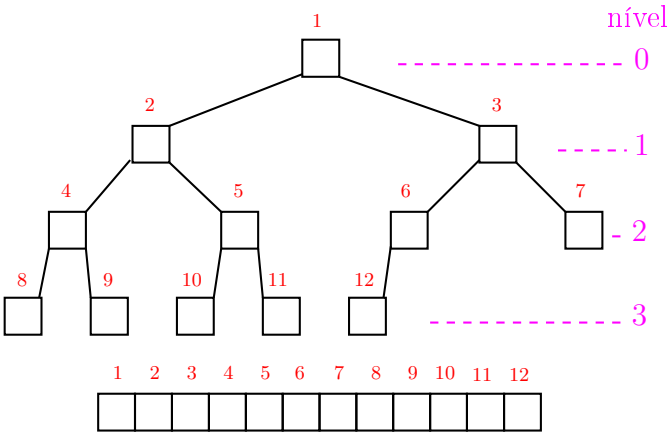
Representação de árvores em vetores e heaps

AULA 25

PF 10
http://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html

Representação de árvores em vetores



Pais e filhos

$v[1\,..\,m]$ é um vetor representando uma árvore.

Diremos que para qualquer **índice** ou **nó** i,

- $\lfloor i/2 \rfloor$ é o **pai** de i;
- $2\,i$ é o **filho esquerdo** de i;
- $2\,i+1$ é o **filho direito**.

Um nó i só tem filho esquerdo se $2\,i \leq m$.

Um nó i só tem filho direito se $2\,i+1 \leq m$.

Raiz e folhas

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

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

Todo nó i é raiz da subárvore formada por

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

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

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

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

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

O nó i pertence ao nível ???.

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

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

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

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

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

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

Prova: Se p é o nível do nó i, então

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

Logo, $p = \lfloor \lg i \rfloor$.

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

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

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

Prova: Se p é o nível do nó i, então

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

Logo, $p = \lfloor \lg i \rfloor$.

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

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

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

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

Prova: Se p é o nível do nó i, então

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

Logo, $p = \lfloor \lg i \rfloor$.

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

Altura

A **altura** de um nó i é o maior comprimento de um caminho de i a uma folha.

Em outras palavras, a altura de um nó i é o maior comprimento de uma seqüência da forma

$\langle \texttt{filho}(i), \texttt{filho}(\texttt{filho}(i)), \texttt{filho}(\texttt{filho}(\texttt{filho}(i))), \ldots$

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

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

Altura

A **altura** de um nó i é o maior comprimento de um caminho de i a uma folha.

Em outras palavras, a altura de um nó i é o maior comprimento de uma seqüência da forma

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

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

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

A altura de um nó i é $\lfloor \lg(m/i) \rfloor$ (MAC0338).

Resumão

filho esquerdo de i:	$2\,i$
filho direito de i:	$2\,i+1$
pai de i:	$\lfloor i/2 \rfloor$
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:	$\lfloor \lg(m/i) \rfloor$ (MAC0338)
altura de uma folha:	0
total de nós de altura h	$\leq \lceil m/2^{h+1} \rceil$ (MAC0338)

Heaps

Um vetor $v[1\,..\,m]$ é um max-heap se

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

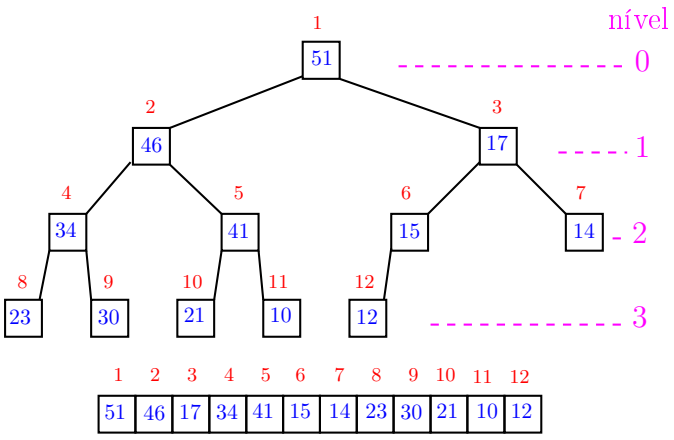
para todo $i = 2, 3, \ldots, m$.

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

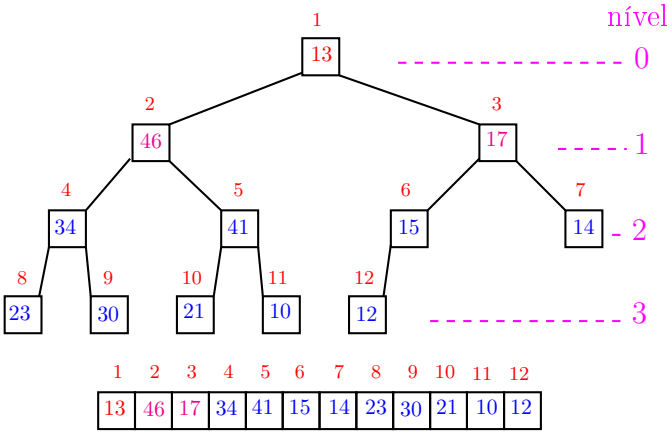
$$v[i/2] \geq v[i]$$

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.

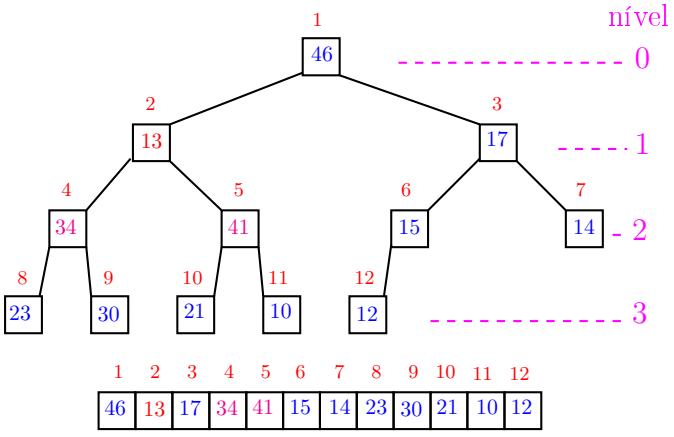
max-heap



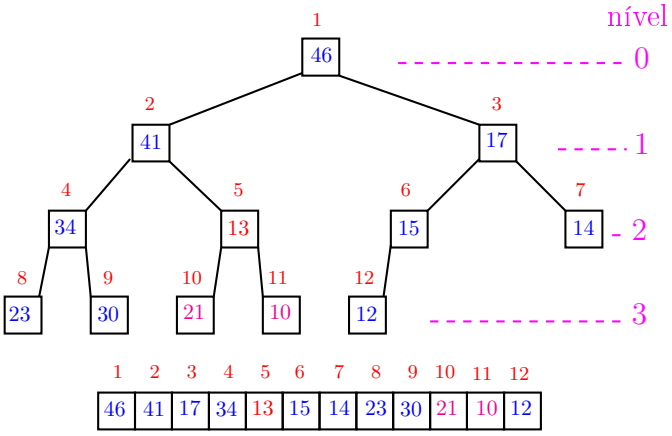
Função básica de manipulação de max-heap



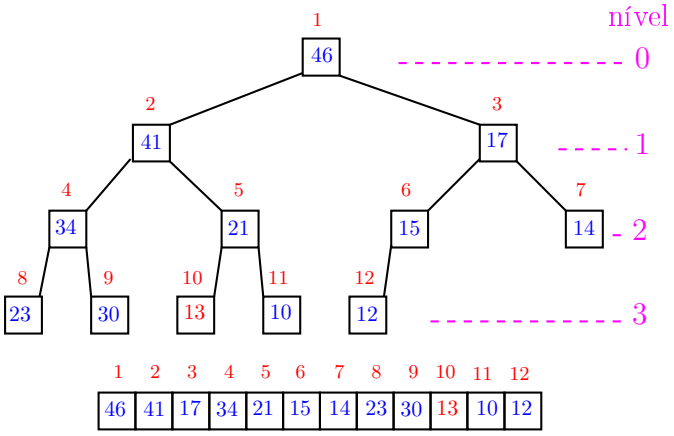
Função básica de manipulação de max-heap



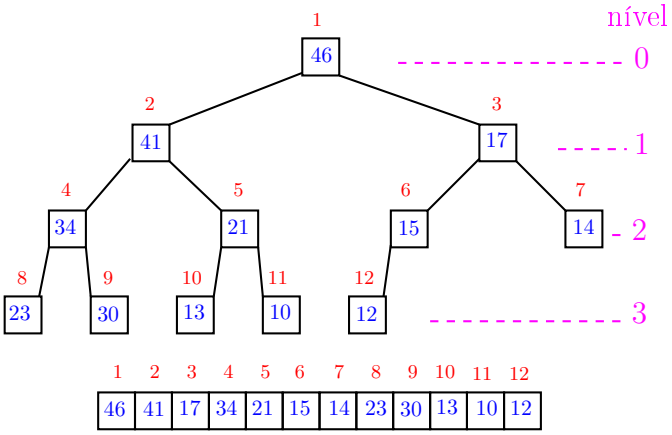
Função básica de manipulação de max-heap



Função básica de manipulação de max-heap



Função básica de manipulação de max-heap



Função peneira

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

Função peneira

Rearranja o vetor v[1 . . m] de modo que o "subvetor" cuja raiz é i seja um max-heap.

```
void peneira (int i, int m, int v[]) {
1   int f = 2*i, x;
2   while (f <= m) {
3       if (f < m && v[f] < v[f+1]) f++;
4       if (v[i] >= v[f]) break;
5       x = v[i]; v[i] = v[f]; v[f] = x;
6       i = f; f = 2*i;
    }
 }
```

Função peneira

Supõe que os "subvetores"cujas raízes são filhos de i já são max-heap.

```
void peneira (int i, int m, int v[]) {
1   int f = 2*i, x;
2   while (f <= m) {
3       if (f < m && v[f] < v[f+1]) f++;
4       if (v[i] >= v[f]) break;
5       x = v[i]; v[i] = v[f]; v[f] = x;
6       i = f; f = 2*i;
    }
 }
```

Função peneira

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

```
void peneira (int i, int m, int v[]) {
1   int f = 2*i, x = v[i];
2   while (f <= m) {
3       if (f < m && v[f] < v[f+1]) f++;
4       if (x >= v[f]) break;
5       v[i] = v[f];
6       i = f; f = 2*i;
    }
7   v[i] = x;
 }
```

Consumo de tempo

linha		todas as execuções da linha
1	$=$	1
2	$\leq$	$1 + \lg m$
3	$\leq$	$\lg m$
4	$\leq$	$\lg m$
5	$\leq$	$\lg m$
6	$\leq$	$\lg m$
7	$=$	1
total	$\leq$	$3 + 5 \lg m = O(\lg m)$

Conclusão

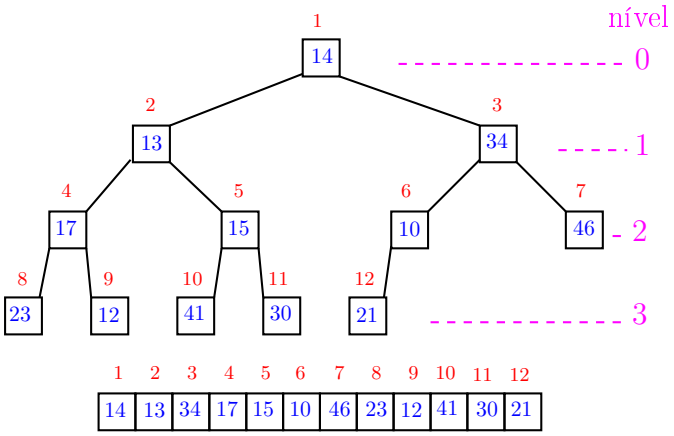
O consumo de tempo da função peneira é proporcional a $\lg m$.

O consumo de tempo da função peneira é $O(\lg m)$.

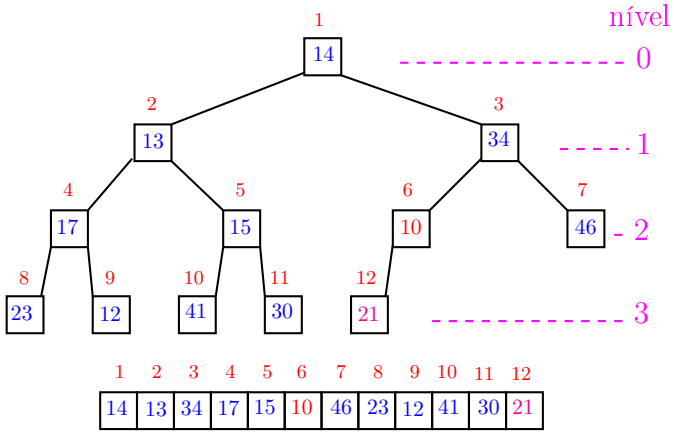
Verdade seja dita . . . (MAC0338)

O consumo de tempo da função peneira é proporcional a $O(\lg m/i)$.

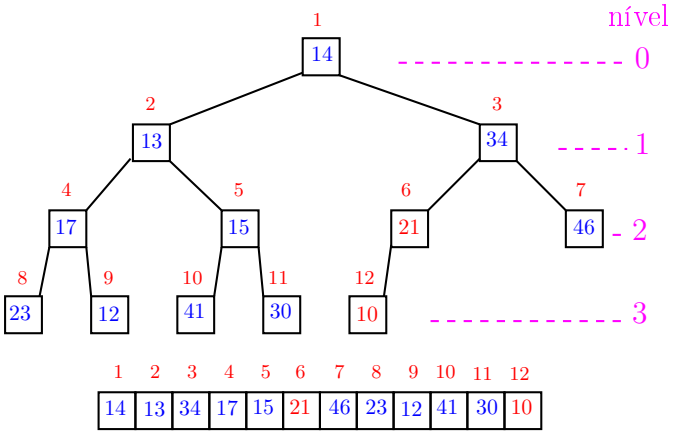
Construção de um max-heap



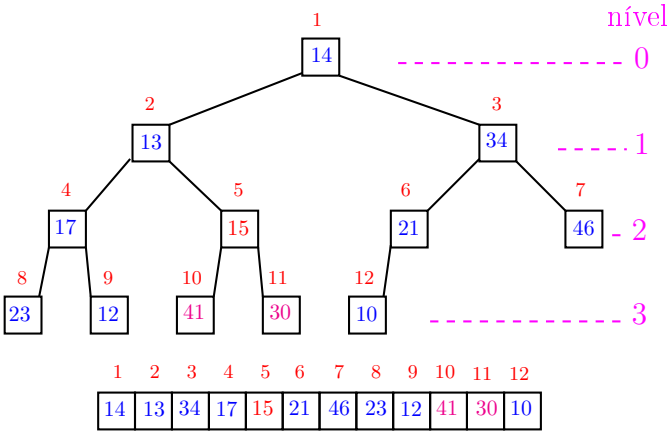
Construção de um max-heap



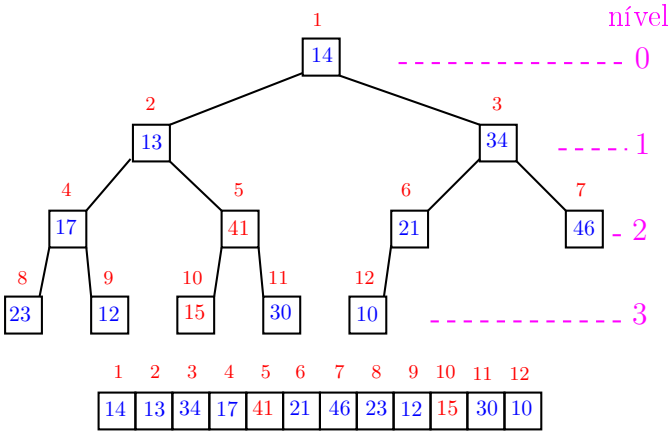
Construção de um max-heap



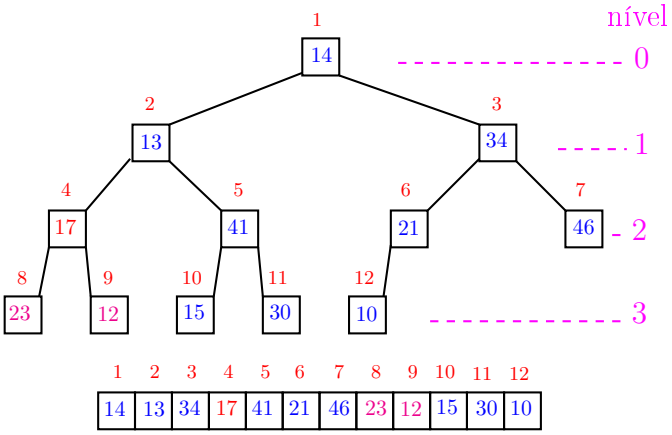
Construção de um max-heap



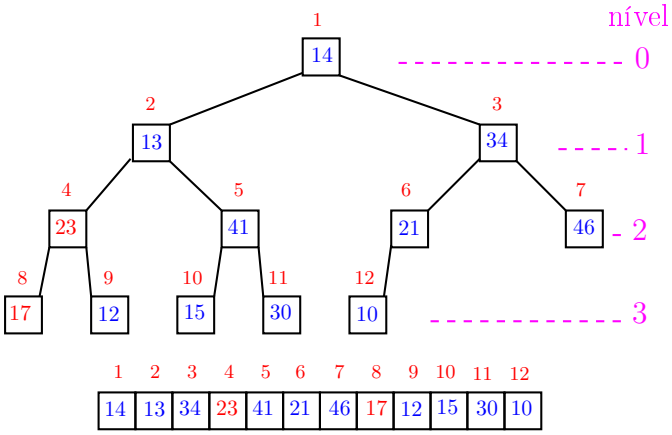
Construção de um max-heap



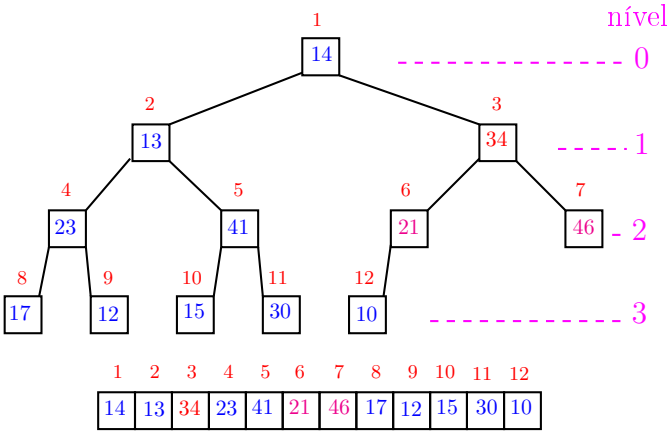
Construção de um max-heap



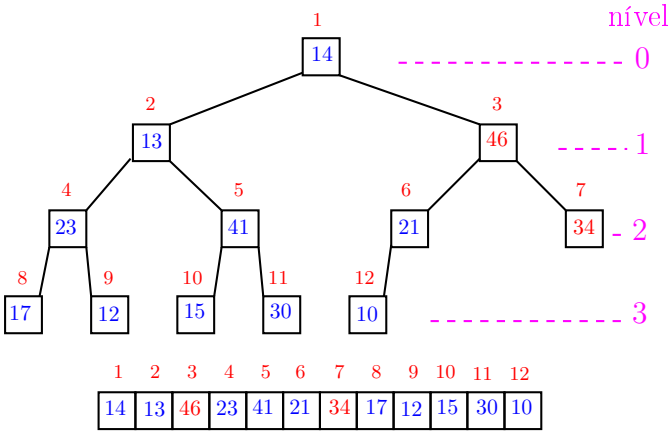
Construção de um max-heap



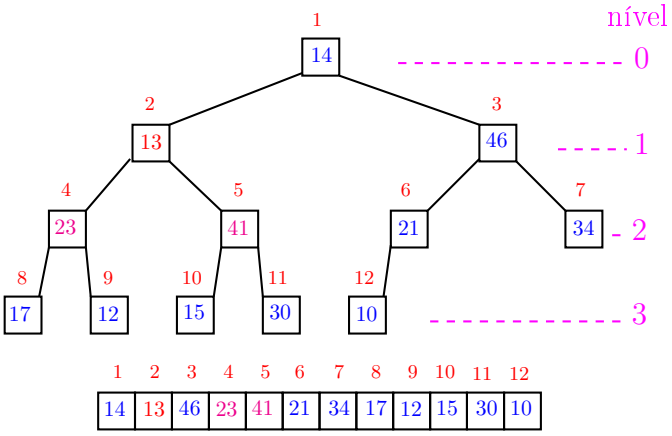
Construção de um max-heap



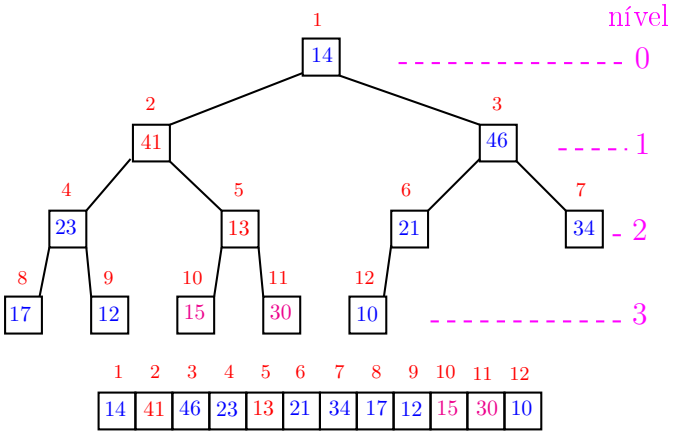
Construção de um max-heap



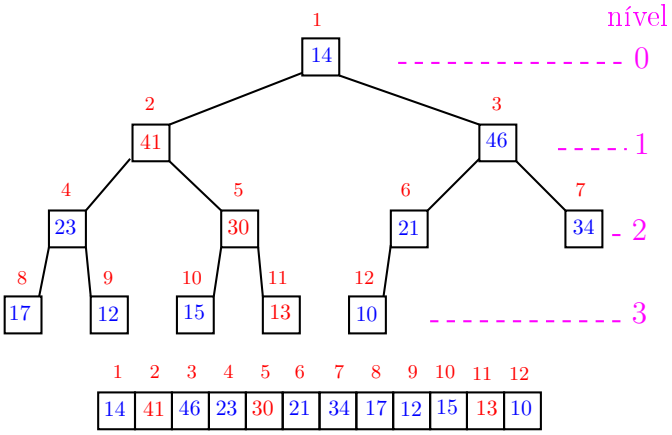
Construção de um max-heap



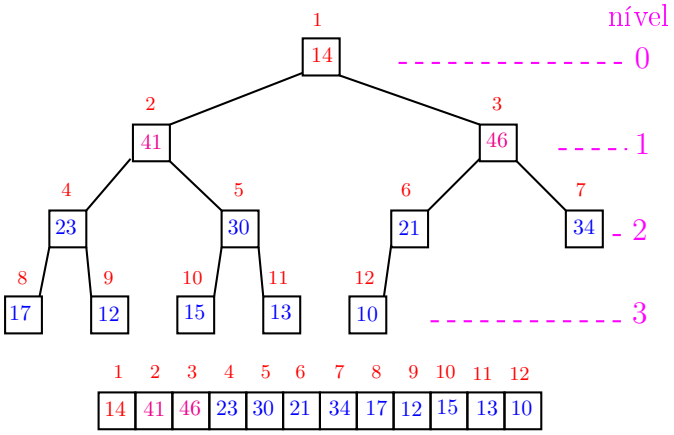
Construção de um max-heap



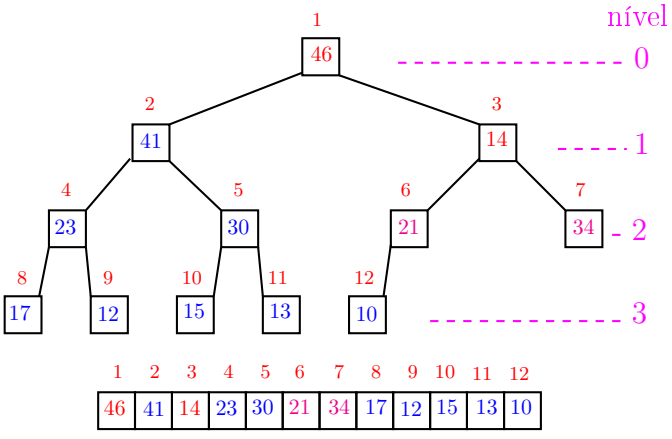
Construção de um max-heap



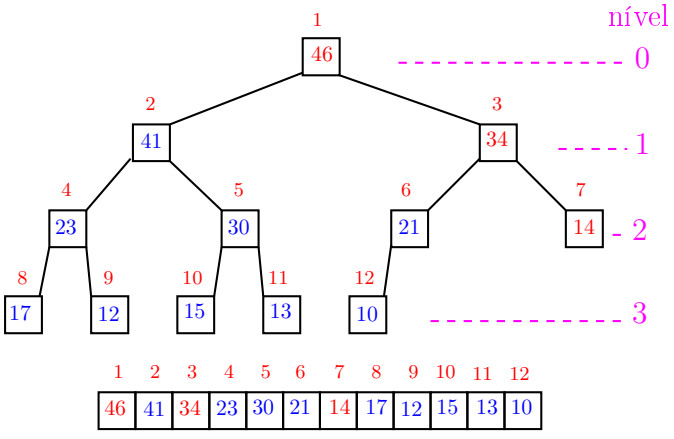
Construção de um max-heap



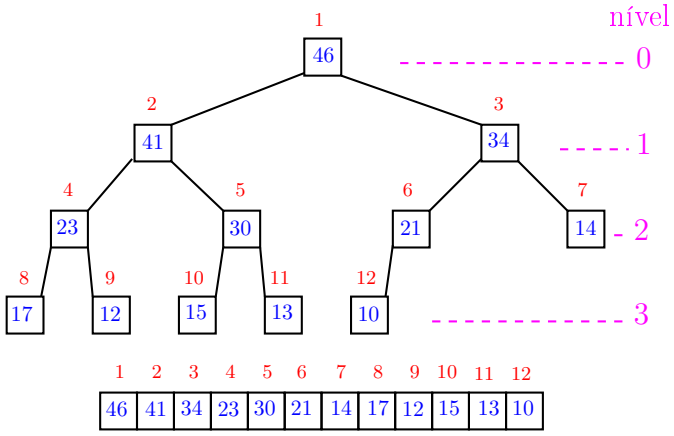
Construção de um max-heap



Construção de um max-heap



Construção de um max-heap



Construção de um max-heap

Recebe um vetor $v[1..n]$ e rearranja v para que seja max-heap.

```
1   for (i = n/2; /*A*/ i >= 1; i--)
2       peneira(i, n, v);
```

Relação invariante:

(i0) em /*A*/ vale que, $i+1, \ldots, n$ 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 ... (MAC0338)

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 ... (MAC0338)

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