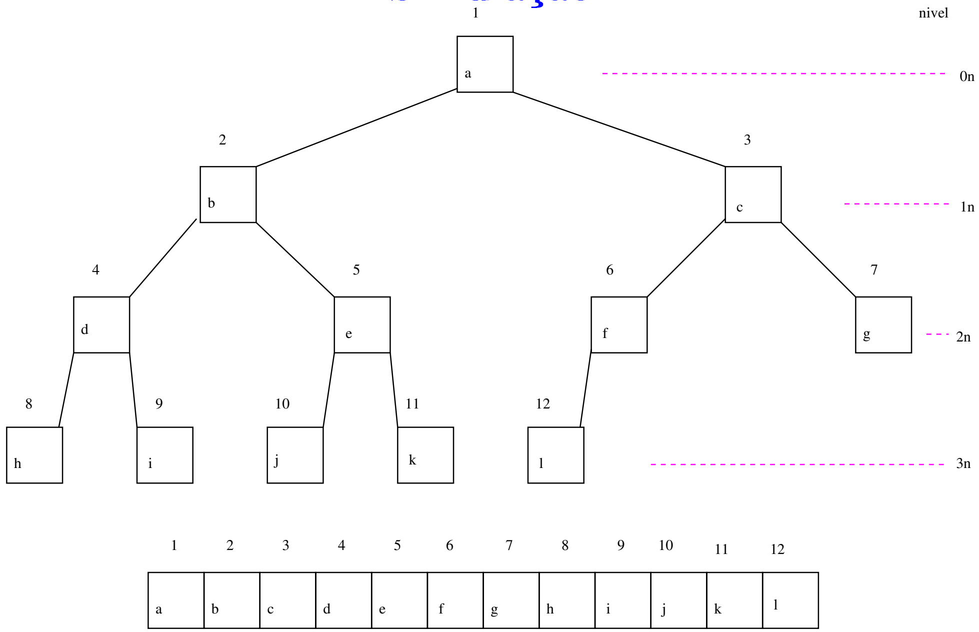


Desce-Heap

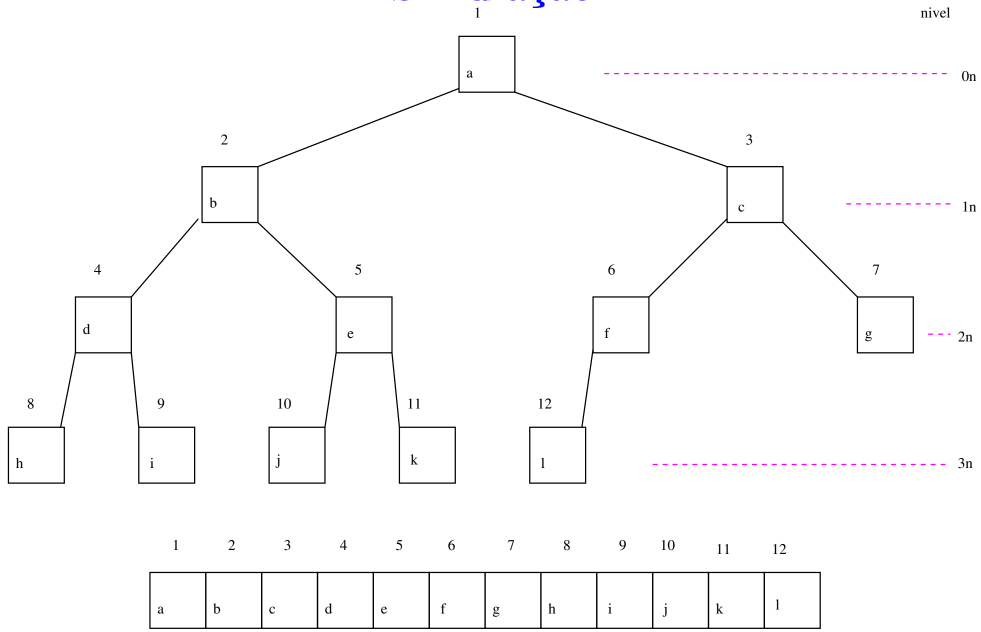
Recebe $A[1 \mathrel{..} m]$ e $i \geq 1$ tais que subárvores com raiz $2i$ e $2i+1$ são heaps e rearranja A de modo que subárvore com raiz i seja heap.

```
DESCE-HEAP (A, m, i)
   e ← 2i
   d ← 2i + 1
   se e ≤ m e A[e] > A[i]
       então maior ← e
       senão maior ← i
   se d ≤ m e A[d] > A[maior]
       então maior ← d
   se maior ≠ i
       então A[i] ↔ A[maior]
               DESCE-HEAP (A, m, maior)
```

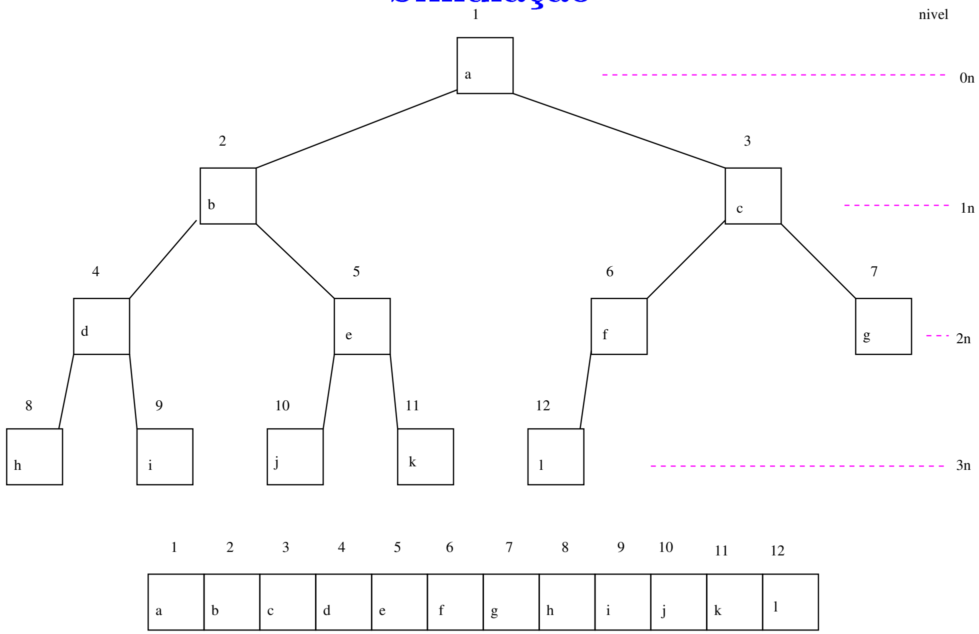
Simulação



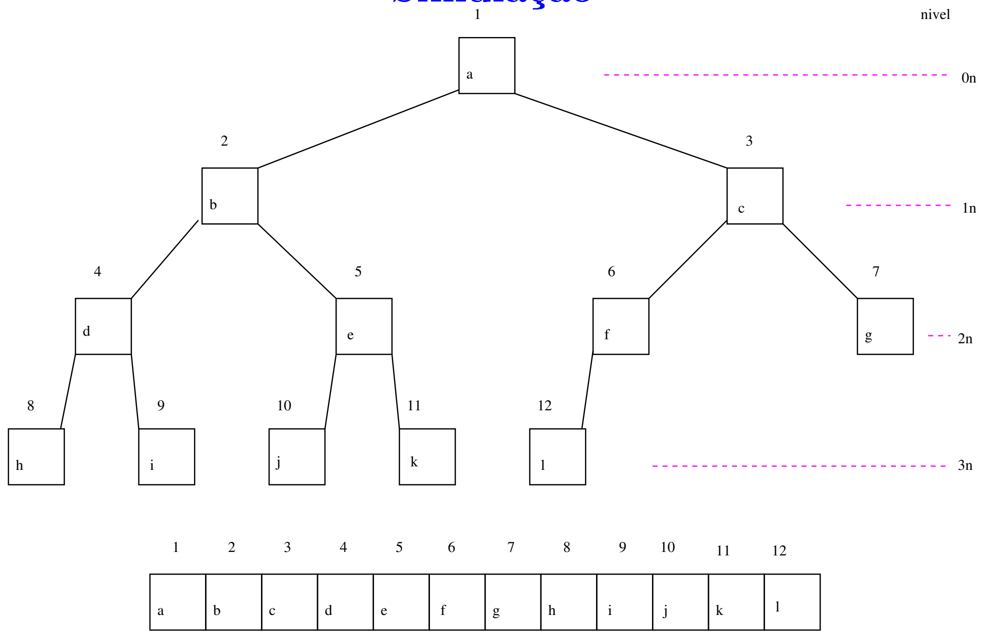
Simulação



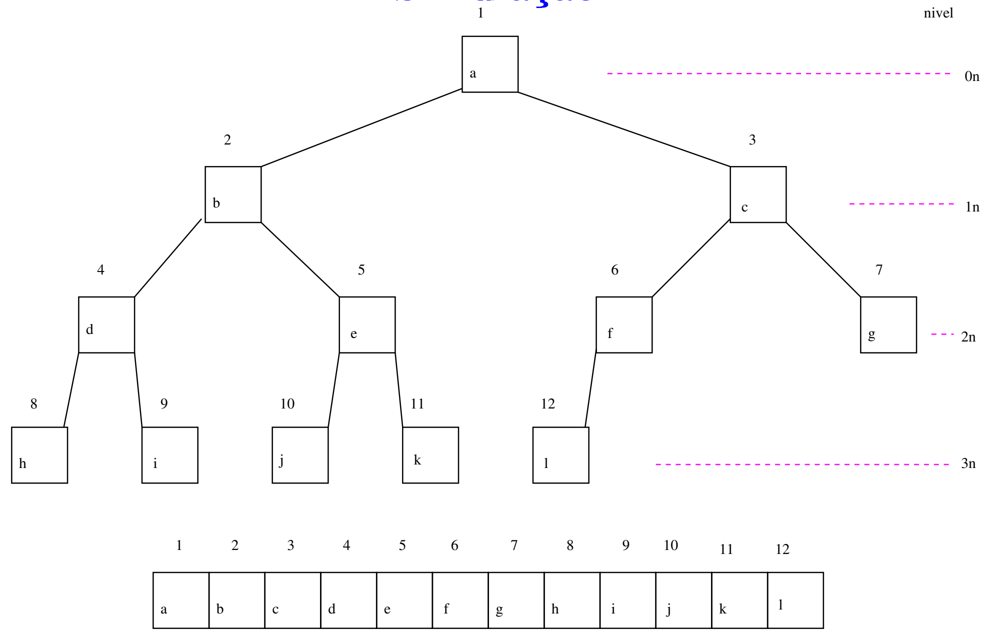
Simulação



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 $h - 1$.

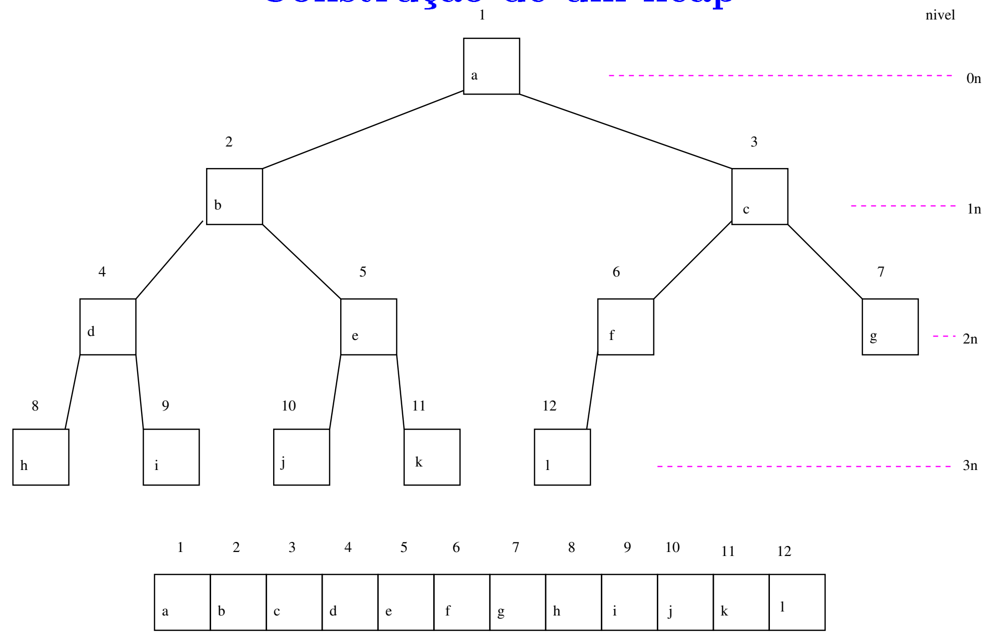
Solução assintótica: $T(h)$ é ???.

Solução assintótica: $T(h)$ é $O(h)$.

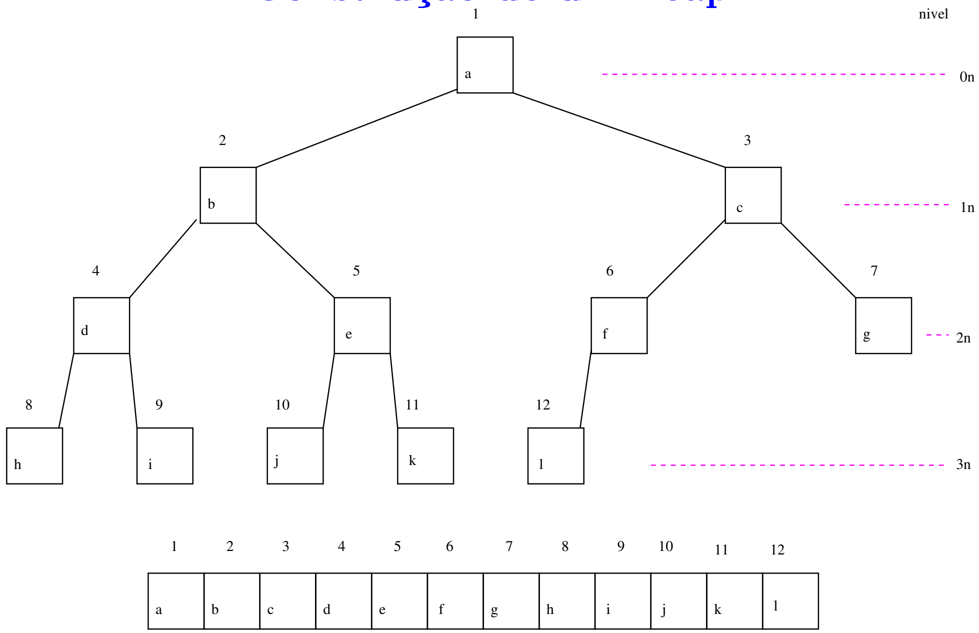
Como $h \leq \lg m$, podemos dizer que:

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

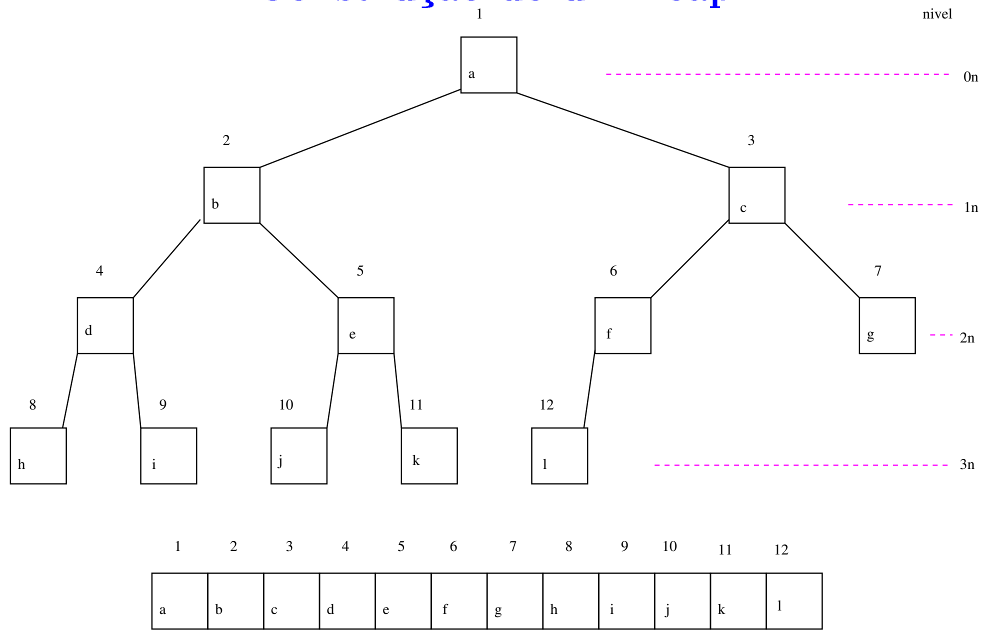
Construção de um heap



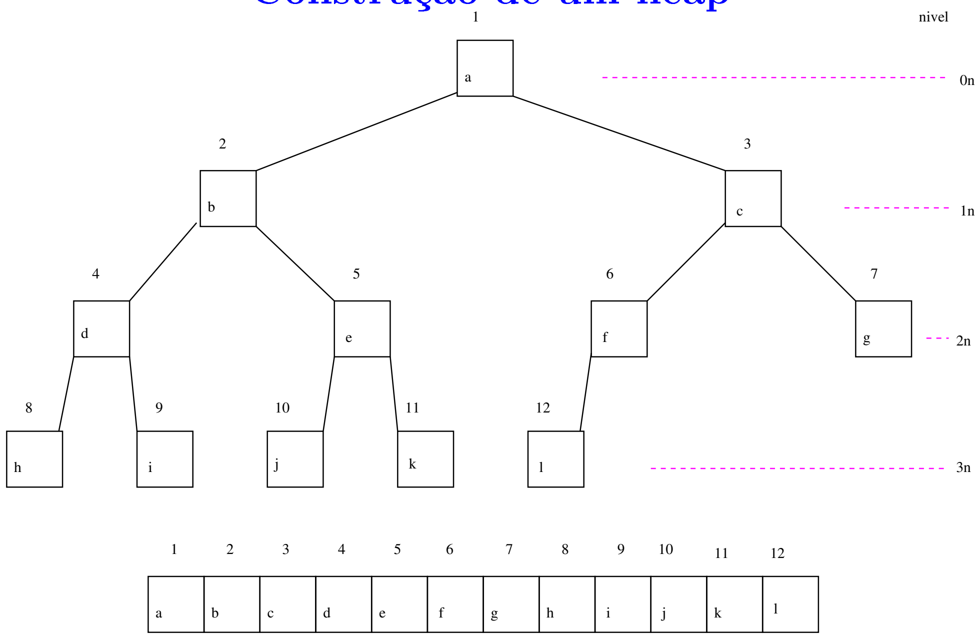
Construção de um heap



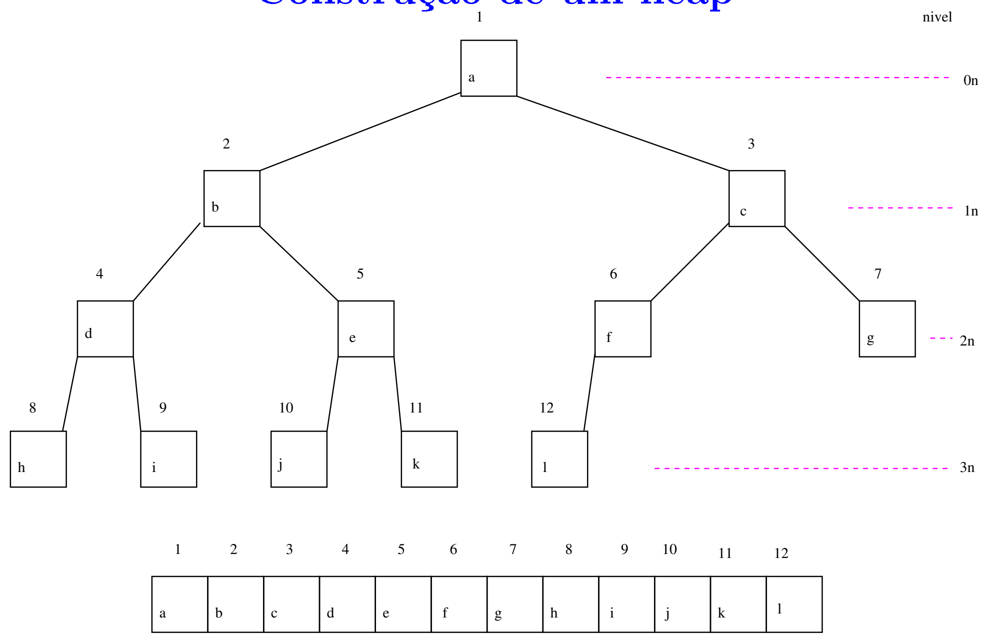
Construção de um heap



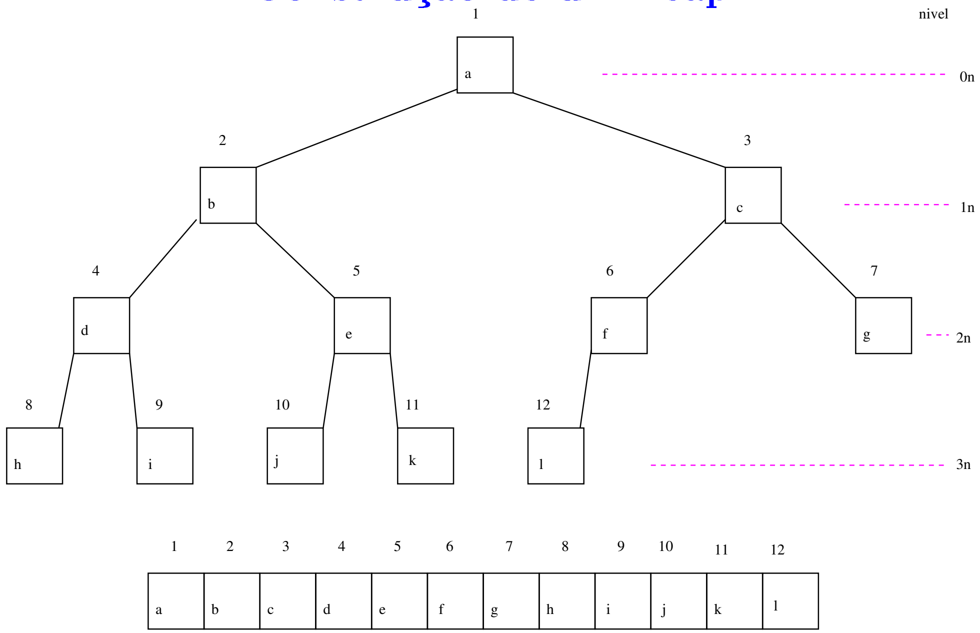
Construção de um heap



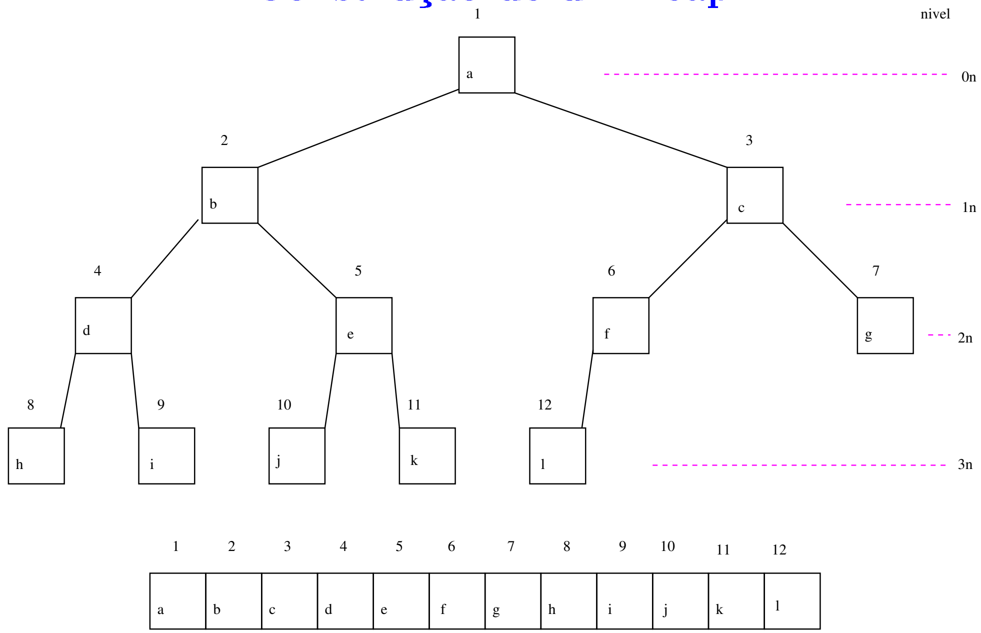
Construção de um heap



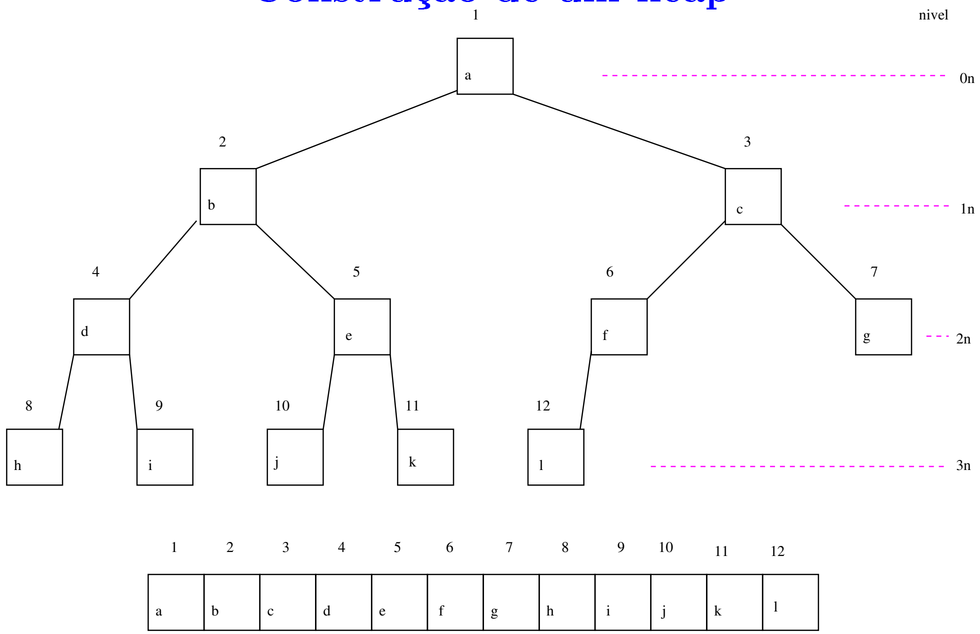
Construção de um heap



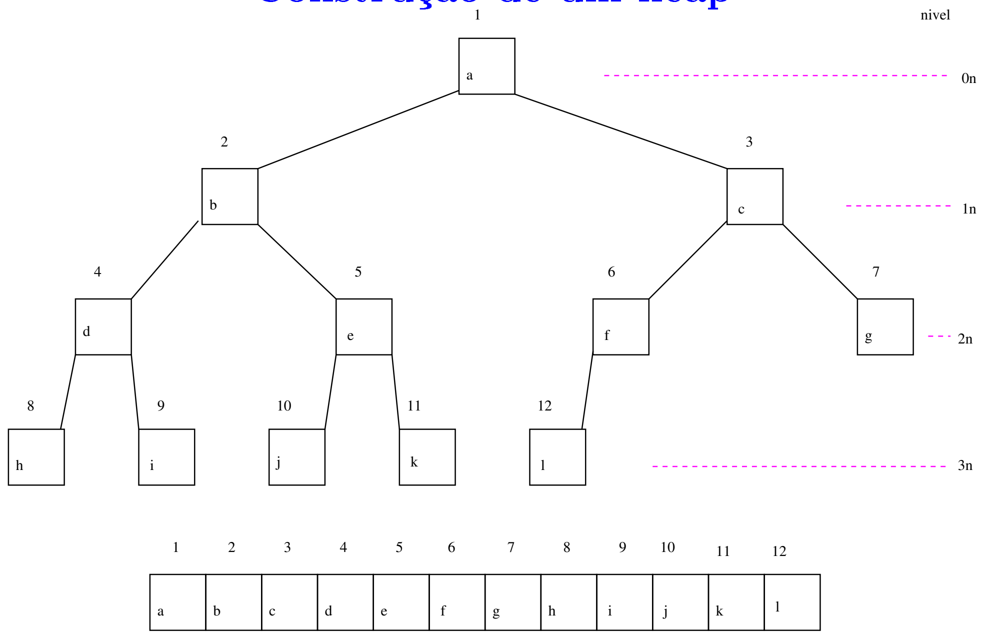
Construção de um heap



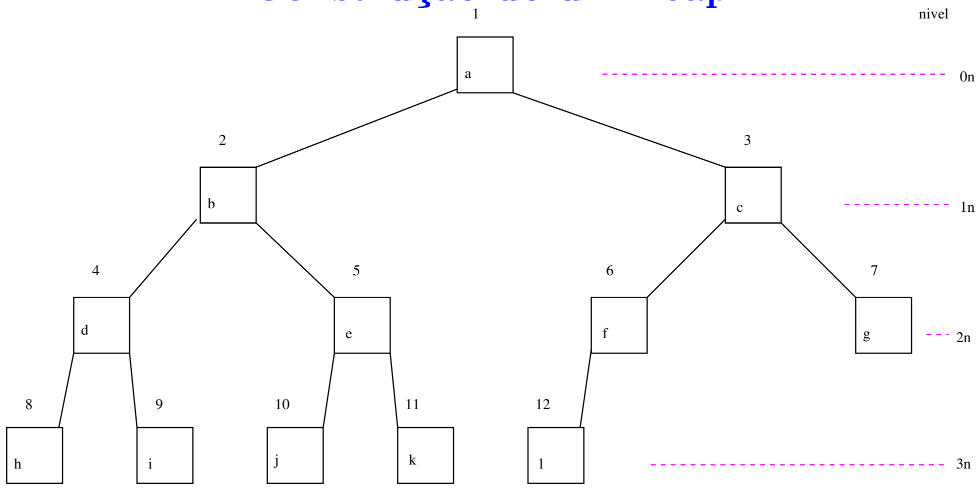
Construção de um heap



Construção de um heap

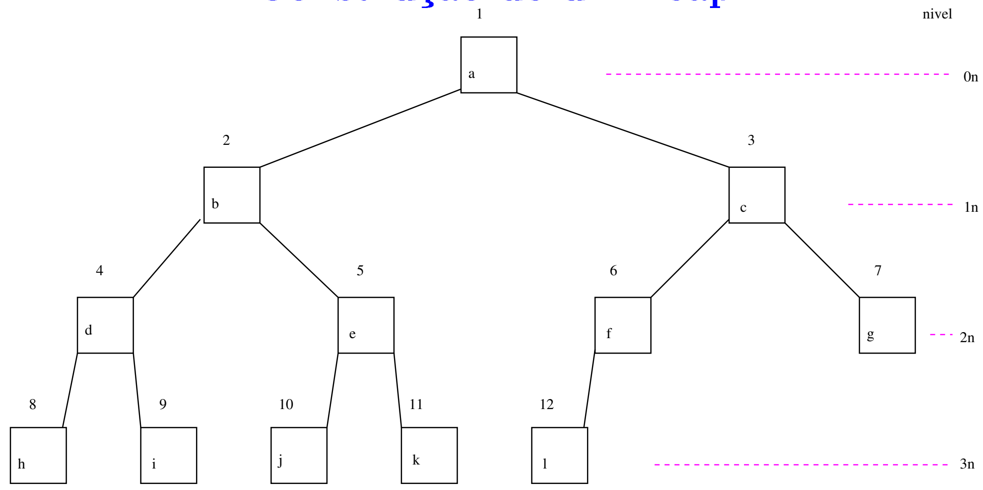


Construção de um heap



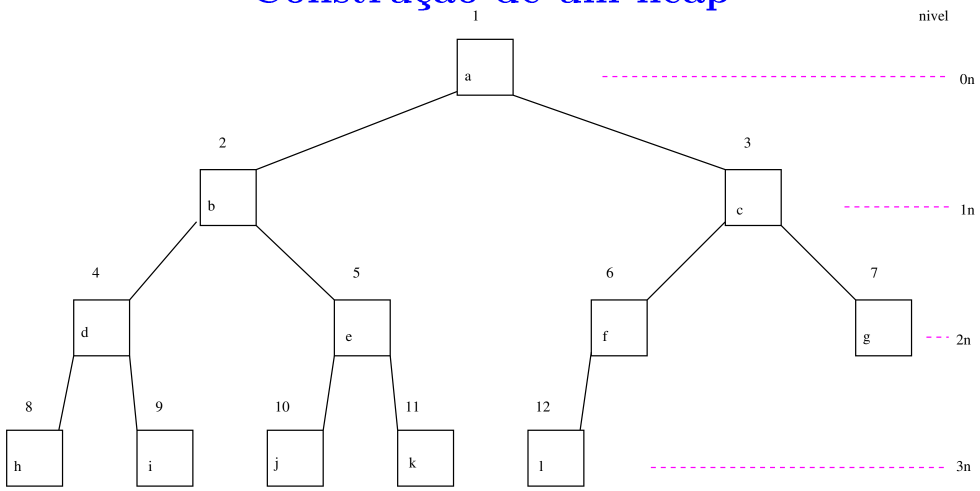
1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

Construção de um heap



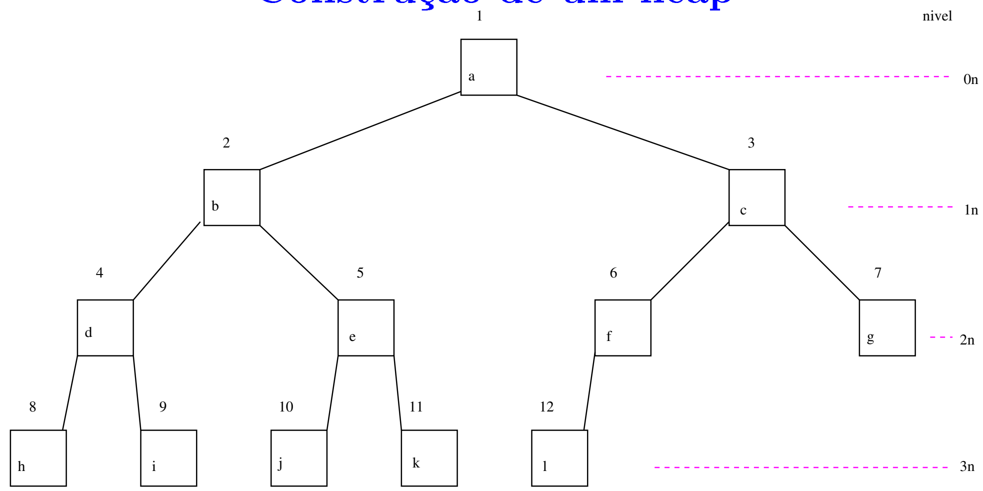
1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

Construção de um heap



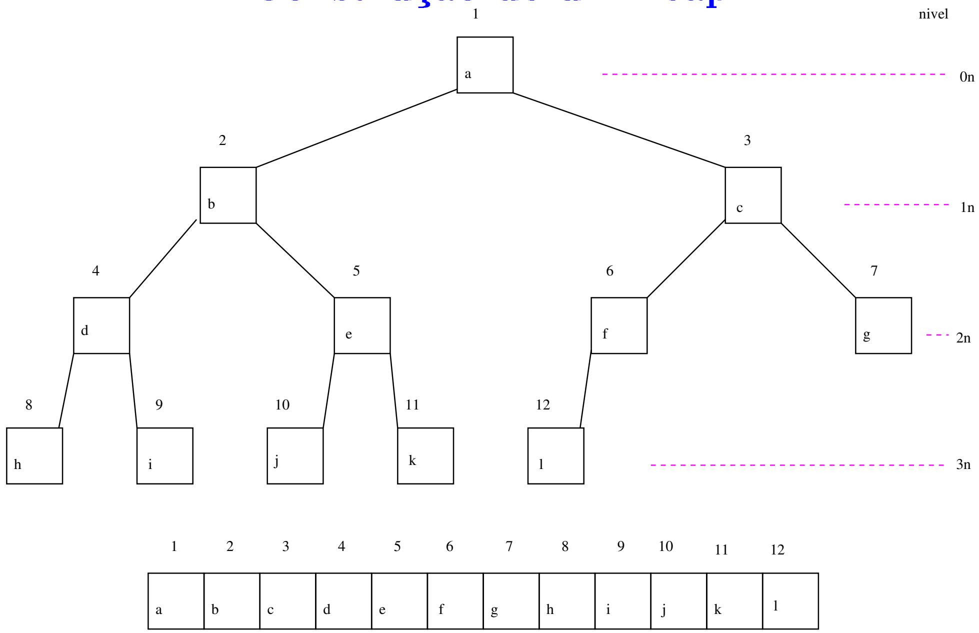
1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

Construção de um heap

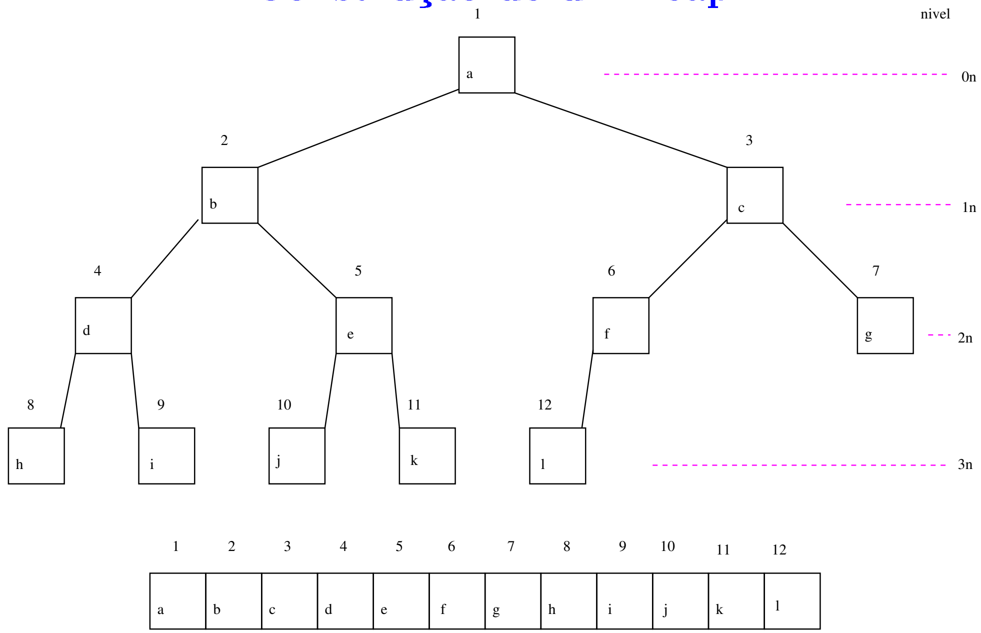


1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

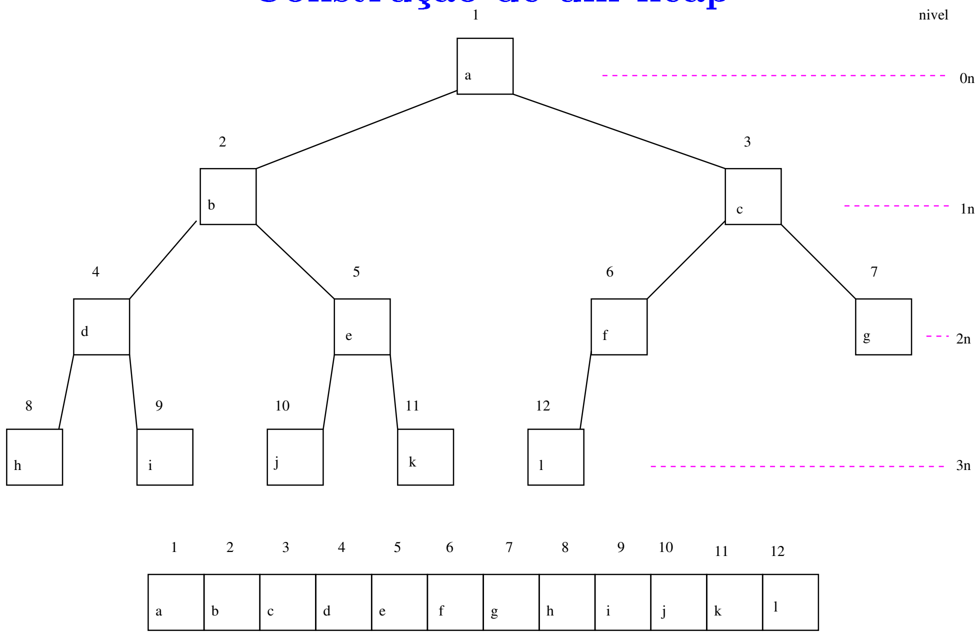
Construção de um heap



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, $i+1, \ldots, n$ são raízes de heaps.

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

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

Análise mais cuidadosa: $T(n)$ é ????.

$T(n)$ é $\mathrm{O}(n)$

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

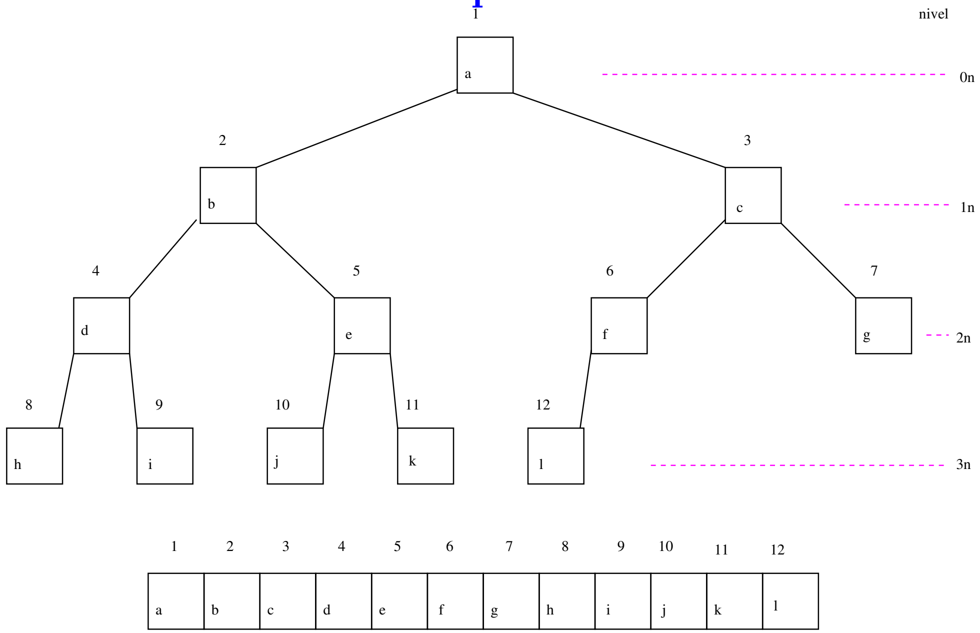
$$
\begin{aligned}
T(n) &= \sum_{h=1}^{\lfloor \lg n \rfloor} 2^{\lfloor \lg n \rfloor - h} h \\
&\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h} h \\
&\leq n \left(\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{\lfloor \lg n \rfloor}{2^{\lfloor \lg n \rfloor}} \right) \\
&< n \frac{1/2}{(1-1/2)^2}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

$T(n)$ é $\mathrm{O}(n)$

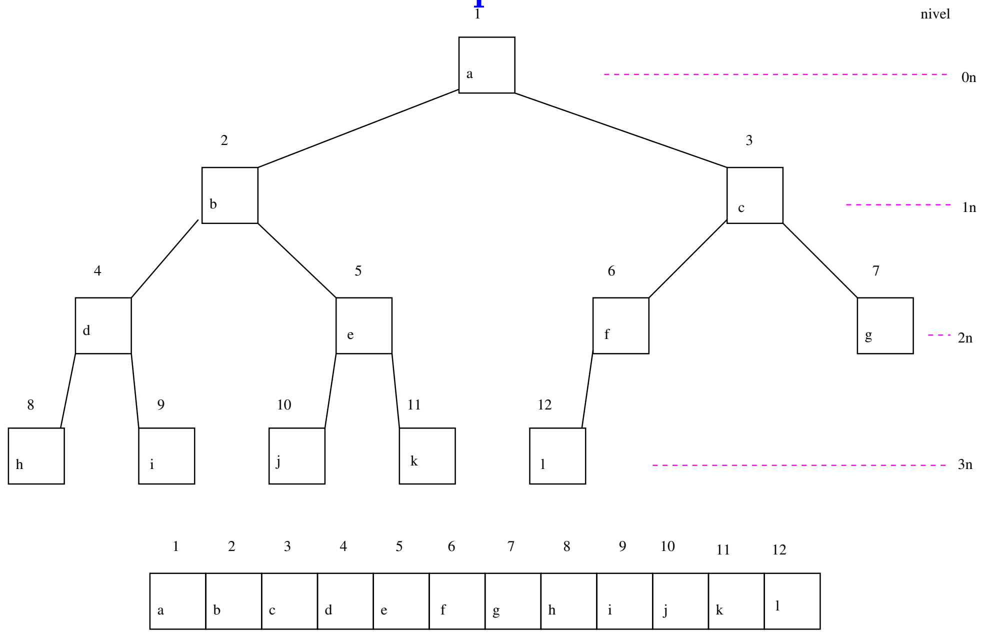
Prova:
O consumo de tempo de DESCE-HEAP (A, n, i) é $\mathrm{O}(h)$, onde h é a altura da árvore de raiz i. Logo,

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

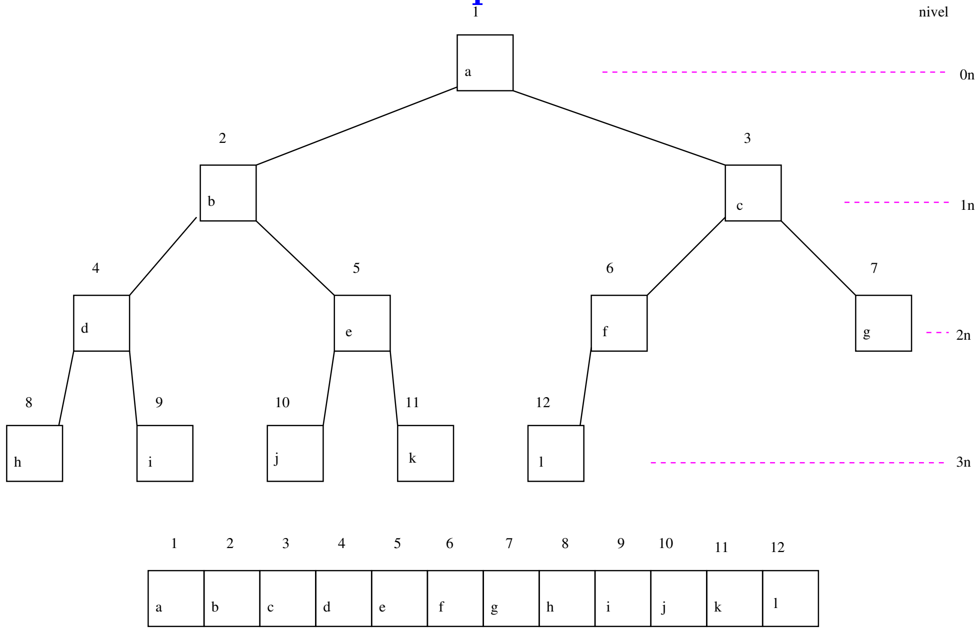
Heap sort



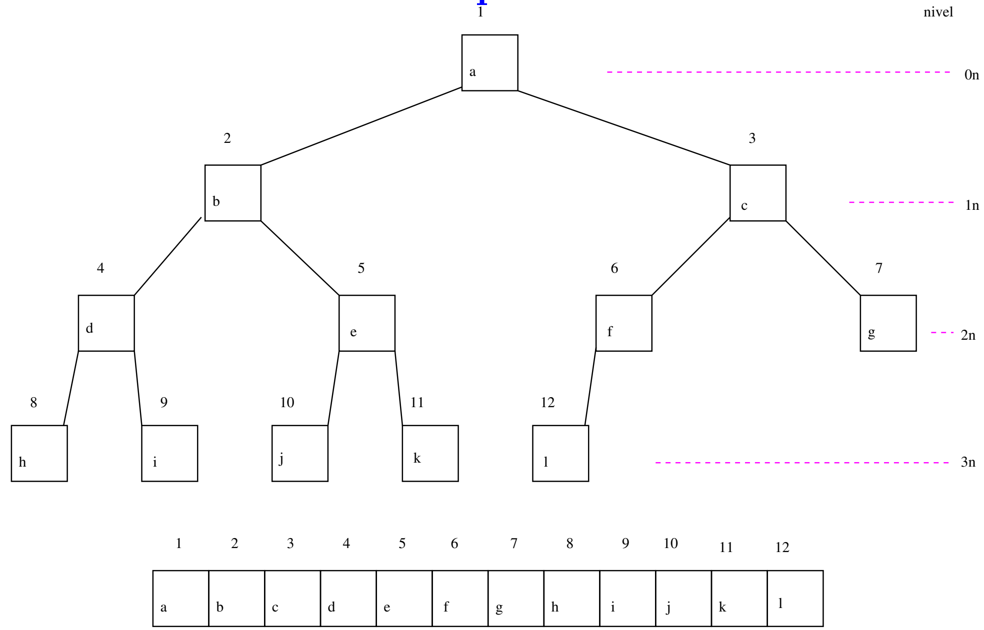
Heap sort



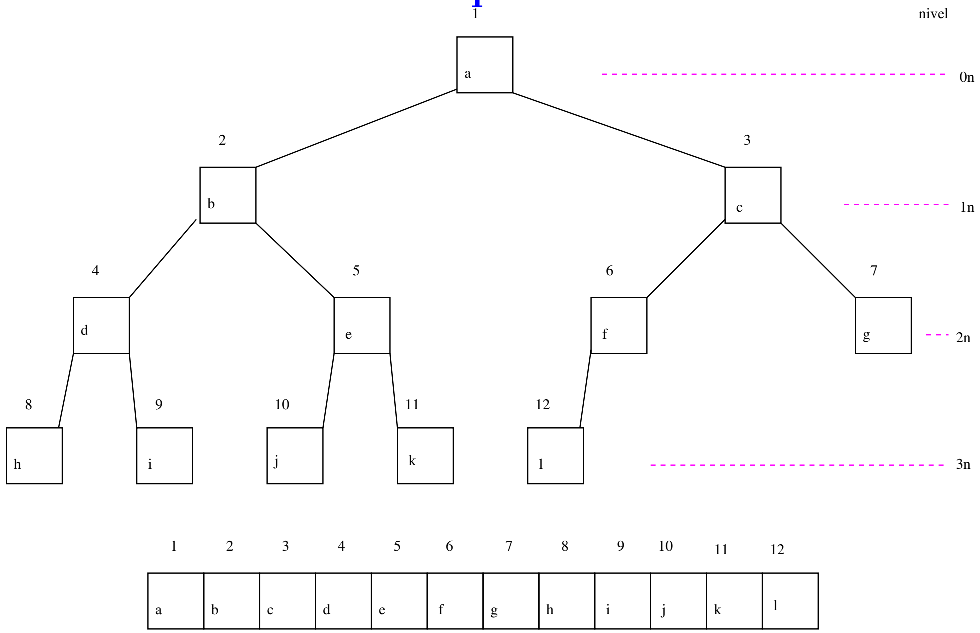
Heap sort



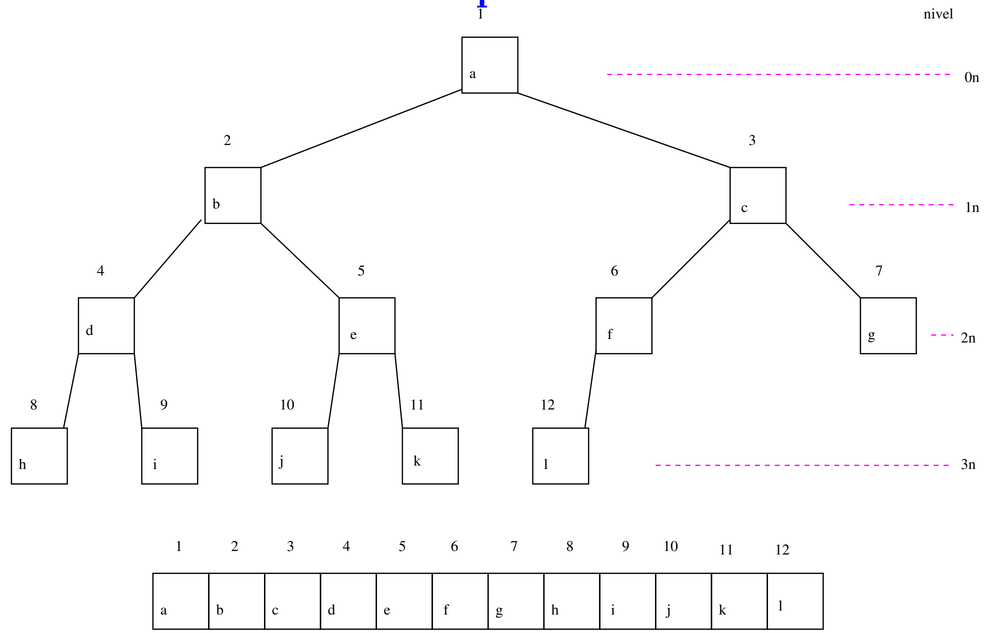
Heap sort



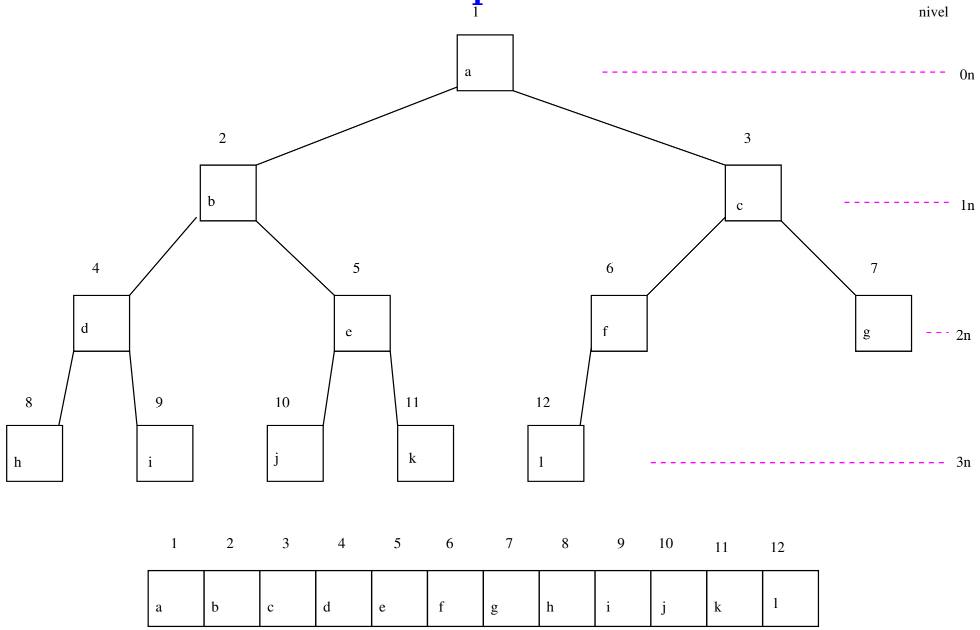
Heap sort



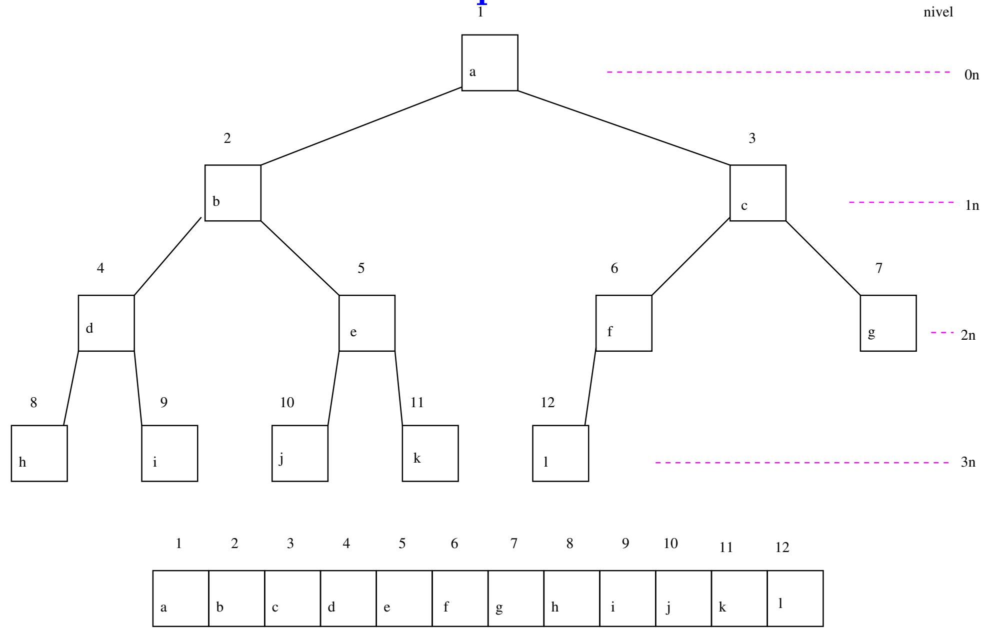
Heap sort



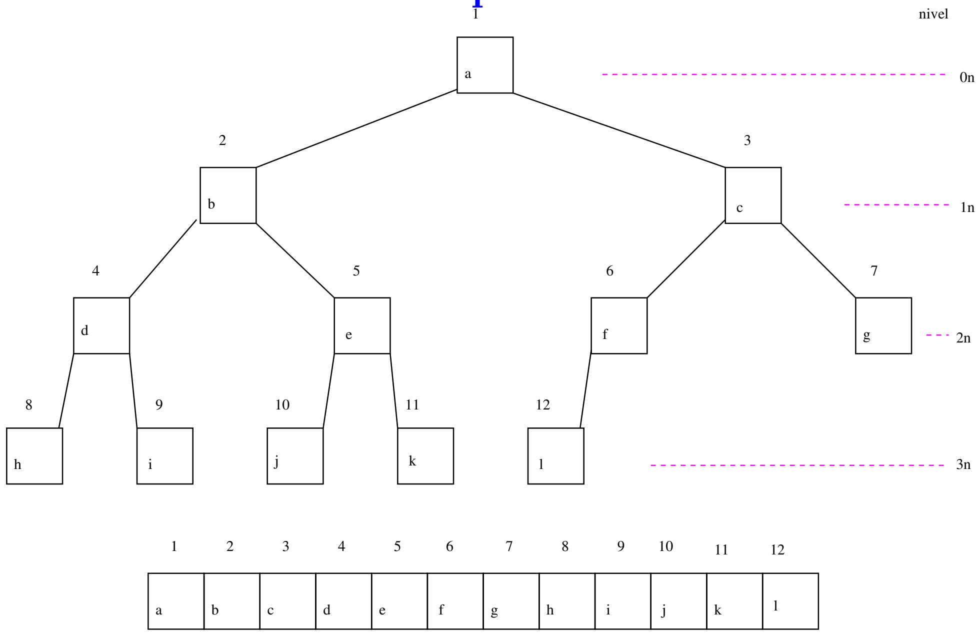
Heap sort



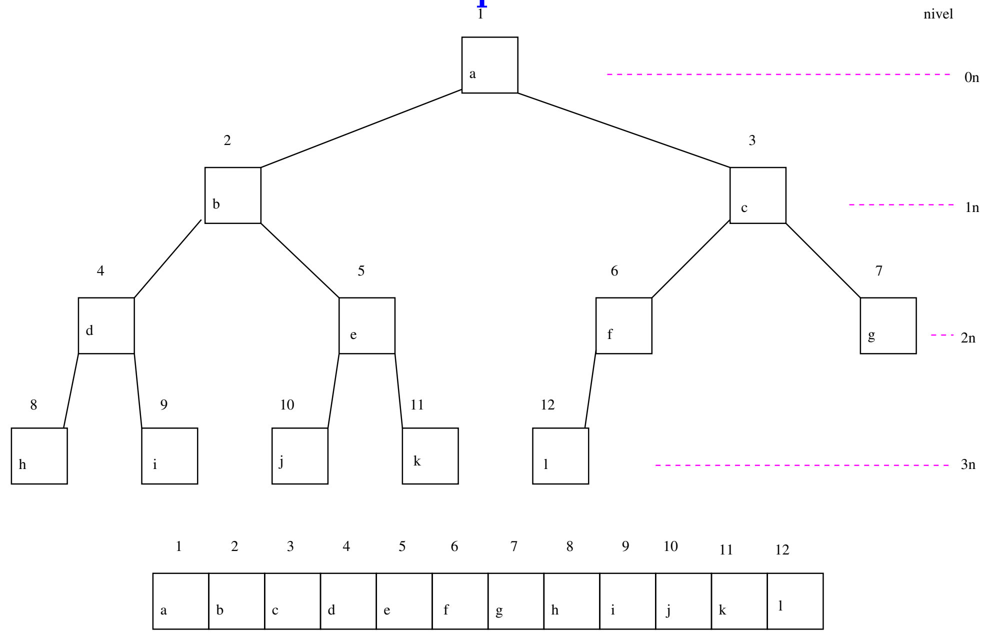
Heap sort



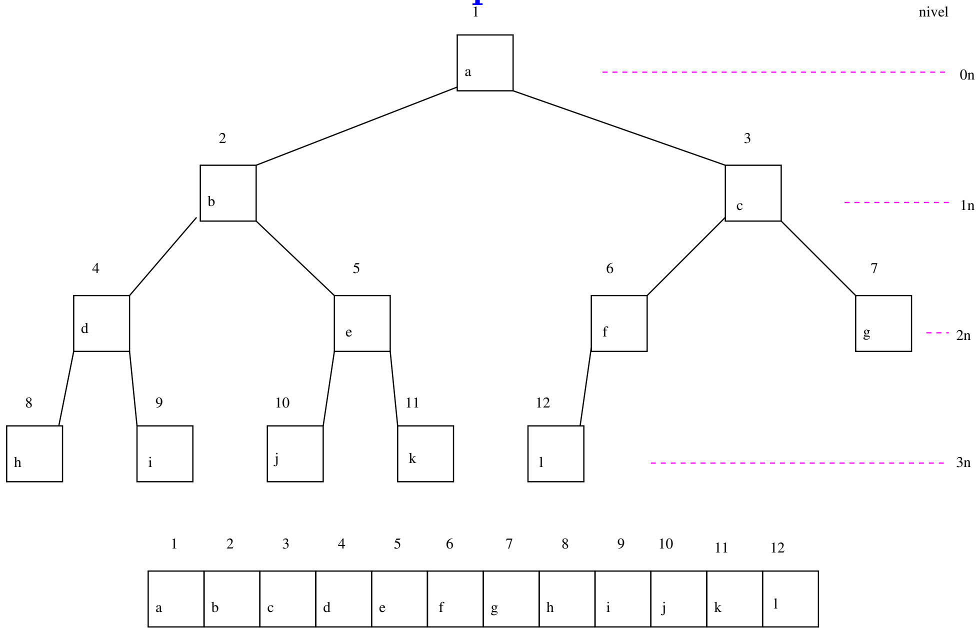
Heap sort



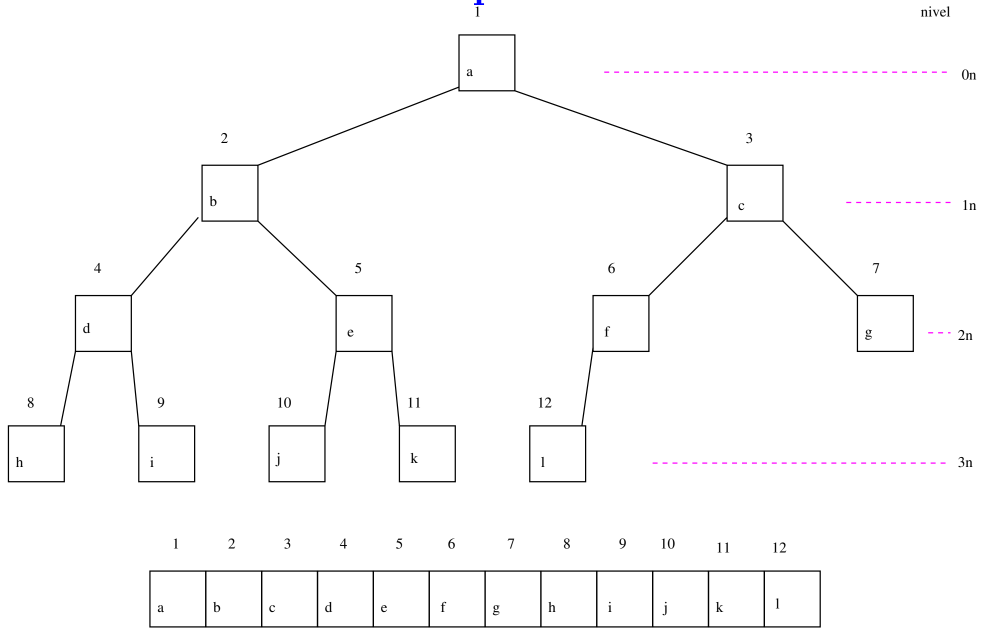
Heap sort



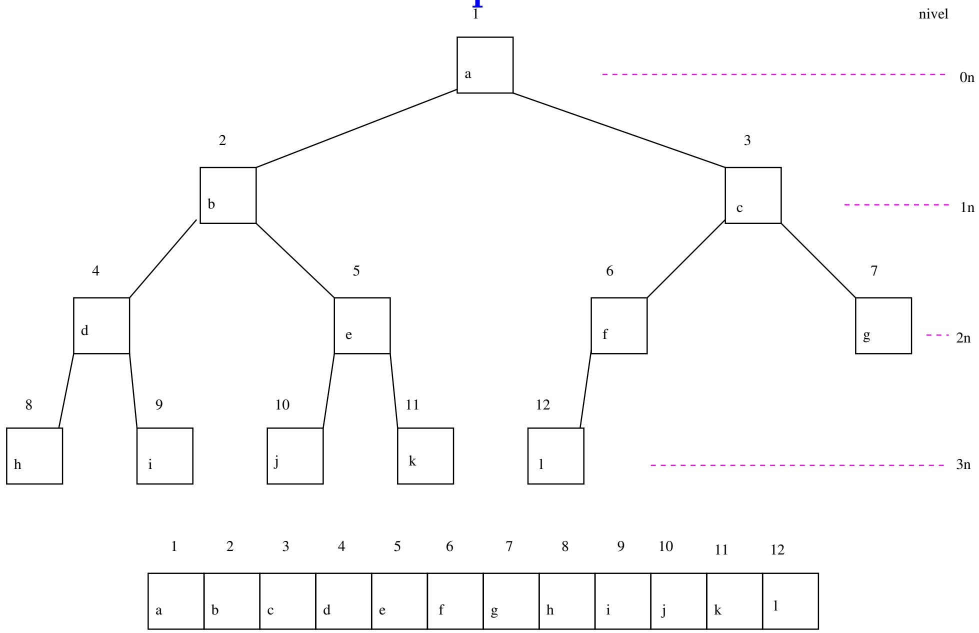
Heap sort



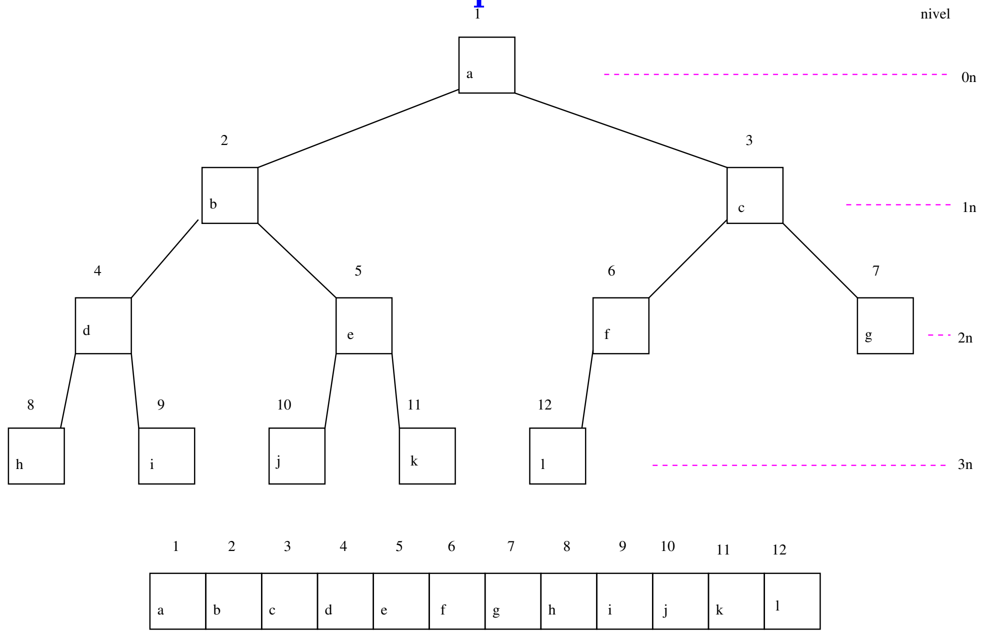
Heap sort



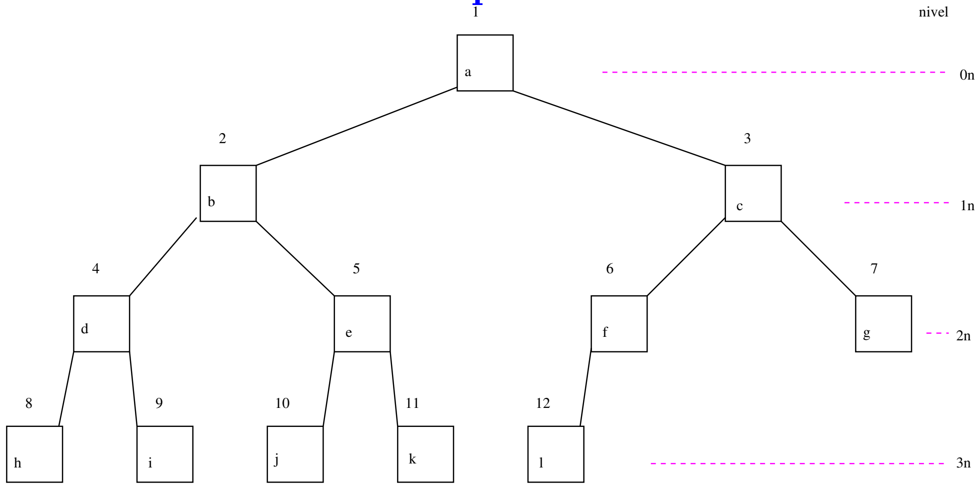
Heap sort



Heap sort

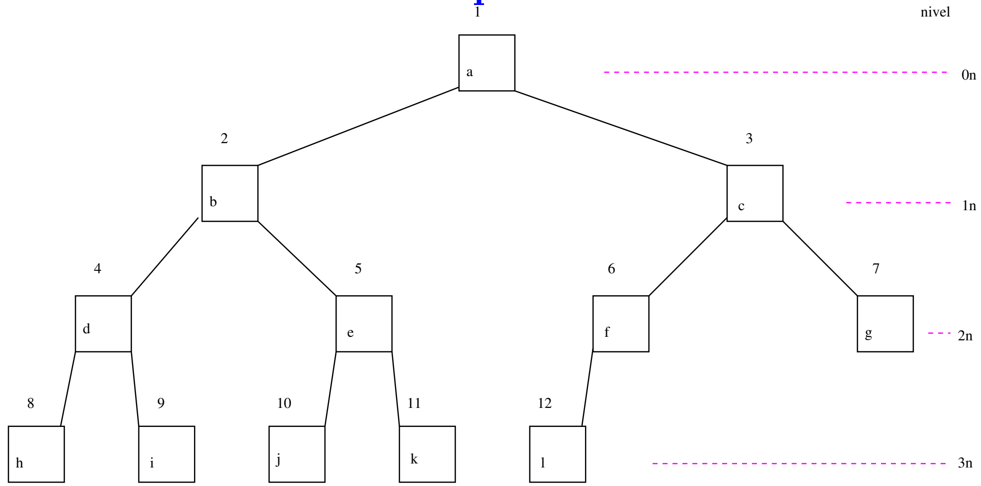


Heap sort



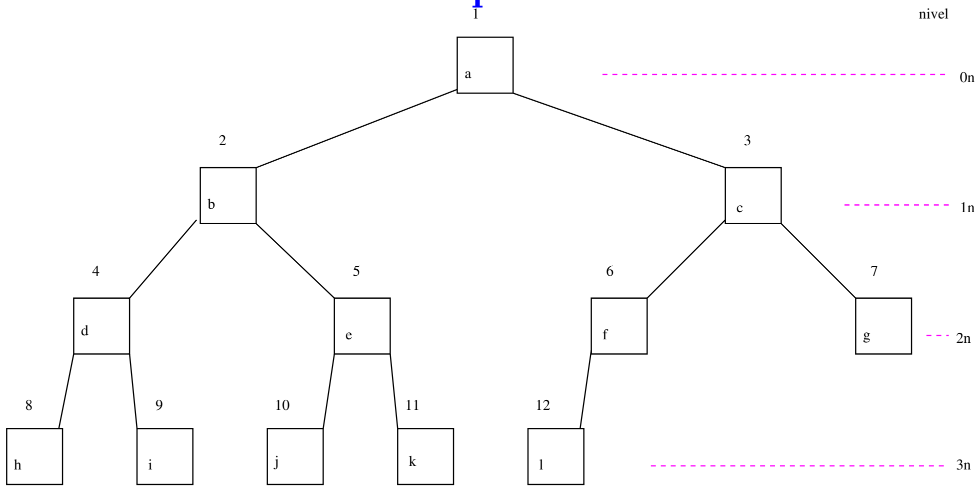
1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

Heap sort



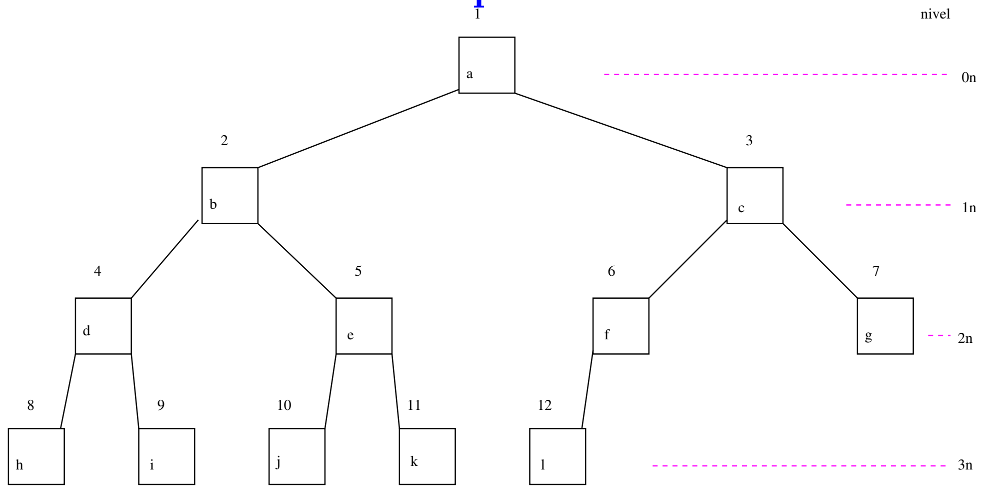
1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

Heap sort



1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

Heap sort



1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

Heap sort

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

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

Relações invariantes: Na linha 2 vale que:

(i0) $A[m\,..\,n]$ 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	$=$	$n\mathrm{O}(\lg n)$
total	$=$	$n\mathrm{O}(\lg n) + \Theta(n) = \mathrm{O}(n \lg n)$

O consumo de tempo do algoritmo HEAPSORT é $\mathrm{O}(n \lg n)$.

Próxima aula: Limites inferiores

CLRS 8.1