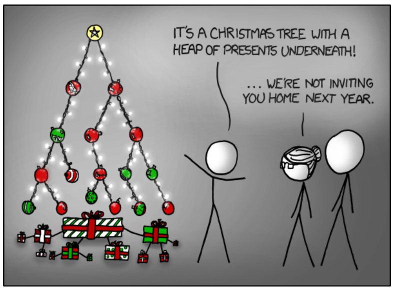
Árvores em listas e heaps

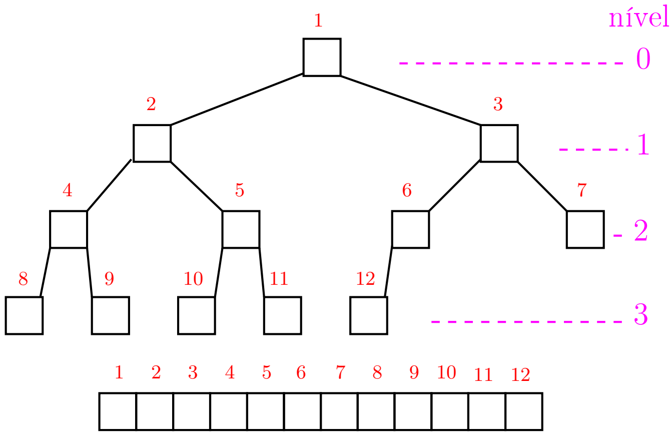


Fonte: `http://xkcd.com/835/`

PF 10

http://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html

Representação de árvores em listas



Pais e filhos

$\mathtt{v}[1:\mathtt{m}]$ é uma lista representando uma árvore.

Diremos que para qualquer **índice** ou **nó** $\mathtt{i}$,

- $\mathtt{i}//2$ é o **pai** de $\mathtt{i}$;
- $2\,\mathtt{i}$ é o **filho esquerdo** de $\mathtt{i}$;
- $2\,\mathtt{i}+1$ é o **filho direito**.

Um nó $\mathtt{i}$ só tem filho esquerdo se $2\,\mathtt{i} < \mathtt{m}$.

Um nó $\mathtt{i}$ só tem filho direito se $2\,\mathtt{i}+1 < \mathtt{m}$.

Raiz e folhas

O nó 1 não tem pai e é chamado de **raiz**.

Um nó $\mathtt{i}$ é um **folha** se não tem **filhos**, ou seja $2\,\mathtt{i} > \mathtt{m}$.

Todo nó $\mathtt{i}$ é raiz da subárvore formada por

$$\mathtt{v}[\mathtt{i}, 2\mathtt{i}, 2\mathtt{i}+1, 4\mathtt{i}, 4\mathtt{i}+1, 4\mathtt{i}+2, 4\mathtt{i}+3, 8\mathtt{i}, \ldots, 8\mathtt{i}+7, \ldots]$$

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ pertence ao nível ???.

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ pertence ao nível $\lfloor \lg \mathtt{i} \rfloor$.

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ pertence ao nível $\lfloor \lg \mathtt{i} \rfloor$.

Prova: Se $\mathtt{p}$ é o nível do nó $\mathtt{i}$, então

$$\begin{array}{rcccll} 2^{\mathtt{p}} & \leq & \mathtt{i} & < & 2^{\mathtt{p}+1} & \Rightarrow \\ \lg 2^{\mathtt{p}} & \leq & \lg \mathtt{i} & < & \lg 2^{\mathtt{p}+1} & \Rightarrow \\ \mathtt{p} & \leq & \lg \mathtt{i} & < & \mathtt{p}+1 & \end{array}$$

Logo, $\mathtt{p} = \lfloor \lg \mathtt{i} \rfloor$.

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ pertence ao nível $\lfloor \lg \mathtt{i} \rfloor$.

Prova: Se $\mathtt{p}$ é o nível do nó $\mathtt{i}$, então

$$\begin{array}{rcccll} 2^{\mathtt{p}} & \leq & \mathtt{i} & < & 2^{\mathtt{p}+1} & \Rightarrow \\ \lg 2^{\mathtt{p}} & \leq & \lg \mathtt{i} & < & \lg 2^{\mathtt{p}+1} & \Rightarrow \\ \mathtt{p} & \leq & \lg \mathtt{i} & < & \mathtt{p}+1 & \end{array}$$

Logo, $\mathtt{p} = \lfloor \lg \mathtt{i} \rfloor$.

Portanto, o número total de níveis é ???.

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ pertence ao nível $\lfloor \lg \mathtt{i} \rfloor$.

Prova: Se $\mathtt{p}$ é o nível do nó $\mathtt{i}$, então

$$\begin{array}{rcccll} 2^{\mathtt{p}} & \leq & \mathtt{i} & < & 2^{\mathtt{p}+1} & \Rightarrow \\ \lg 2^{\mathtt{p}} & \leq & \lg \mathtt{i} & < & \lg 2^{\mathtt{p}+1} & \Rightarrow \\ \mathtt{p} & \leq & \lg \mathtt{i} & < & \mathtt{p}+1 & \end{array}$$

Logo, $\mathtt{p} = \lfloor \lg \mathtt{i} \rfloor$.

Portanto, o número total de níveis é $1 + \lfloor \lg \mathtt{m} \rfloor$.

Altura

A **altura** de um nó $\mathtt{i}$ é o maior comprimento de um caminho de $\mathtt{i}$ a uma folha.

Em outras palavras, a altura de um nó $\mathtt{i}$ é o maior comprimento de uma seqüência da forma

$$\langle \mathtt{filho}(\mathtt{i}), \mathtt{filho}(\mathtt{filho}(\mathtt{i})), \mathtt{filho}(\mathtt{filho}(\mathtt{filho}(\mathtt{i}))), .$$

onde $\mathtt{filho}(\mathtt{i})$ vale $2\mathtt{i}$ ou $2\mathtt{i}+1$.

Os nós que têm **altura zero** são as folhas.

Altura

A **altura** de um nó $\mathtt{i}$ é o maior comprimento de um caminho de $\mathtt{i}$ a uma folha.

Em outras palavras, a altura de um nó $\mathtt{i}$ é o maior comprimento de uma seqüência da forma

$$\langle \mathtt{filho}(\mathtt{i}), \mathtt{filho}(\mathtt{filho}(\mathtt{i})), \mathtt{filho}(\mathtt{filho}(\mathtt{filho}(\mathtt{i}))), .$$

onde $\mathtt{filho}(\mathtt{i})$ vale $2\mathtt{i}$ ou $2\mathtt{i}+1$.

Os nós que têm **altura zero** são as folhas.

A altura de um nó $\mathtt{i}$ é $\lfloor \lg(\mathtt{m}/\mathtt{i}) \rfloor$ $(\ldots)$.

Resumão

filho esquerdo de i: $2\,i$

filho direito de i: $2\,i+1$

pai de i: $i//2$

nível da raiz: 0

nível de i: $\lfloor \lg i \rfloor$

altura da raiz: $\lfloor \lg m \rfloor$

altura da árvore: $\lfloor \lg m \rfloor$

altura de i: $\lfloor \lg(m/i) \rfloor$ (...)

altura de uma folha: 0

total de nós de altura h $\leq \lceil m/2^{h+1} \rceil$ (...)

Heaps

Uma lista $v[1:m]$ é um max-heap se

$$v[i//2] \geq v[i]$$

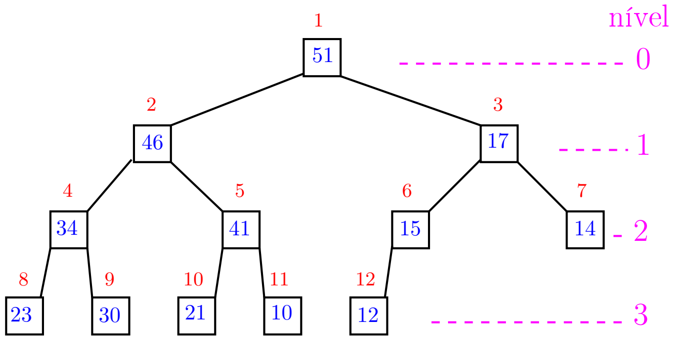
para todo $i = 2, 3, \ldots, m{-}1$.

De uma forma mais geral, $v[j:m]$ é um max-heap se

$$v[i//2] \geq v[i]$$

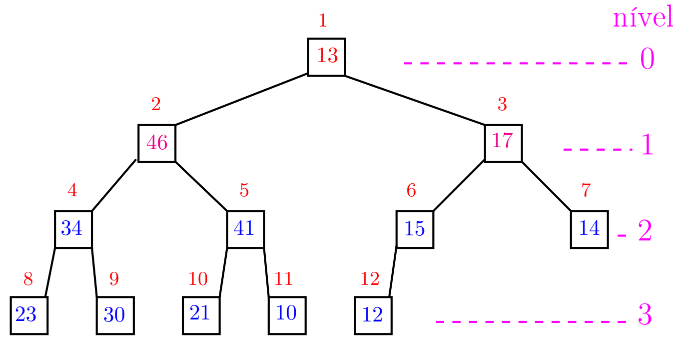
para todo
$i = 2j, 2j+1, 4j, \ldots, 4j+3, 8j, \ldots, 8j+7, \ldots$.
Neste caso também diremos que a subárvore com raiz j é um max-heap.

max-heap



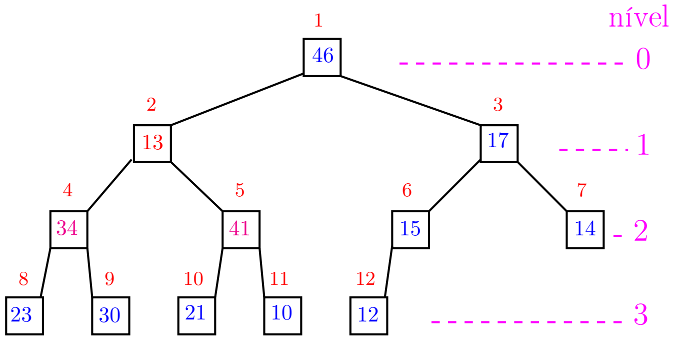
1	2	3	4	5	6	7	8	9	10	11	12
51	46	17	34	41	15	14	23	30	21	10	12

Função básica de manipulação de max-heap



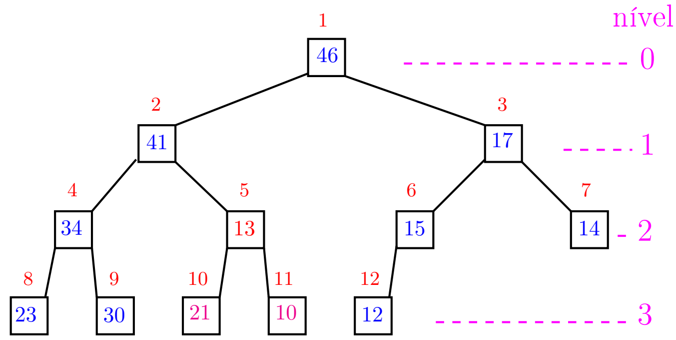
1	2	3	4	5	6	7	8	9	10	11	12
13	46	17	34	41	15	14	23	30	21	10	12

Função básica de manipulação de max-heap



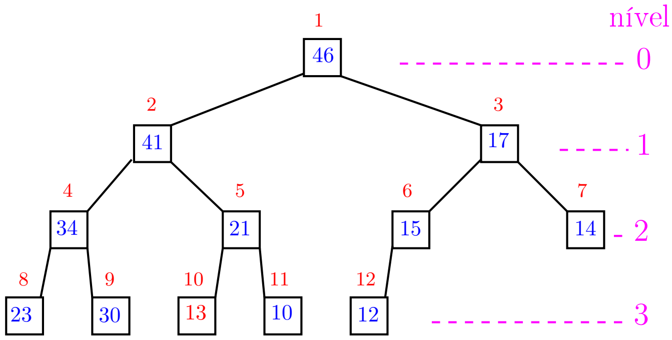
1	2	3	4	5	6	7	8	9	10	11	12
46	13	17	34	41	15	14	23	30	21	10	12

Função básica de manipulação de max-heap



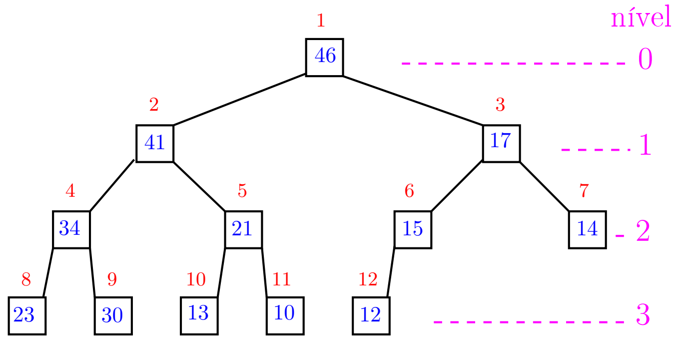
1	2	3	4	5	6	7	8	9	10	11	12
46	41	17	34	13	15	14	23	30	21	10	12

Função básica de manipulação de max-heap



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

Função básica de manipulação de max-heap



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

Função peneire

O coração de qualquer algoritmo que manipule um max-heap é uma função que recebe uma lista arbitrário v[1:m] e um índice i e faz v[i] "descer" para sua posição correta.

Função peneire

Rearranja a lista v[1:m] de modo que a "sublista" cuja raiz é i seja um max-heap.

```
def peneire(i, m, v):
    f = 2*i
    if f < m-1 and v[f] < v[f+1]:f+=1
    while f < m and v[i] < v[f]:
        v[i], v[f] = v[f], v[i]
        i = f
        f = 2*i
        if f < m-1 and v[f] < v[f+1]:f+=1
```

Função peneire

Supõe que as "sublistas" cujas raízes são filhos de i já são max-heap.

```
def peneire(i, m, v):
    f = 2*i
    if f < m-1 and v[f] < v[f+1]:f+=1
    while f < m and v[i] < v[f]:
        v[i], v[f] = v[f], v[i]
        i = f
        f = 2*i
        if f < m-1 and v[f] < v[f+1]:f+=1
```

Função peneire

A seguinte implementação é um pouco melhor pois em vez de trocas faz apenas deslocamentos (linha 5).

```
def peneire(i, m, v):
1   x = v[i]
2   f = 2*i
3   if f < m-1 and v[f] < v[f+1]:f+=1
4   while f < m and x < v[f]:
5       v[i] = v[f]
6       i = f
7       f = 2*i
8       if f < m-1 and v[f] < v[f+1]:f+=1
9   v[i] = x
```

Consumo de tempo

linha		todas as execuções da linha
1-3	$=$	1
4	$\le$	$1 + \lg \mathtt{m}$
5	$\le$	$\lg \mathtt{m}$
6	$\le$	$\lg \mathtt{m}$
7	$\le$	$\lg \mathtt{m}$
8	$\le$	$\lg \mathtt{m}$
9	$=$	1
total	$\le$	$3 + 5\lg \mathtt{m} = \mathrm{O}(\lg \mathtt{m})$

Conclusão

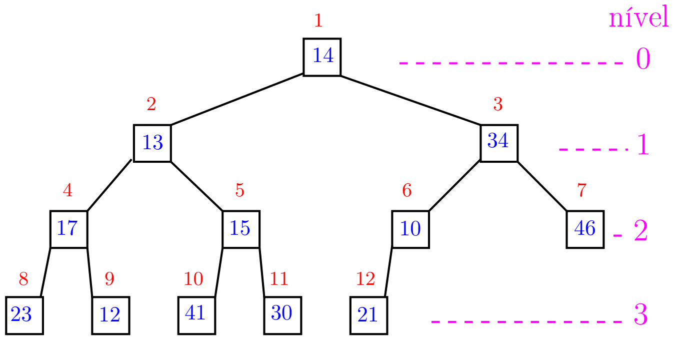
O consumo de tempo da função `peneire` é proporcional a $\lg \mathtt{m}$.

O consumo de tempo da função `peneire` é $\mathrm{O}(\lg \mathtt{m})$.

Verdade seja dita ... (...)

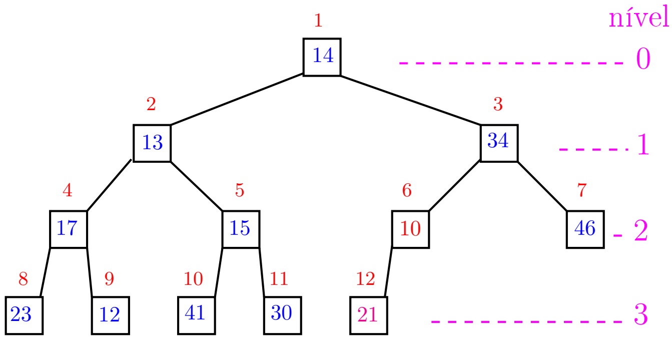
O consumo de tempo da função `peneire` é proporcional a $\mathrm{O}(\lg \mathtt{m}/\mathtt{i})$.

Construção de um `max-heap`



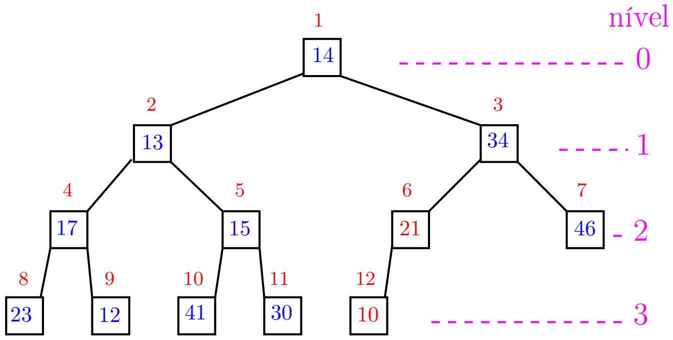
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	10	46	23	12	41	30	21

Construção de um `max-heap`



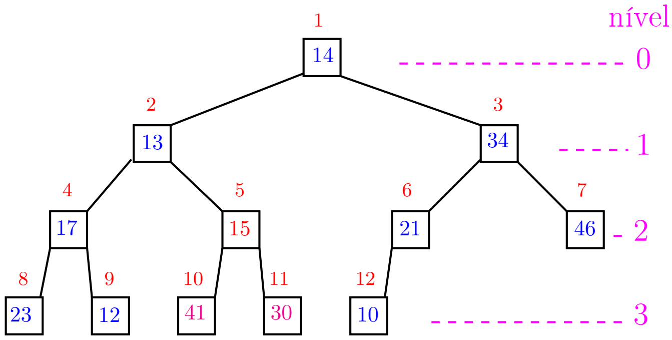
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	10	46	23	12	41	30	21

Construção de um `max-heap`



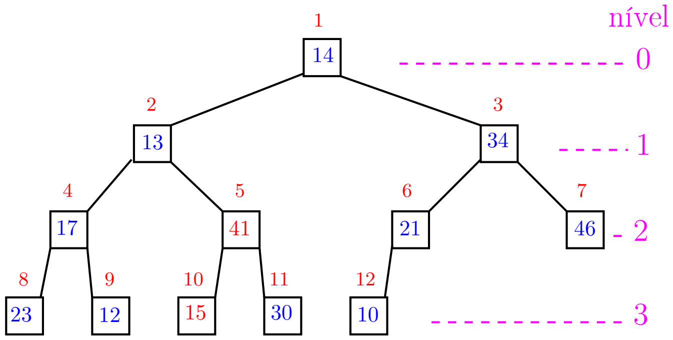
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	21	46	23	12	41	30	10

Construção de um `max-heap`



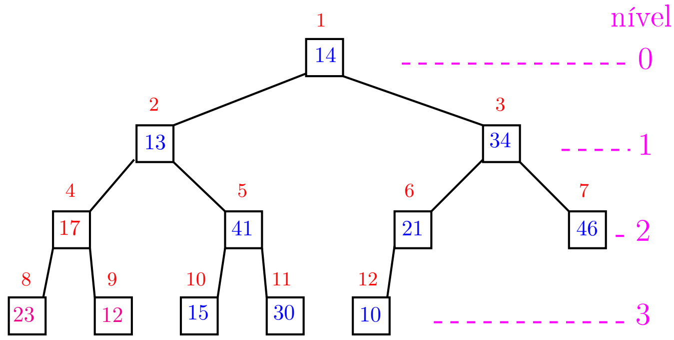
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	21	46	23	12	41	30	10

Construção de um max-heap



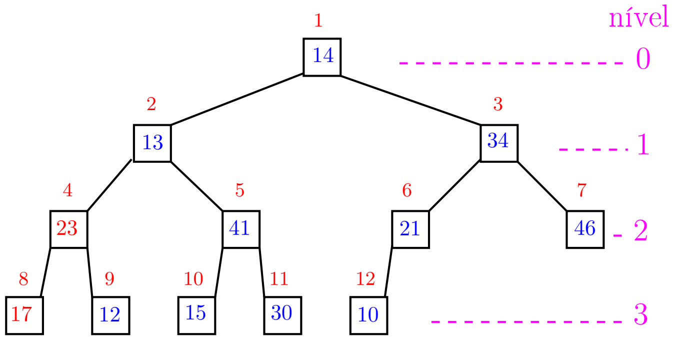
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	41	21	46	23	12	15	30	10

Construção de um max-heap



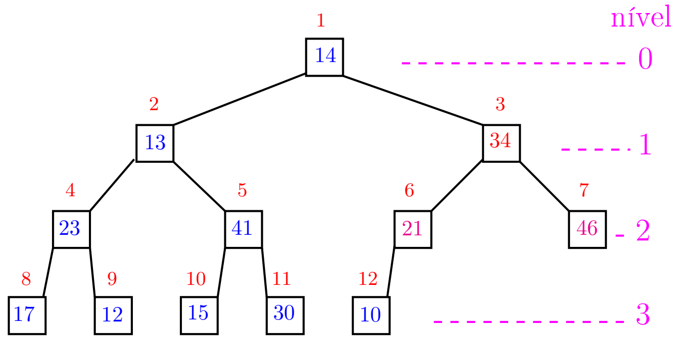
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	41	21	46	23	12	15	30	10

Construção de um max-heap



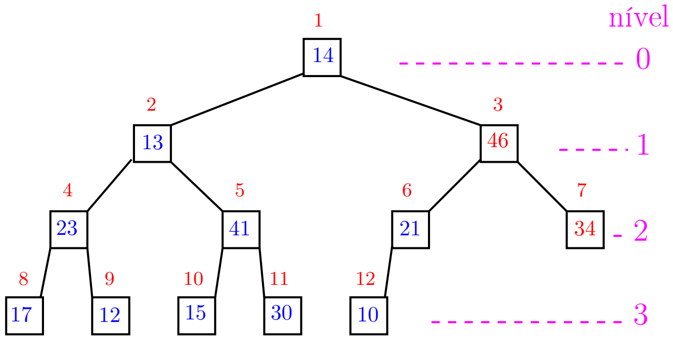
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	23	41	21	46	17	12	15	30	10

Construção de um max-heap



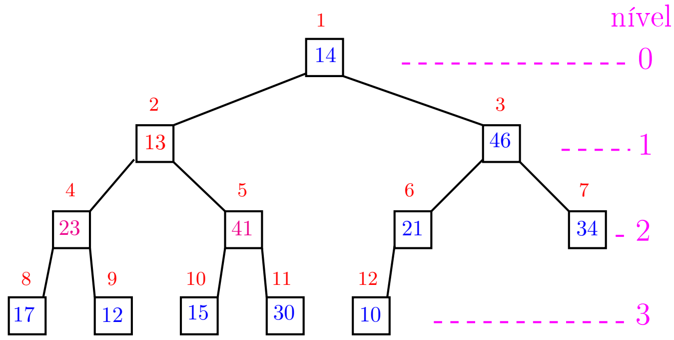
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	23	41	21	46	17	12	15	30	10

Construção de um max-heap



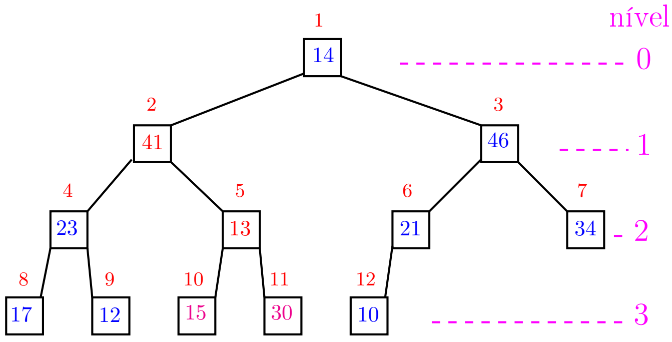
1	2	3	4	5	6	7	8	9	10	11	12
14	13	46	23	41	21	34	17	12	15	30	10

Construção de um max-heap



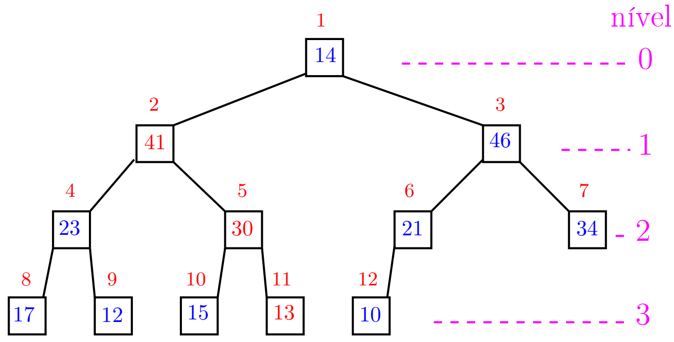
1	2	3	4	5	6	7	8	9	10	11	12
14	13	46	23	41	21	34	17	12	15	30	10

Construção de um max-heap



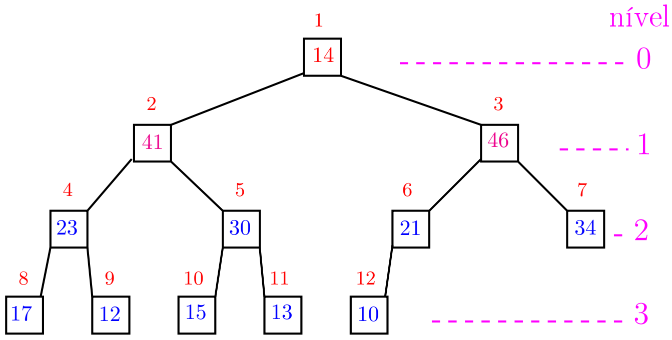
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	13	21	34	17	12	15	30	10

Construção de um max-heap



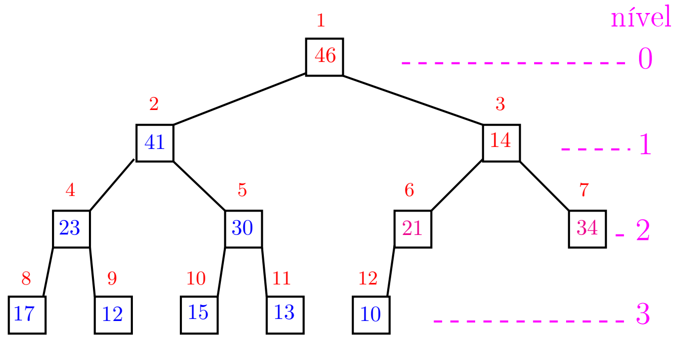
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	30	21	34	17	12	15	13	10

Construção de um max-heap



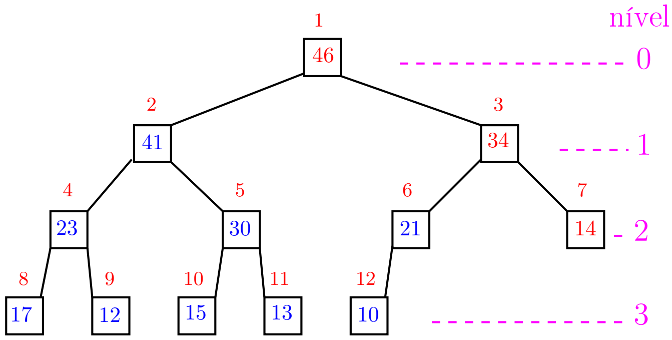
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	30	21	34	17	12	15	13	10

Construção de um max-heap



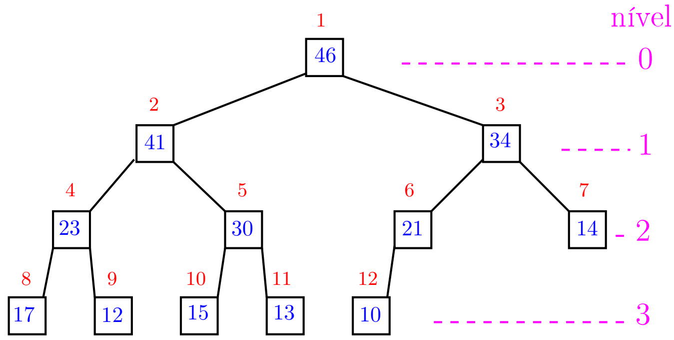
1	2	3	4	5	6	7	8	9	10	11	12
46	41	14	23	30	21	34	17	12	15	13	10

Construção de um max-heap



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

Construção de um max-heap



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

Construção de um max-heap

Recebe uma lista $v[1:n]$ e rearranja v para que seja max-heap.

```
for i in range((n-1)//2, 0, -1):  #A#
    peneire(i, n, v)
```

Relação invariante:

(i0) em #A# vale que, $i+1, \ldots, n-1$ são raízes de max-heaps.

Consumo de tempo

Análise grosseira: consumo de tempo é

$$\frac{n}{2} \times \lg n = O(n \lg n).$$

Verdade seja dita ... (...)

Análise mais cuidadosa: consumo de tempo é $O(n)$.

Conclusão

O consumo de tempo para construir um max-heap é $O(n \lg n)$.

Verdade seja dita ... (...)

O consumo de tempo para construir um max-heap é $O(n)$.