AULA 7

Heapsort

CLRS 6

(veja tb CLRS B.5.3)

Merge sort ilustrou uma técnica (ou paradigma, ou estratégia) de concepção de algoritmos eficientes: a divisão e conquista.

Heapsort ilustra o uso de estruturas de dados no projeto de algoritmos eficientes.

Ordenação

$A[1 \mathinner{..} n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1 \mathinner{..} n]$ de mode que ele fique crescente.

Entra:

1										n
33	55	33	44	33	22	11	99	22	55	77

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \le \cdots \le A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de mode que ele fique crescente.

Entra:

1										n
33	55	33	44	33	22	11	99	22	55	77

Sai:

1										n
11	22	22	33	33	33	44	55	55	77	99

Ordenação por seleção

$m = 5$

1				max						n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

1	j		max							n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

1	j		max							n
38	50	20	44	10	50	55	60	75	85	99

j	max									n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

1	j		max							n
38	50	20	44	10	50	55	60	75	85	99

j	max									n
38	50	20	44	10	50	55	60	75	85	99

1	max									n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
20	10	38	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
20	10	38	44	50	50	55	60	75	85	99

1	m									n
20	10	38	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
20	10	38	44	50	50	55	60	75	85	99

1	m									n
10	20	38	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
20	10	38	44	50	50	55	60	75	85	99

1	m									n
10	20	38	44	50	50	55	60	75	85	99

1										n
10	20	38	44	50	50	55	60	75	85	99

Ordenação por seleção

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

ORDENA-POR-SELEÇÃO (A, n)

para $m \leftarrow n$ **decrescendo até** 2 **faça**

    $max \leftarrow m$

    para $j \leftarrow m - 1$ **decrescendo até** 1 **faça**

        se $A[max] < A[j]$

            então $max \leftarrow j$

    $A[m] \leftrightarrow A[max]$

Invariantes

Relação invariante chave:

(i0) na linha 1 vale que: $A[m\,..\,n]$ é crescente.

1				m						n
38	50	20	44	10	50	55	60	75	85	99

Invariantes

Relação invariante chave:

(i0) na linha 1 vale que: $A[m\,..\,n]$ é crescente.

1				m						n
38	50	20	44	10	50	55	60	75	85	99

Supondo que a invariante vale. Correção do algoritmo é evidente.

No início da última iteração das linhas 1–6 tem-se que $m = 1$. Da invariante conclui-se que $A[1\,..\,n]$ é crescente.

Mais invariantes

Na linha 1 vale que:

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

Na linha 3 vale que:

(i2) $A[j+1\,..\,m] \leq A[max]$

1	j		max		m					n
38	50	20	44	10	25	55	60	75	85	99

Mais invariantes

Na linha 1 vale que:

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

Na linha 3 vale que:

(i2) $A[j+1\,..\,m] \leq A[max]$

1	j		max		m					n
38	50	20	44	10	25	55	60	75	85	99

invariantes (i1),(i2)
$+$ condição de parada do **para** da linha 3
$+$ troca linha 6 $\Rightarrow$ validade (i0)

Verifique!

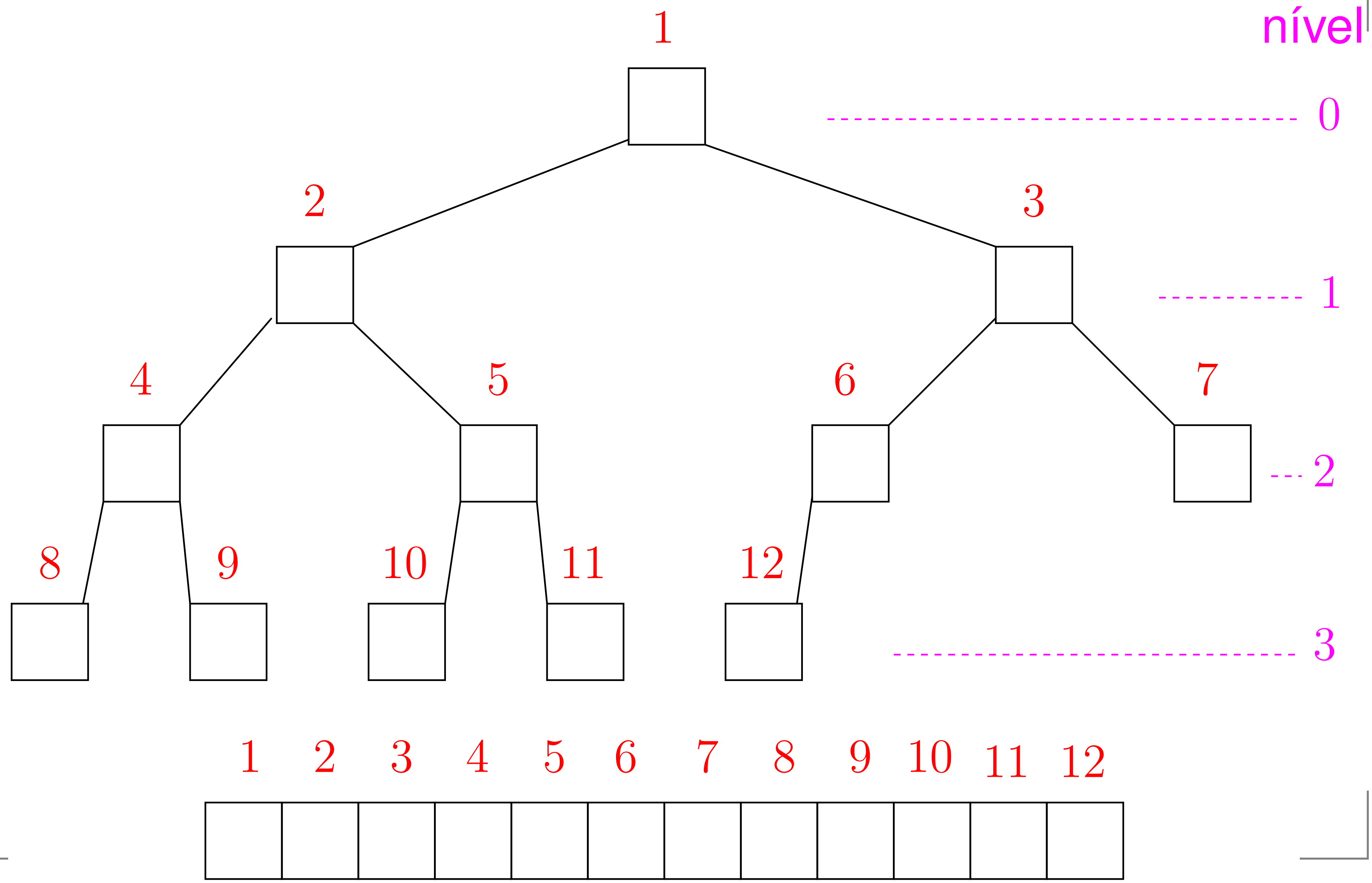
Consumo de tempo

linha		todas as execuções da linha
1	$=$	$\Theta(n)$
2	$=$	$\Theta(n)$
3	$=$	$\Theta(n^2)$
4	$=$	$\Theta(n^2)$
5	$=$	$\mathrm{O}(n^2)$
6	$=$	$\Theta(n)$
total	$=$	$\Theta(2n^2 + 3n) + \mathrm{O}(n^2) = \Theta(n^2)$

Conclusão

O consumo de tempo do algoritmo
ORDENA-POR-SELEÇÃO é $\Theta(n^2)$.

Representação de árvores em vetores



Pais e filhos

$A[1\,..\,m]$ 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.

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

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

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

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

Todo nó i é raiz da subárvore formada por

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

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 $\lfloor \lg i \rfloor$.

Prova: Se p é o nível do nó i, então

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

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

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

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

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

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

$$\langle \mathrm{filho}(i), \mathrm{filho}(\mathrm{filho}(i)), \mathrm{filho}(\mathrm{filho}(\mathrm{filho}(i))), \ldots \rangle$$

onde $\mathrm{filho}(i)$ 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

$$\langle \text{filho}(i), \text{filho}(\text{filho}(i)), \text{filho}(\text{filho}(\text{filho}(i))), \ldots \rangle$$

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

Os nós que têm altura zero são as folhas.

A altura de um nó i é ????.

Exercício 12.A

A altura de um nó i é o comprimento da seqüência

$$\langle 2\,i, 2^2\,i, 2^3\,i, \ldots, 2^h\,i \rangle$$

onde $2^h\,i \leq m < 2^{(h+1)}\,i$. Assim,

$$\begin{array}{rcccll} 2^h\,i & \leq & m & < & 2^{h+1} i & \Rightarrow \\ 2^h & \leq & m/i & < & \lg 2^{h+1} & \Rightarrow \\ h & \leq & \lg(m/i) & < & h+1 & \end{array}$$

Portanto, a altura de i é $\lfloor \lg(m/i) \rfloor$.

Exercício 12.B

Mostre que $A[1\,.\,.\,m]$ tem no máximo $\lceil m/2^{h+1} \rceil$ nós com altura h.

Exemplo: N_h = número de nós à altura h

m	$\lfloor m/2^{0+1} \rfloor$	N_0	$\lceil m/2^{0+1} \rceil$	$\lfloor m/2^{1+1} \rfloor$	N_1	$\lceil m/2^{1+1} \rceil$
16	8	8	8	4	4	4
17	8	9	9	4	4	5
18	9	9	9	4	5	5
19	9	10	10	4	5	5
20	10	10	10	5	5	5
21	10	11	11	5	5	6
22	11	11	11	5	6	6
23	11	12	12	5	6	6
24	12	12	12	6	6	6

Solução

Prova: Exercício 12.B

Resumão

Considere uma árvore representada em um vetor $A[1 \ldots m]$.

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:	$\lfloor \lg(m/i) \rfloor$
altura de uma folha:	0
total de nós de altura h	$\leq \lceil m/2^{h+1} \rceil$ (Exercício 12.B)

Heap

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

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

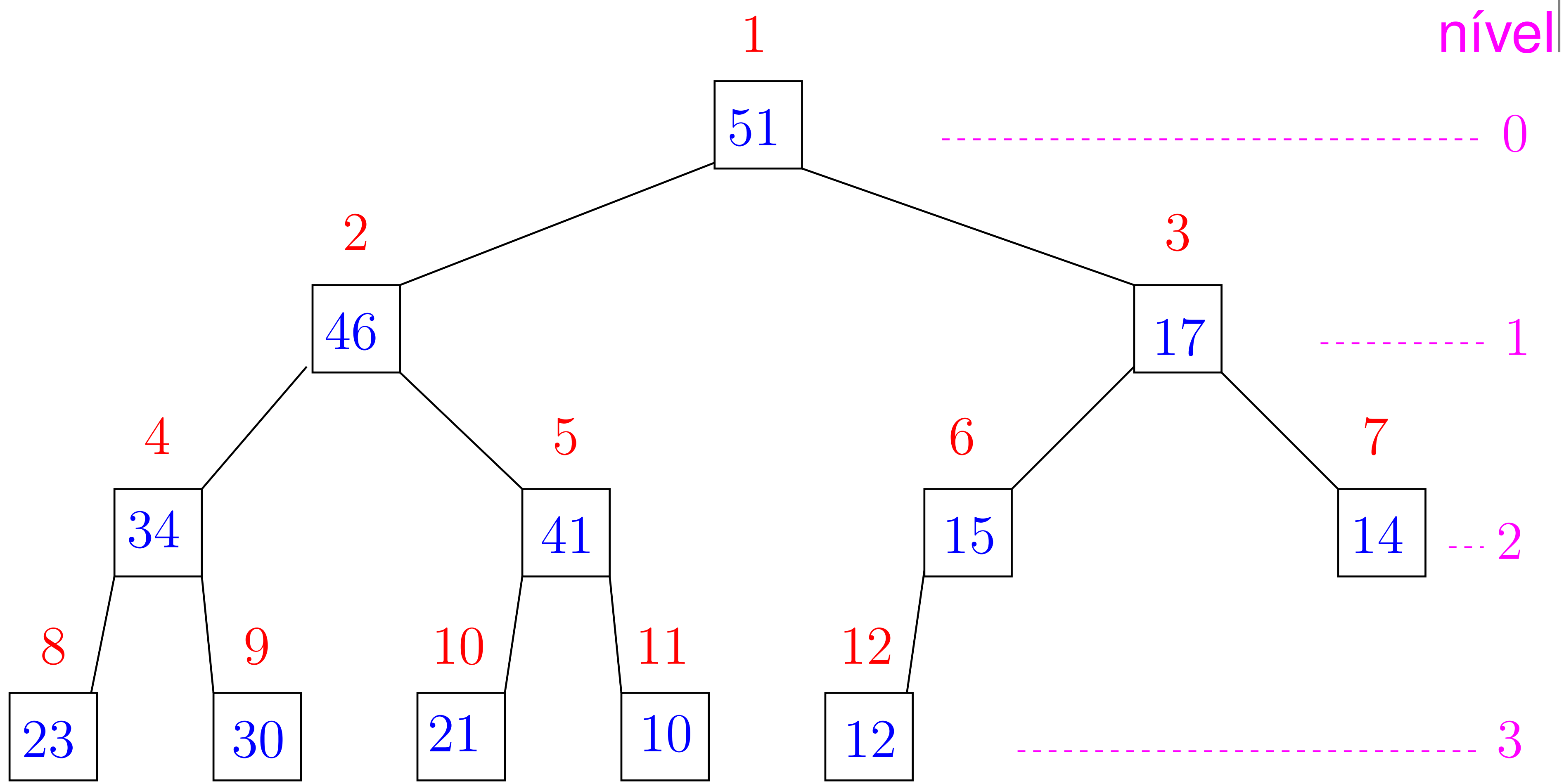
para todo $i = 2, 3, \ldots, m$.

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

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

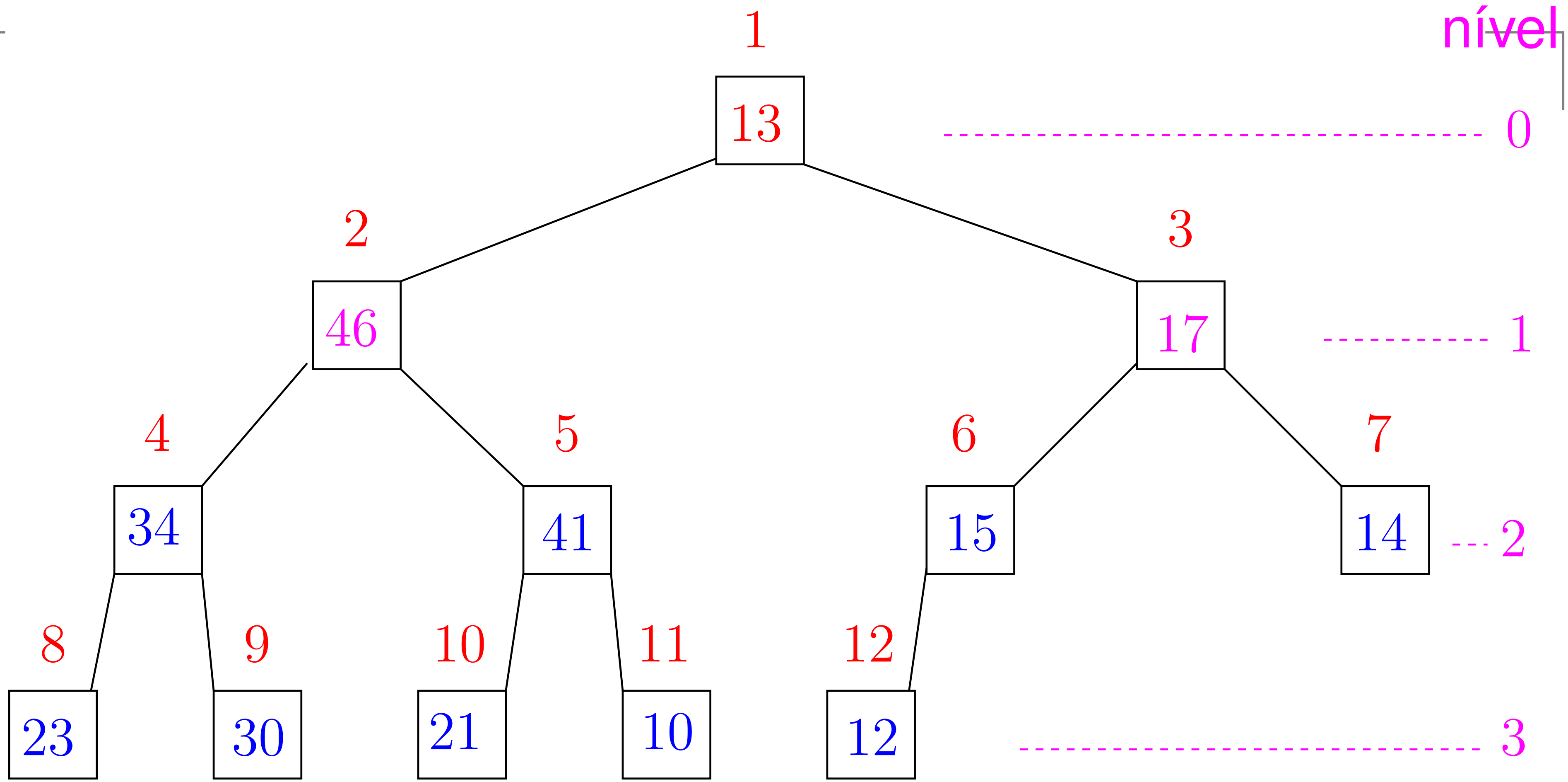
para todo $i = 2j, 2j+1, 4j, \ldots, 4j+3, 8j, \ldots, 8j+7, \ldots$.
Neste caso também diremos que a subárvore com raiz j é um max-heap.

Max-heap



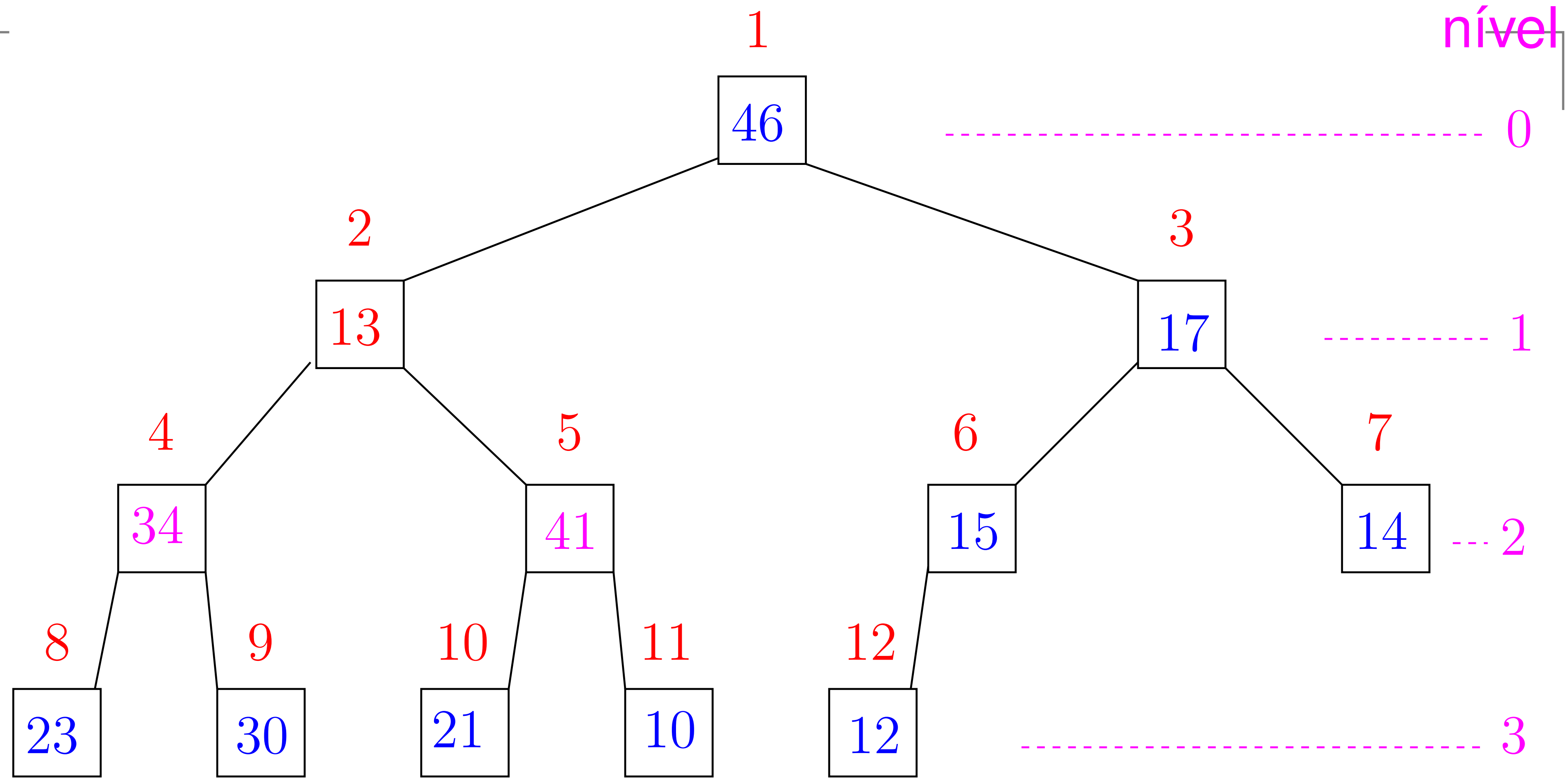
1	2	3	4	5	6	7	8	9	10	11	12
51	46	17	34	41	15	14	23	30	21	10	12

Rotina básica de manipulação de max-heap



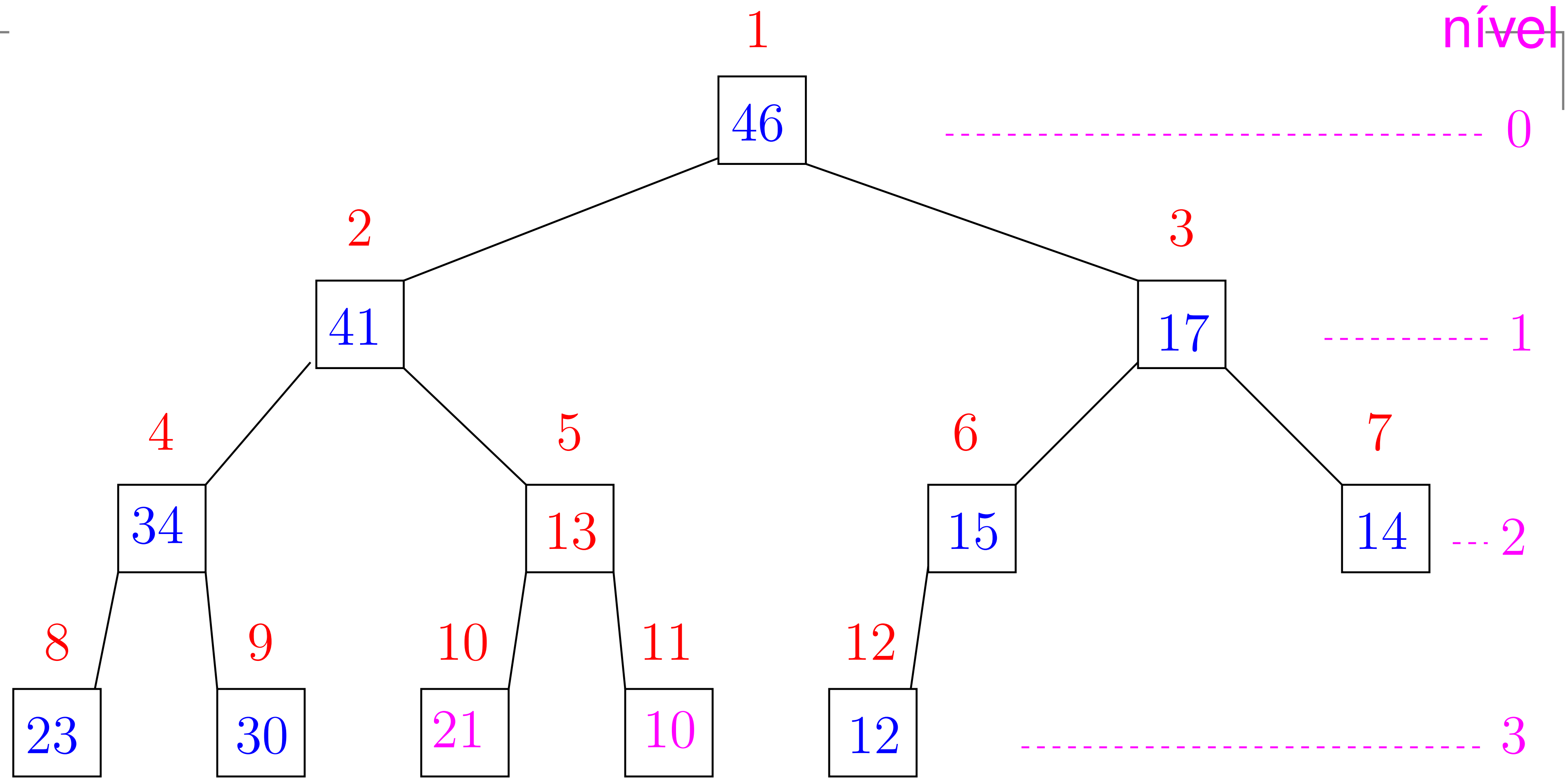
1	2	3	4	5	6	7	8	9	10	11	12
13	46	17	34	41	15	14	23	30	21	10	12

Rotina básica de manipulação de max-heap



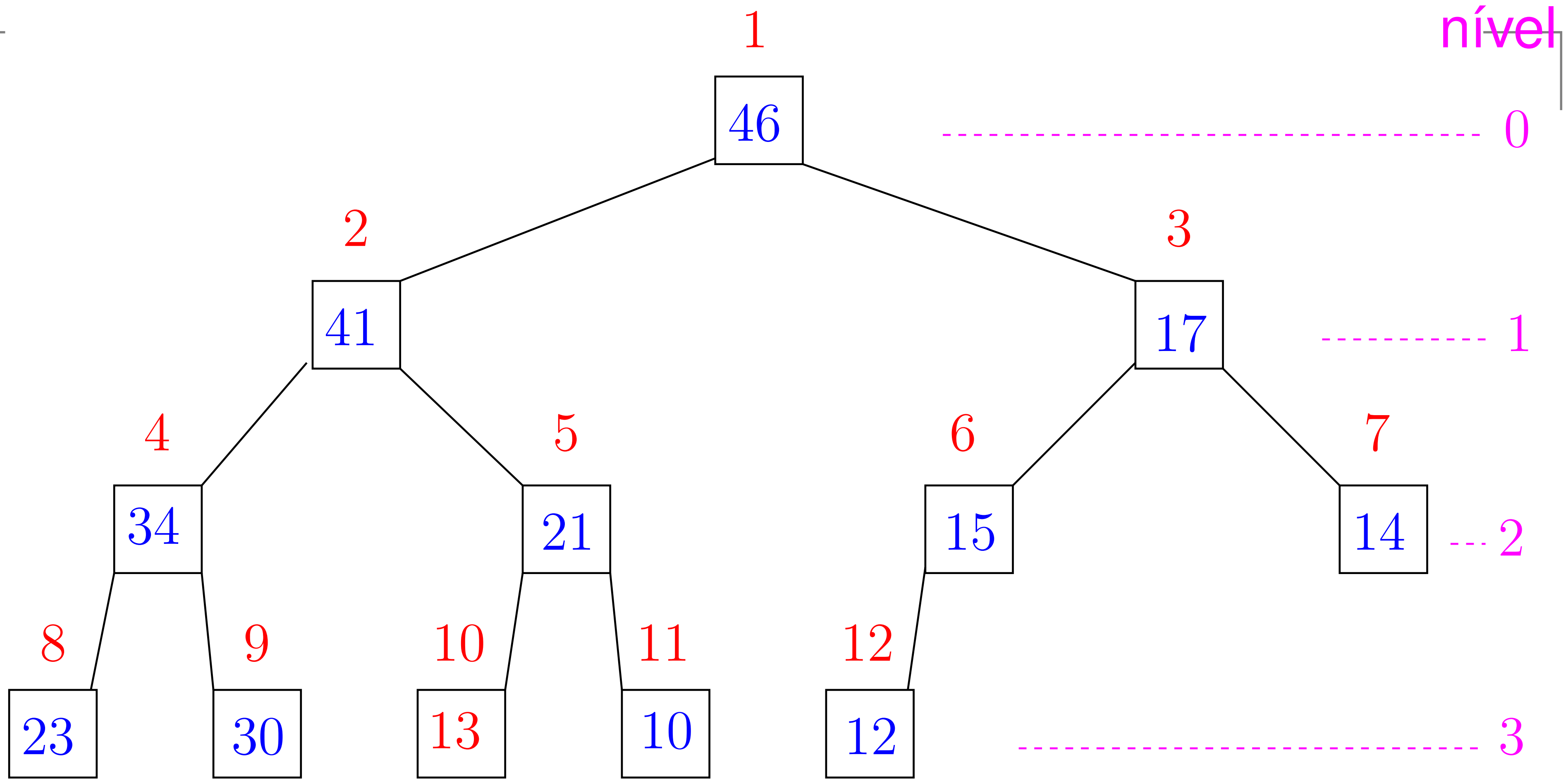
1	2	3	4	5	6	7	8	9	10	11	12
46	13	17	34	41	15	14	23	30	21	10	12

Rotina básica de manipulação de max-heap



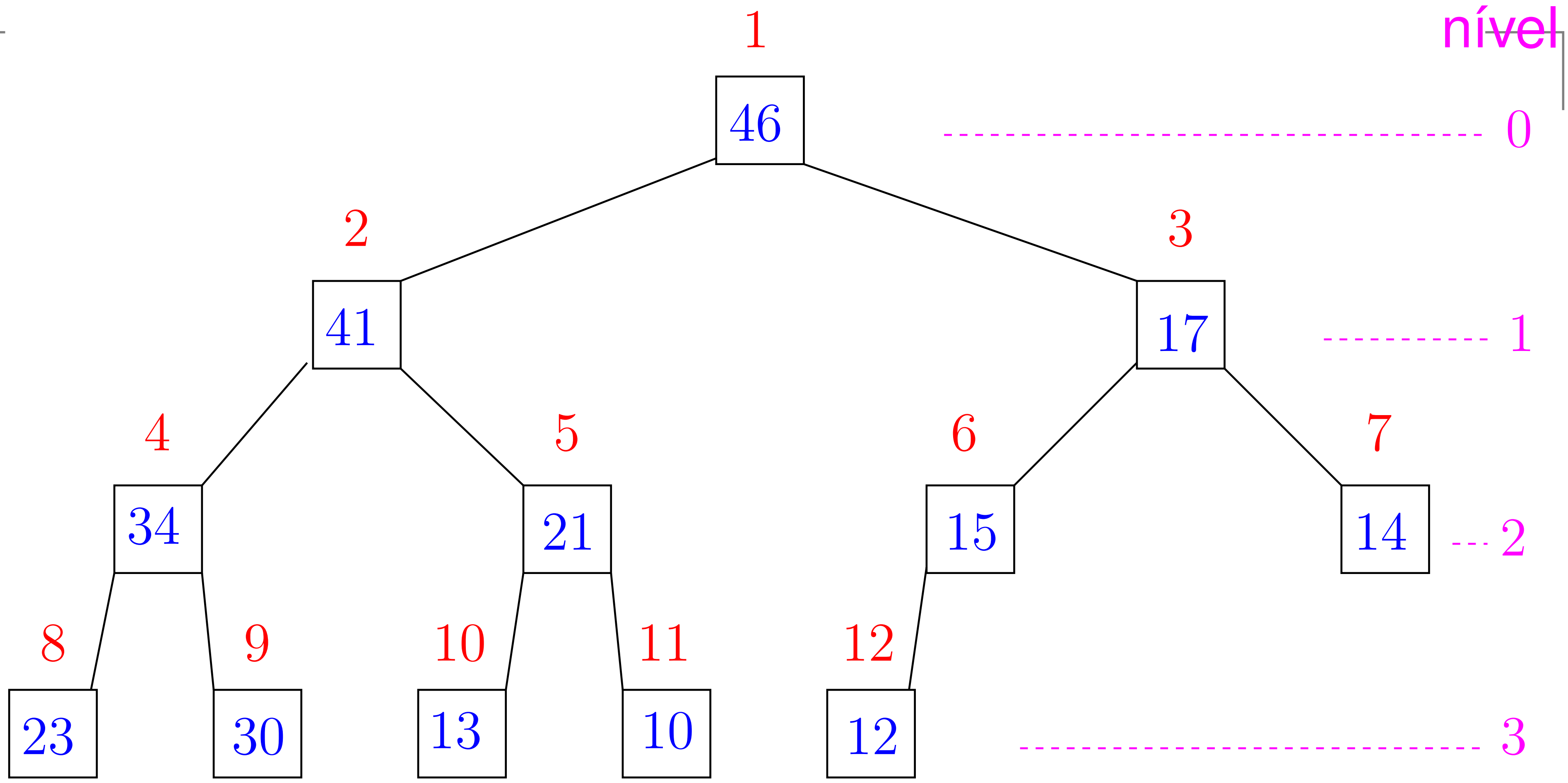
1	2	3	4	5	6	7	8	9	10	11	12
46	41	17	34	13	15	14	23	30	21	10	12

Rotina 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

Rotina 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

Rotina básica de manipulação de max-heap

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

MAX-HEAPIFY (A, m, i)

$e \leftarrow 2i$
$d \leftarrow 2i + 1$
se $e \leq m$ e $A[e] > A[i]$
    então $maior \leftarrow e$
    senão $maior \leftarrow i$
se $d \leq m$ e $A[d] > A[maior]$
    então $maior \leftarrow d$
se $maior \neq i$
    então $A[i] \leftrightarrow A[maior]$
        MAX-HEAPIFY $(A, m, maior)$

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	$=$	$3\,\Theta(1)$
4-5	$=$	$2\,\mathrm{O}(1)$
6	$=$	$\Theta(1)$
7	$=$	$\mathrm{O}(1)$
8	$=$	$\Theta(1)$
9	$=$	$\mathrm{O}(1)$
10	$\leq$	$T(h-1)$
total	$\leq$	$T(h-1) + \Theta(5) + \mathrm{O}(2)$

Consumo de tempo

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

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

h := altura de i = $\lfloor \lg \frac{m}{i} \rfloor$
$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

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

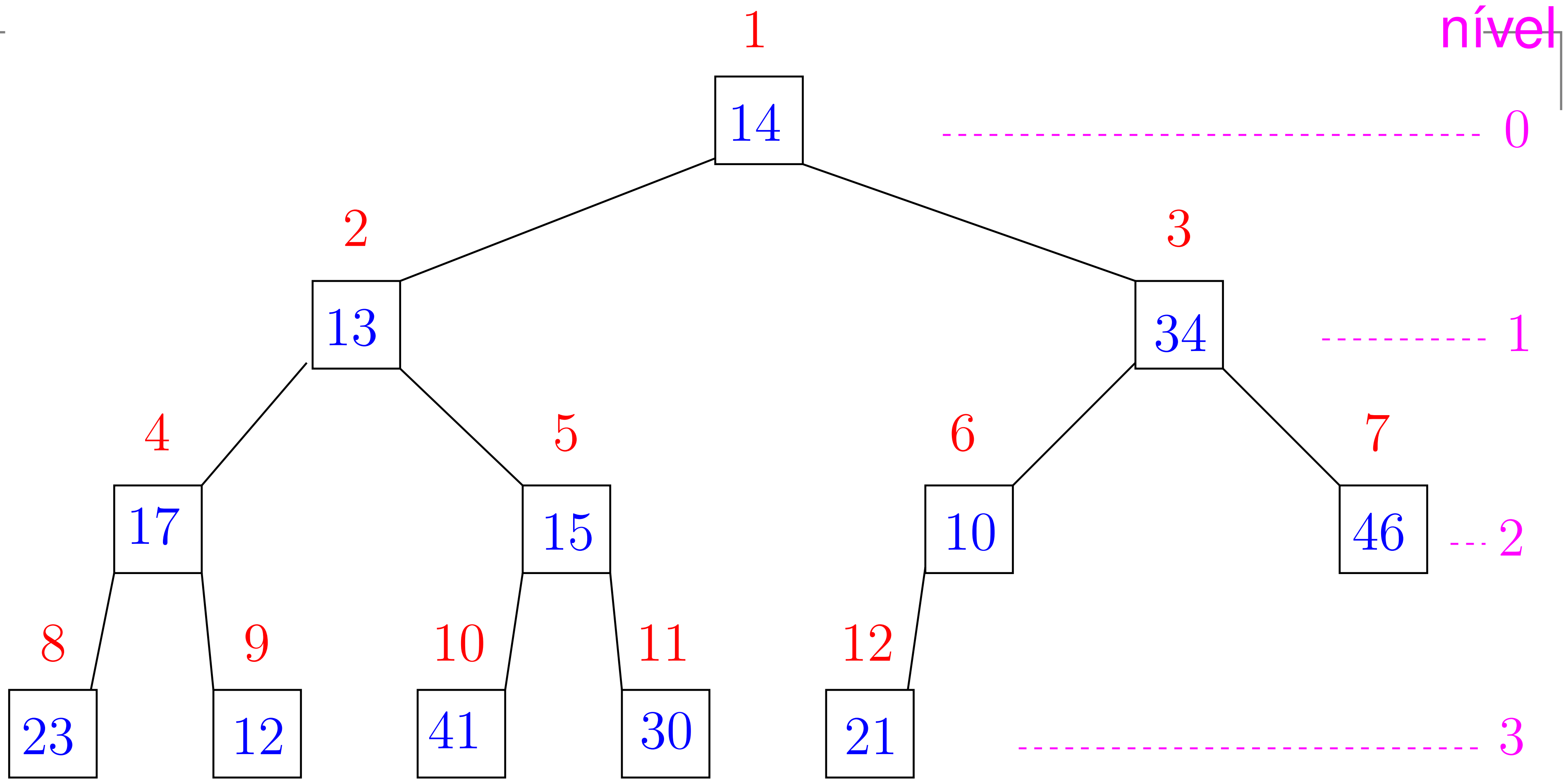
pois altura de $maior$ é $h-1$.

Solução assintótica: $T(h)$ é $\mathrm{O}(h)$.

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

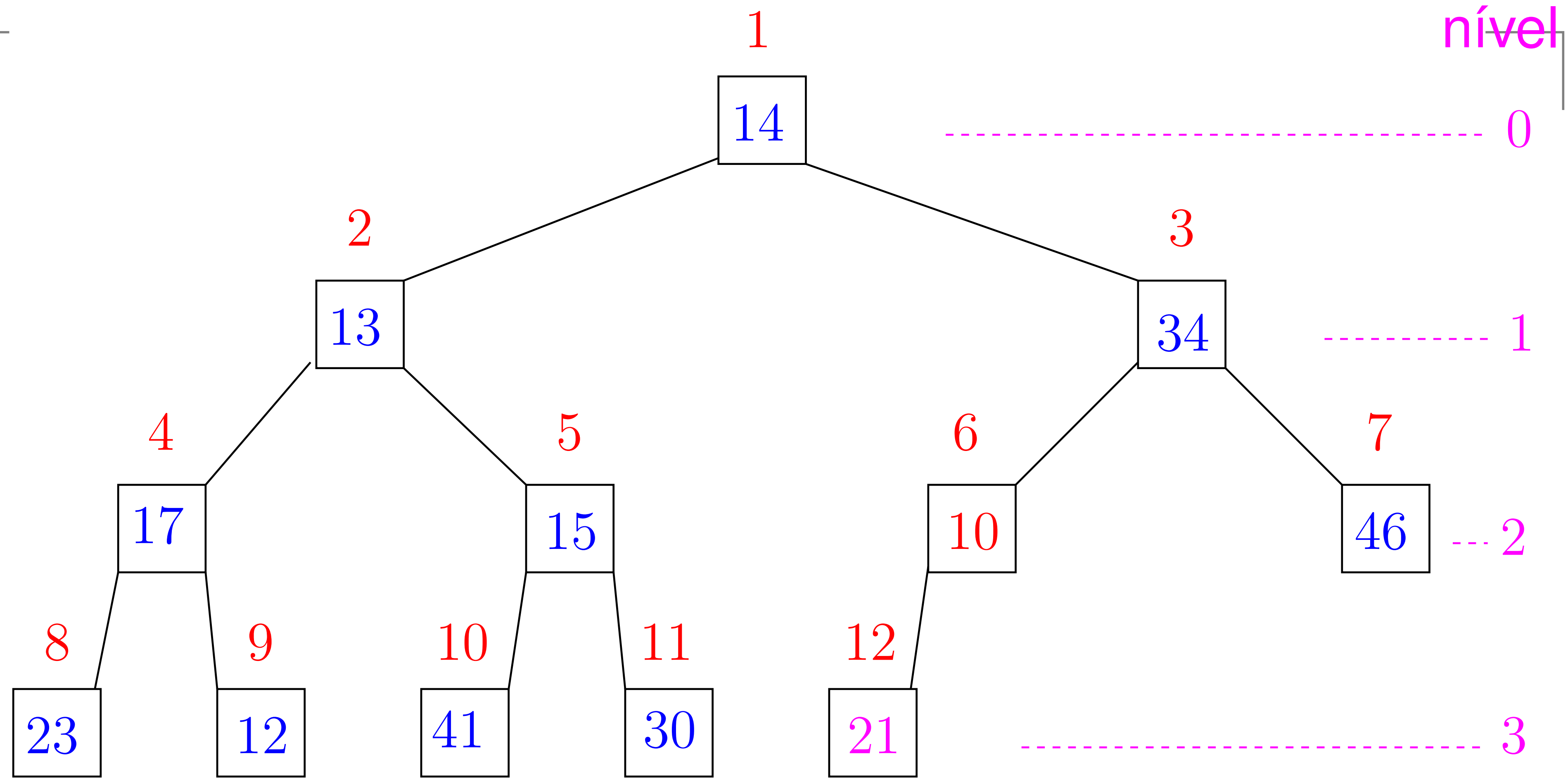
O consumo de tempo do algoritmo MAX-HEAPIFY é $\mathrm{O}(\lg m)$ (ou melhor ainda, $\mathrm{O}(\lg \frac{m}{i})$).

Construção de um max-heap



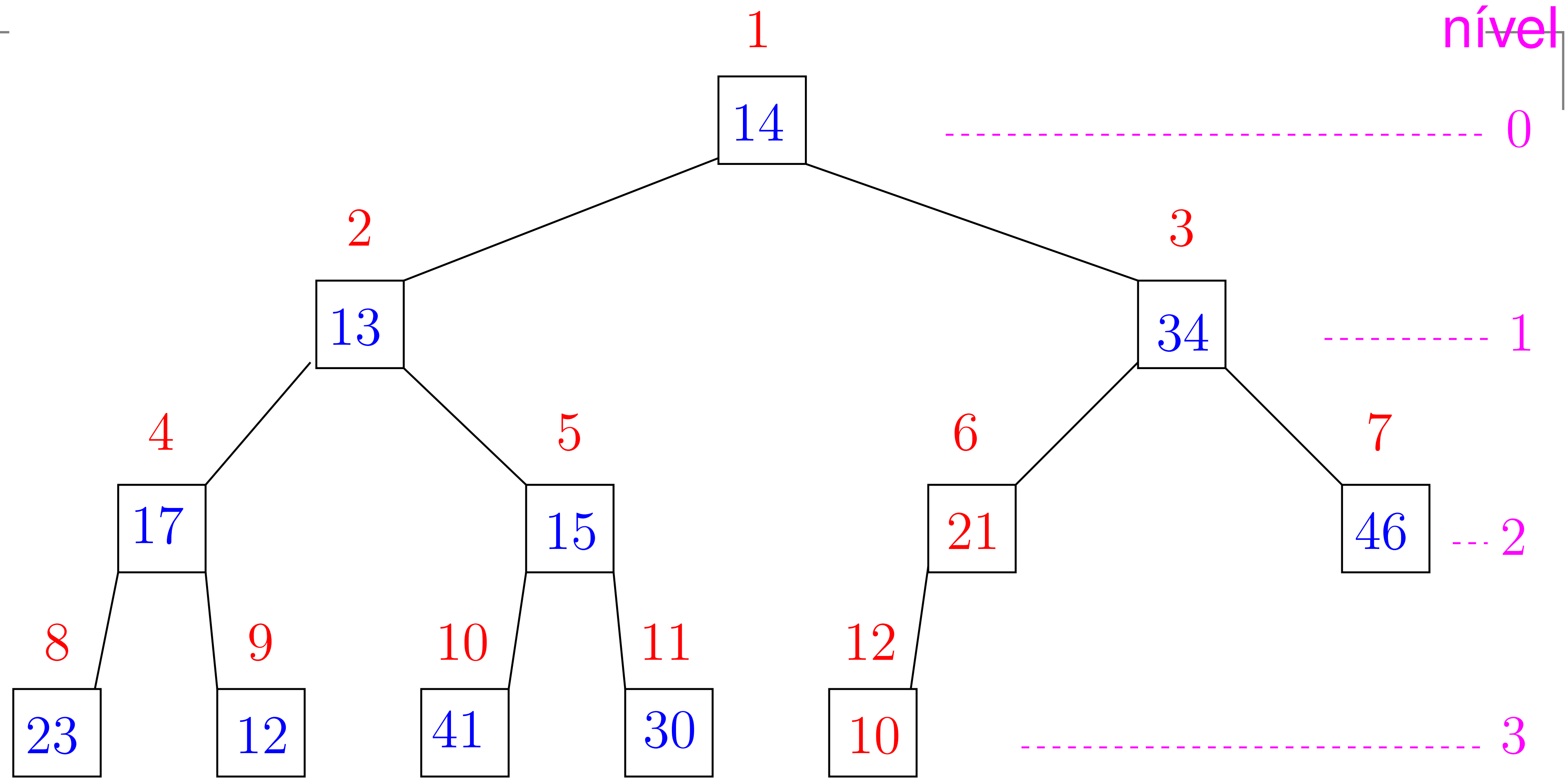
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	10	46	23	12	41	30	21

Construção de um max-heap



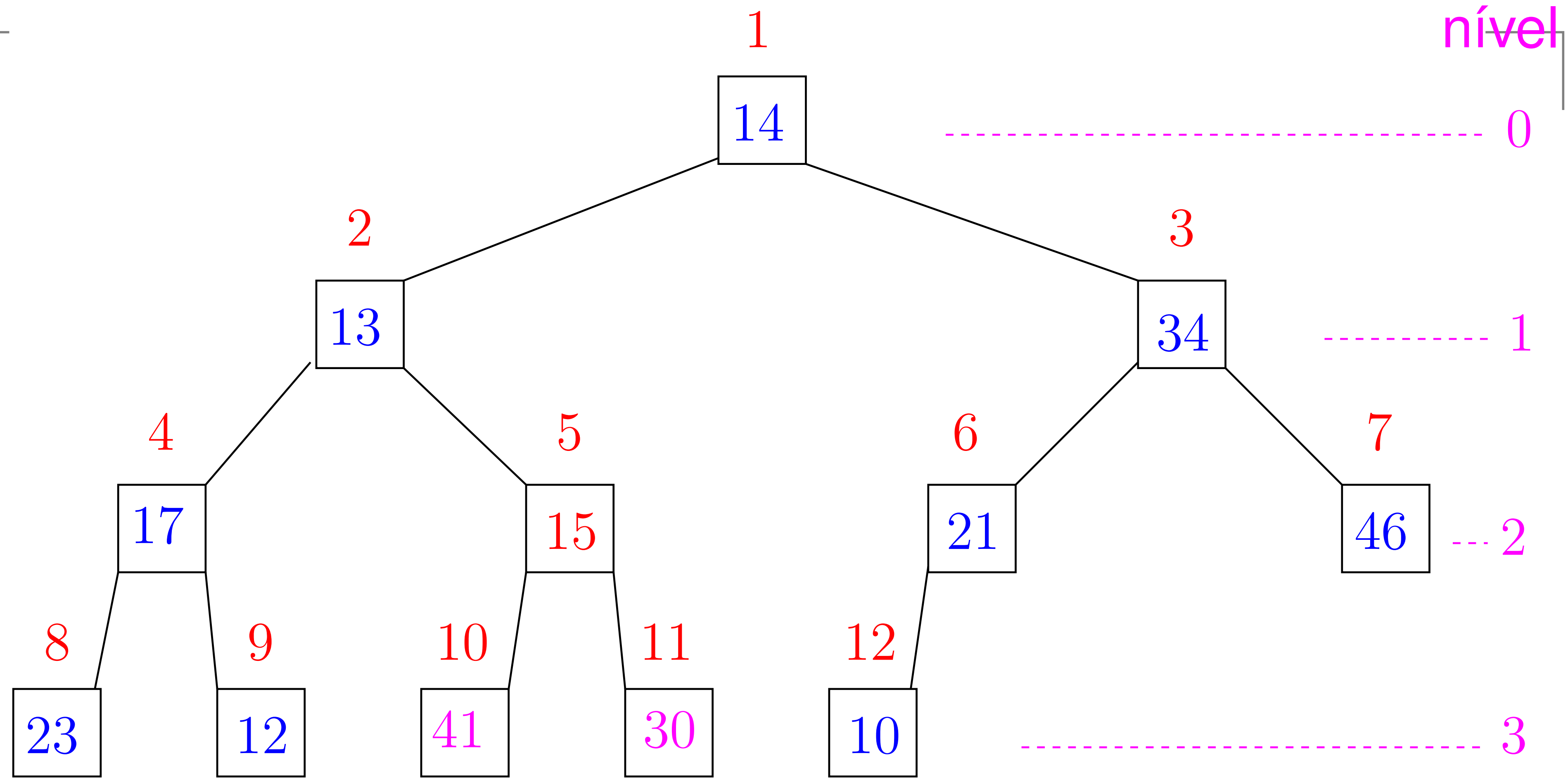
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	10	46	23	12	41	30	21

Construção de um max-heap



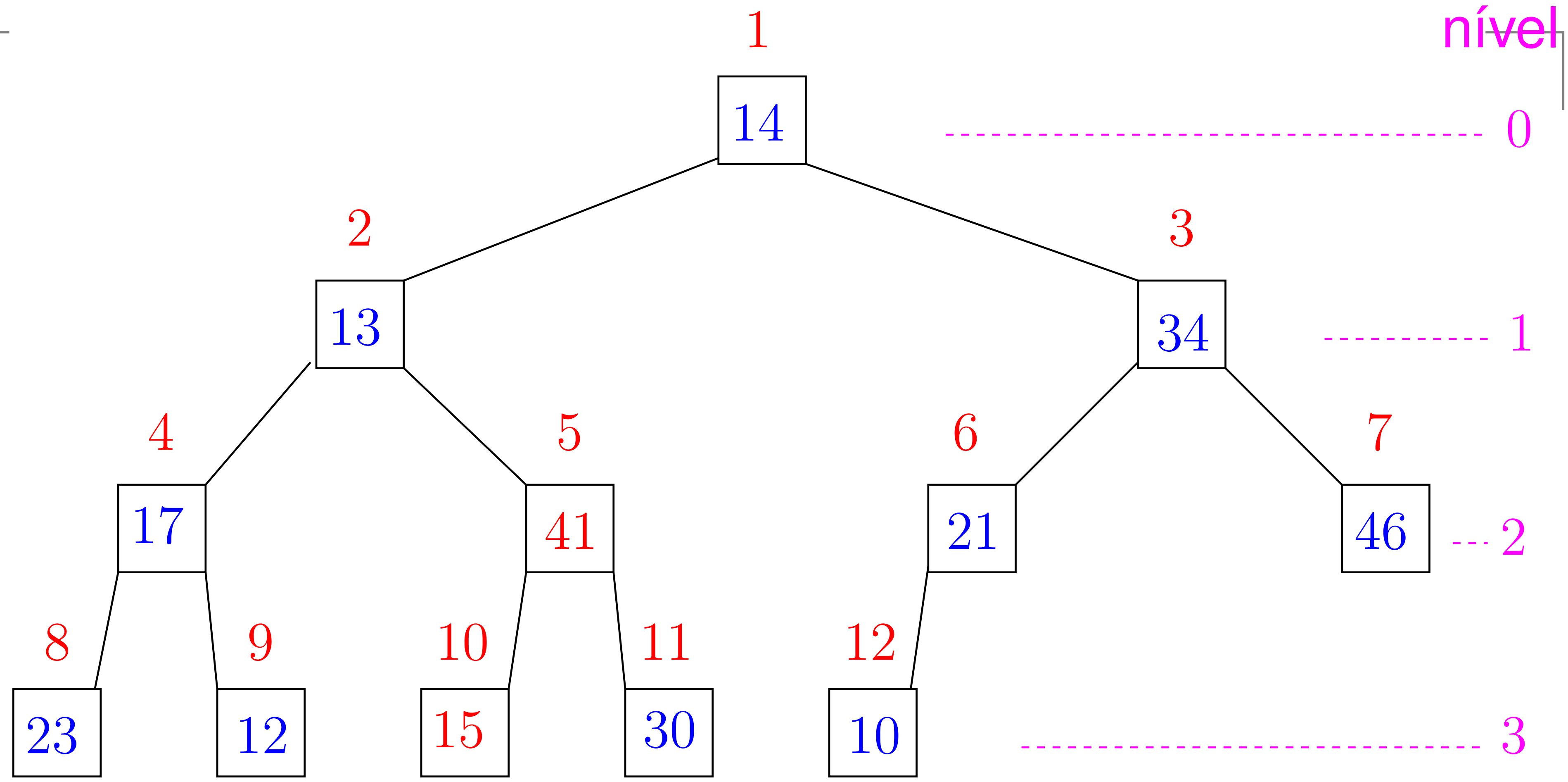
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	21	46	23	12	41	30	10

Construção de um max-heap



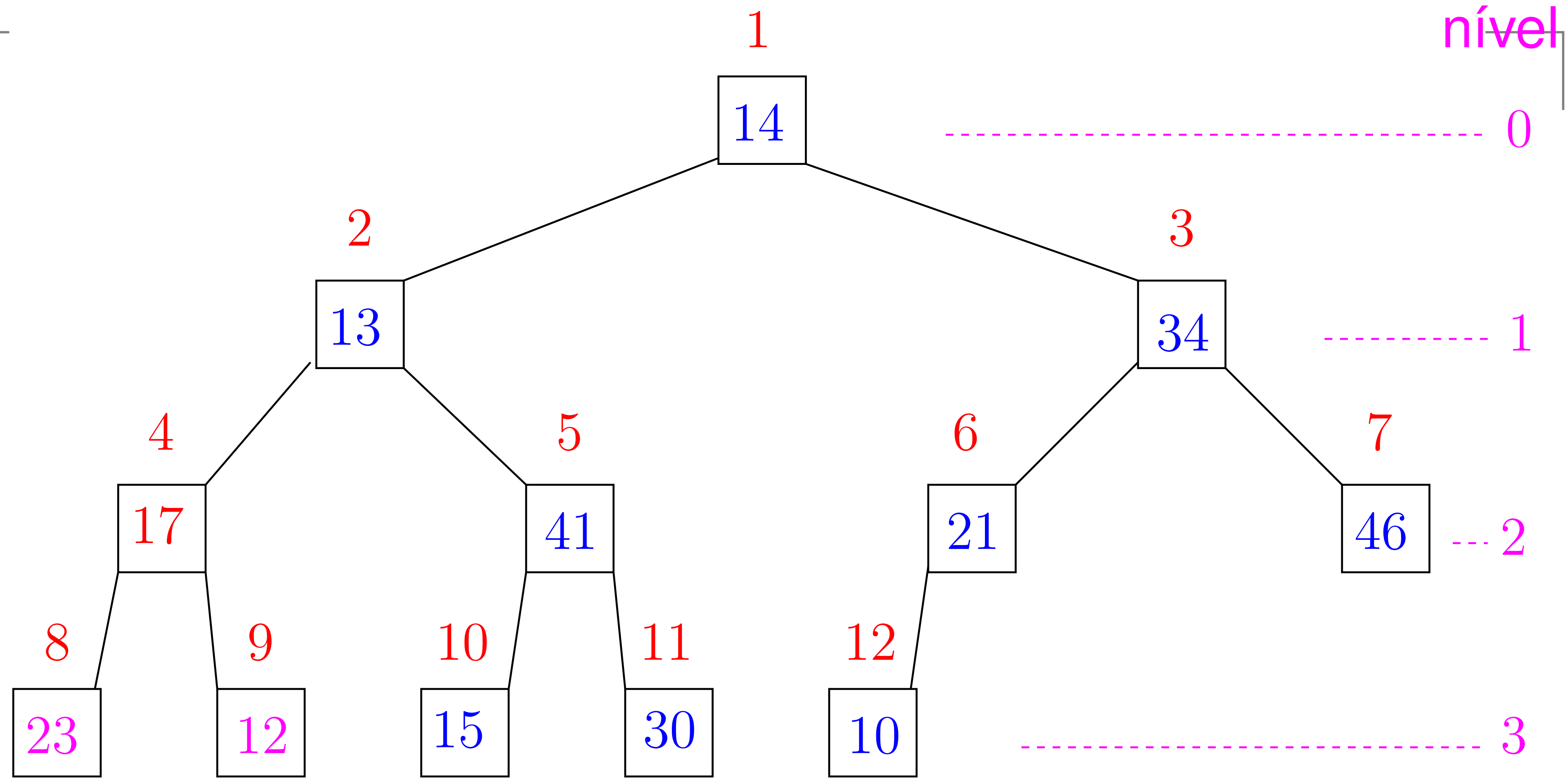
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	21	46	23	12	41	30	10

Construção de um max-heap



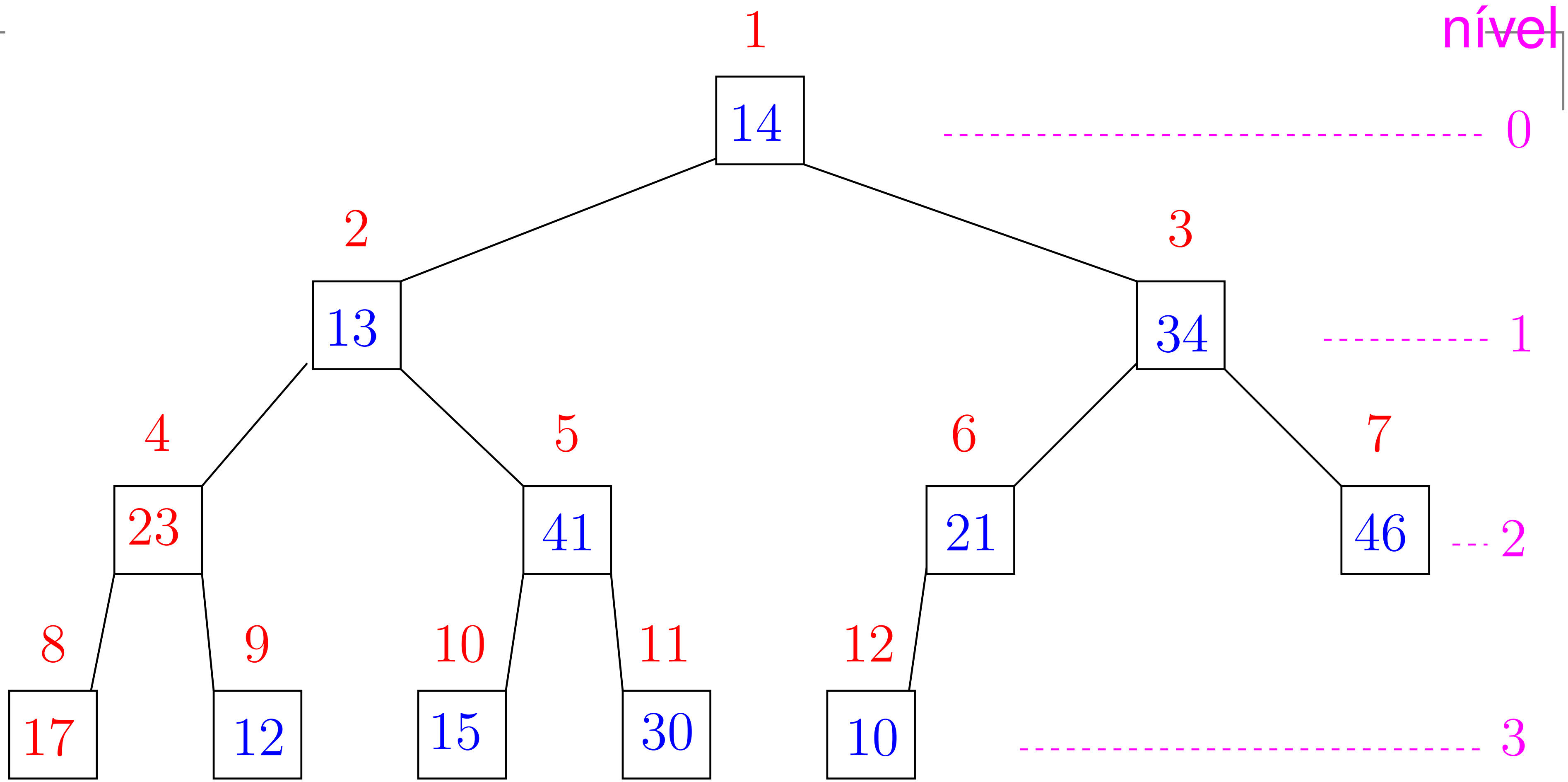
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	41	21	46	23	12	15	30	10

Construção de um max-heap



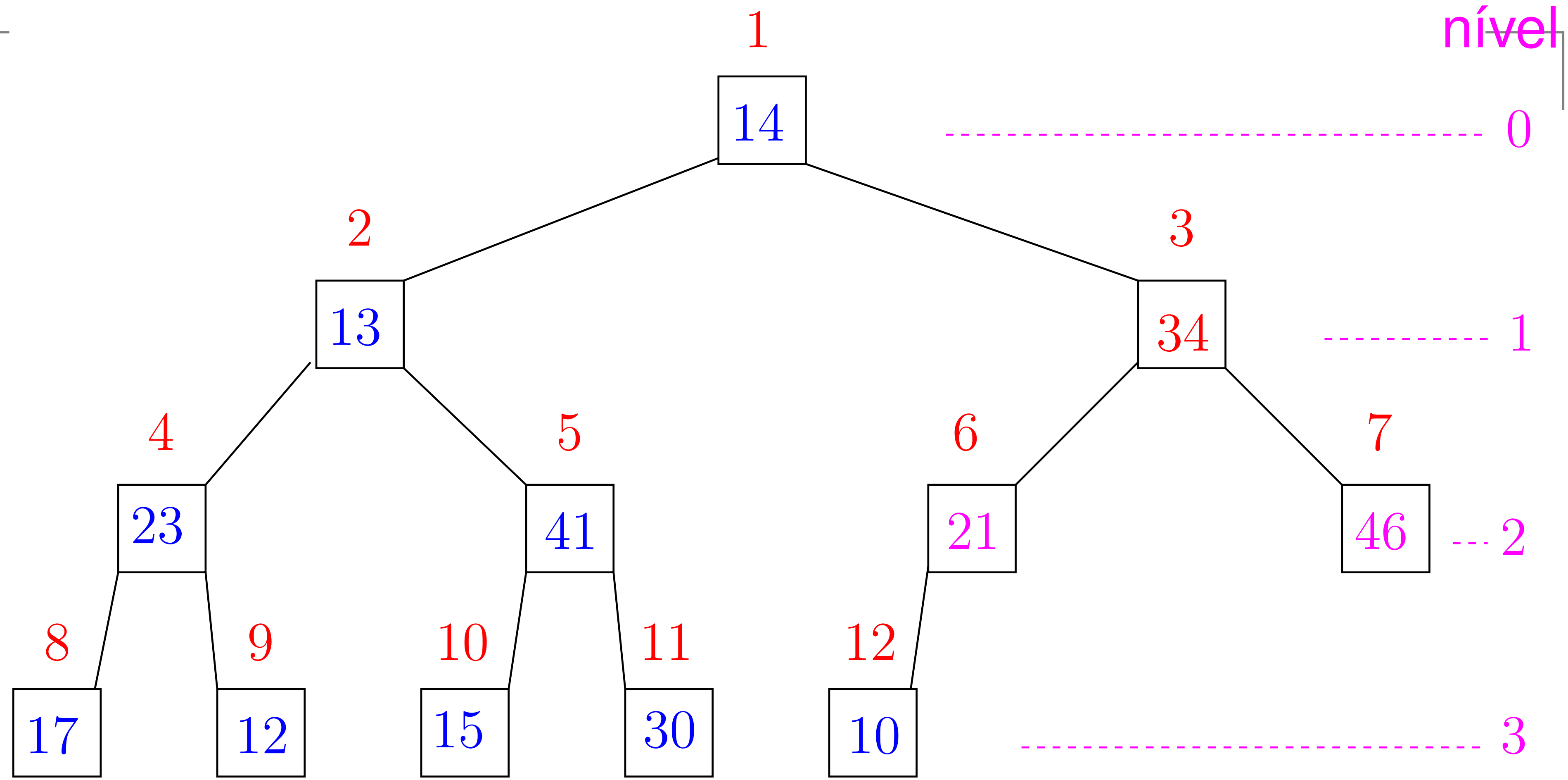
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	41	21	46	23	12	15	30	10

Construção de um max-heap



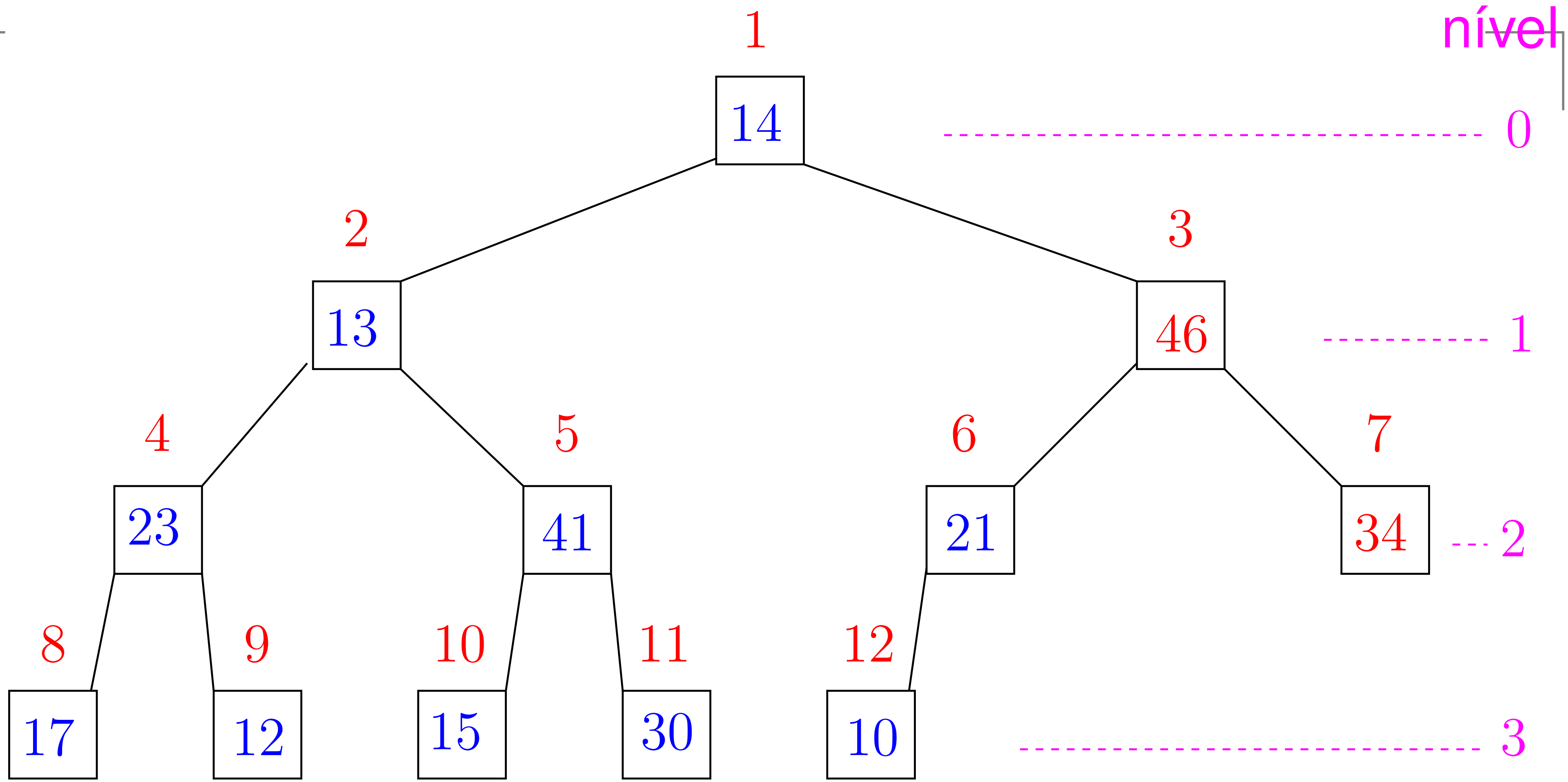
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	23	41	21	46	17	12	15	30	10

Construção de um max-heap



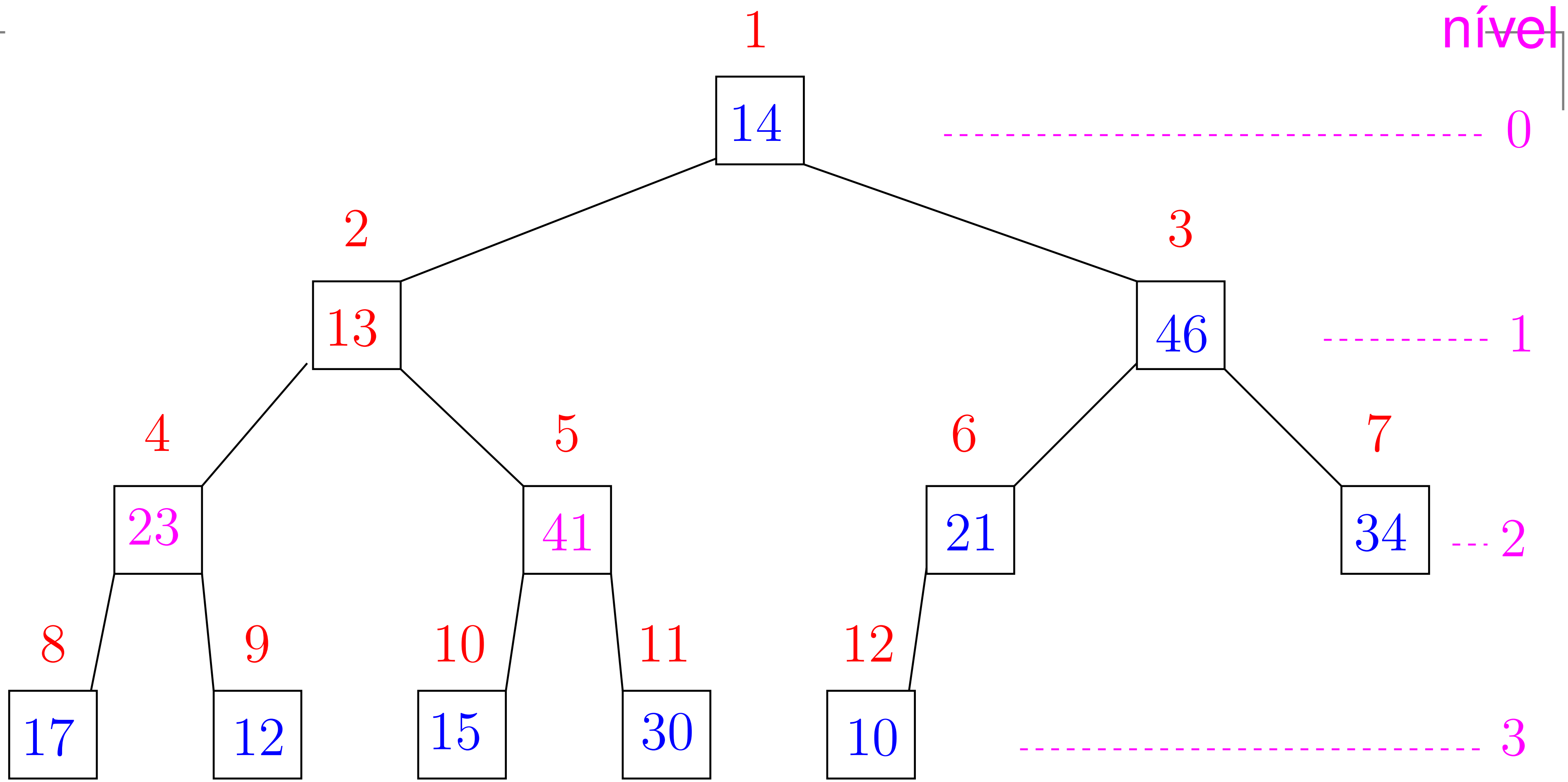
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	23	41	21	46	17	12	15	30	10

Construção de um max-heap



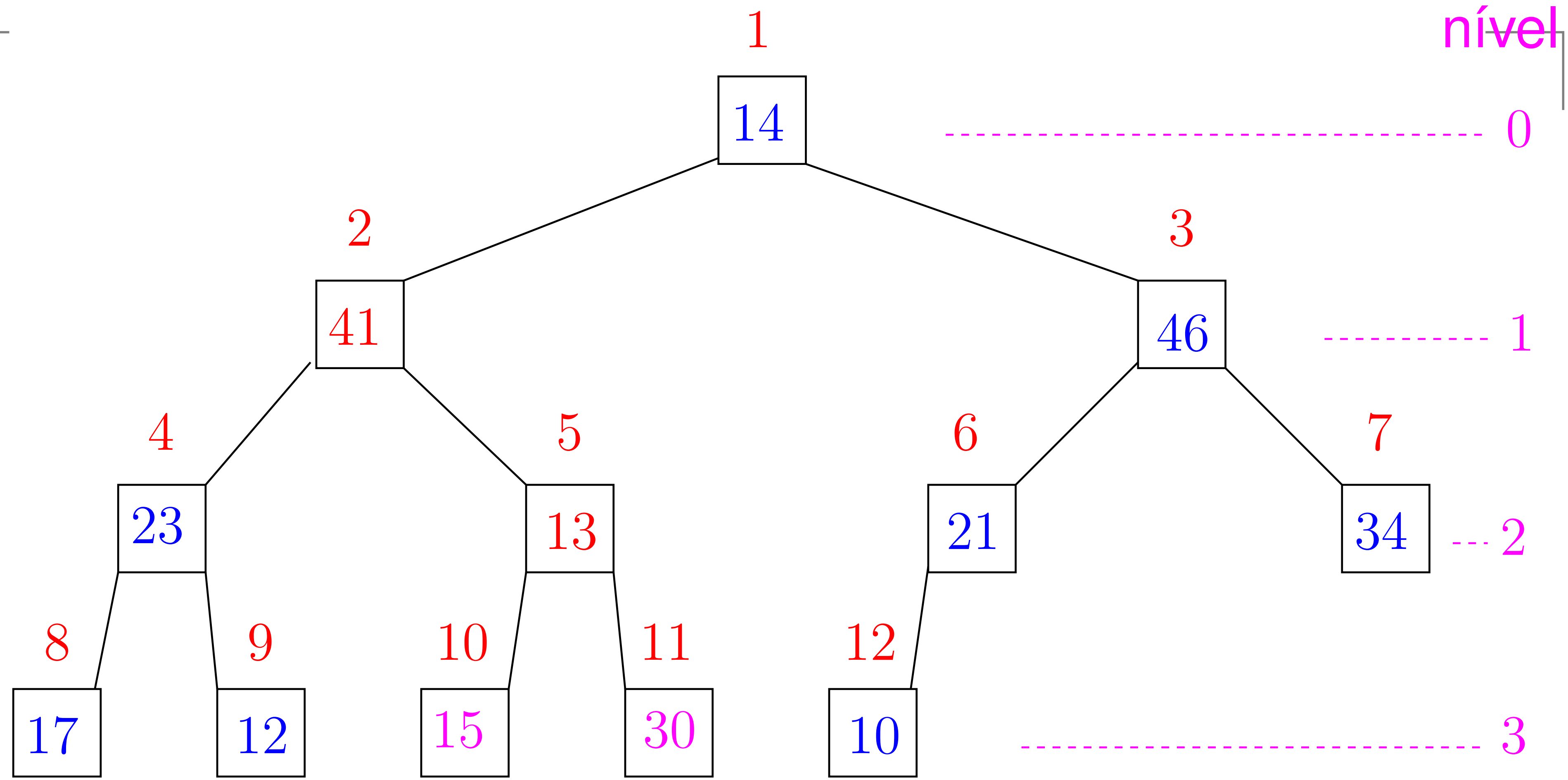
1	2	3	4	5	6	7	8	9	10	11	12
14	13	46	23	41	21	34	17	12	15	30	10

Construção de um max-heap



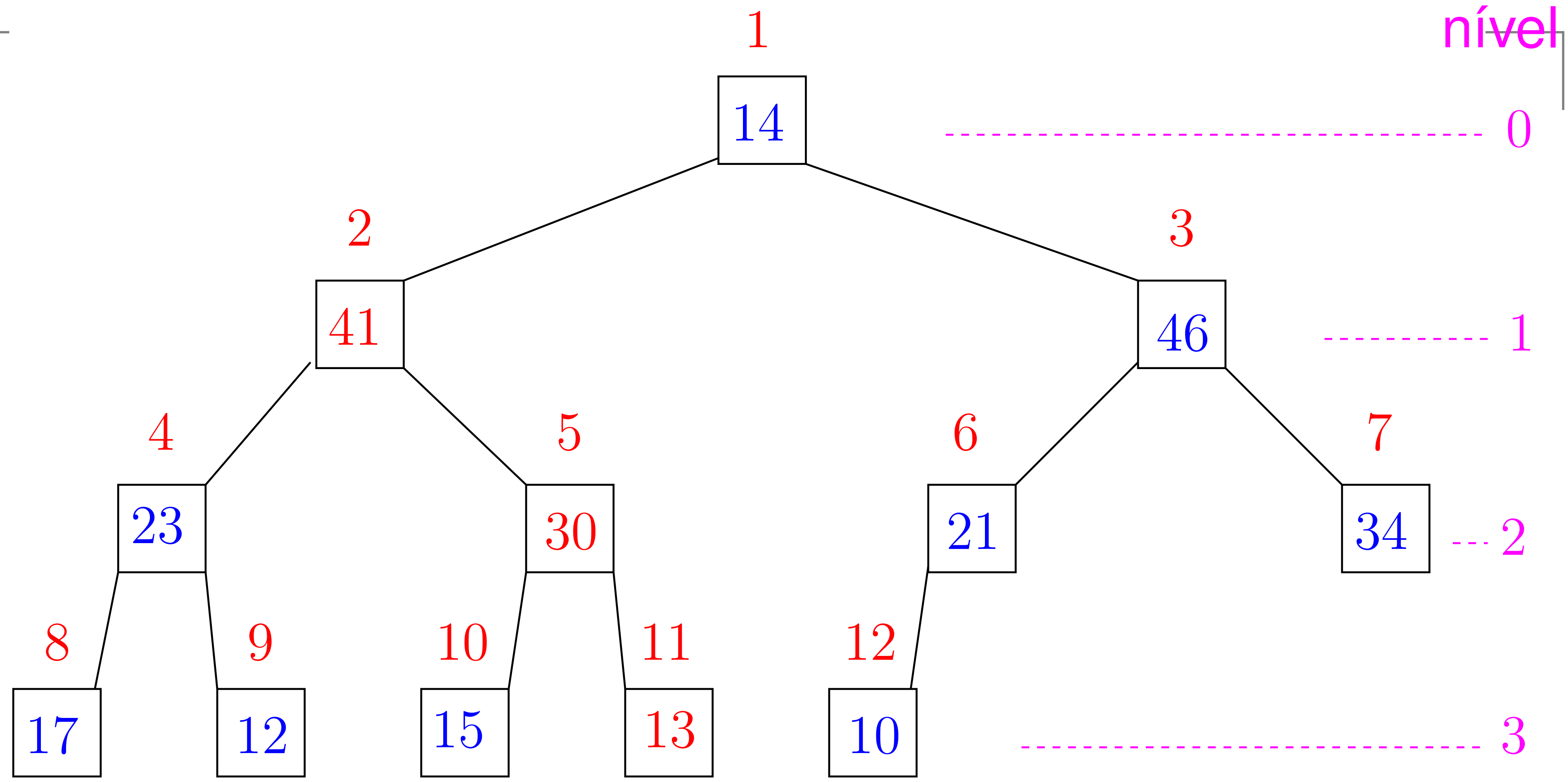
1	2	3	4	5	6	7	8	9	10	11	12
14	13	46	23	41	21	34	17	12	15	30	10

Construção de um max-heap



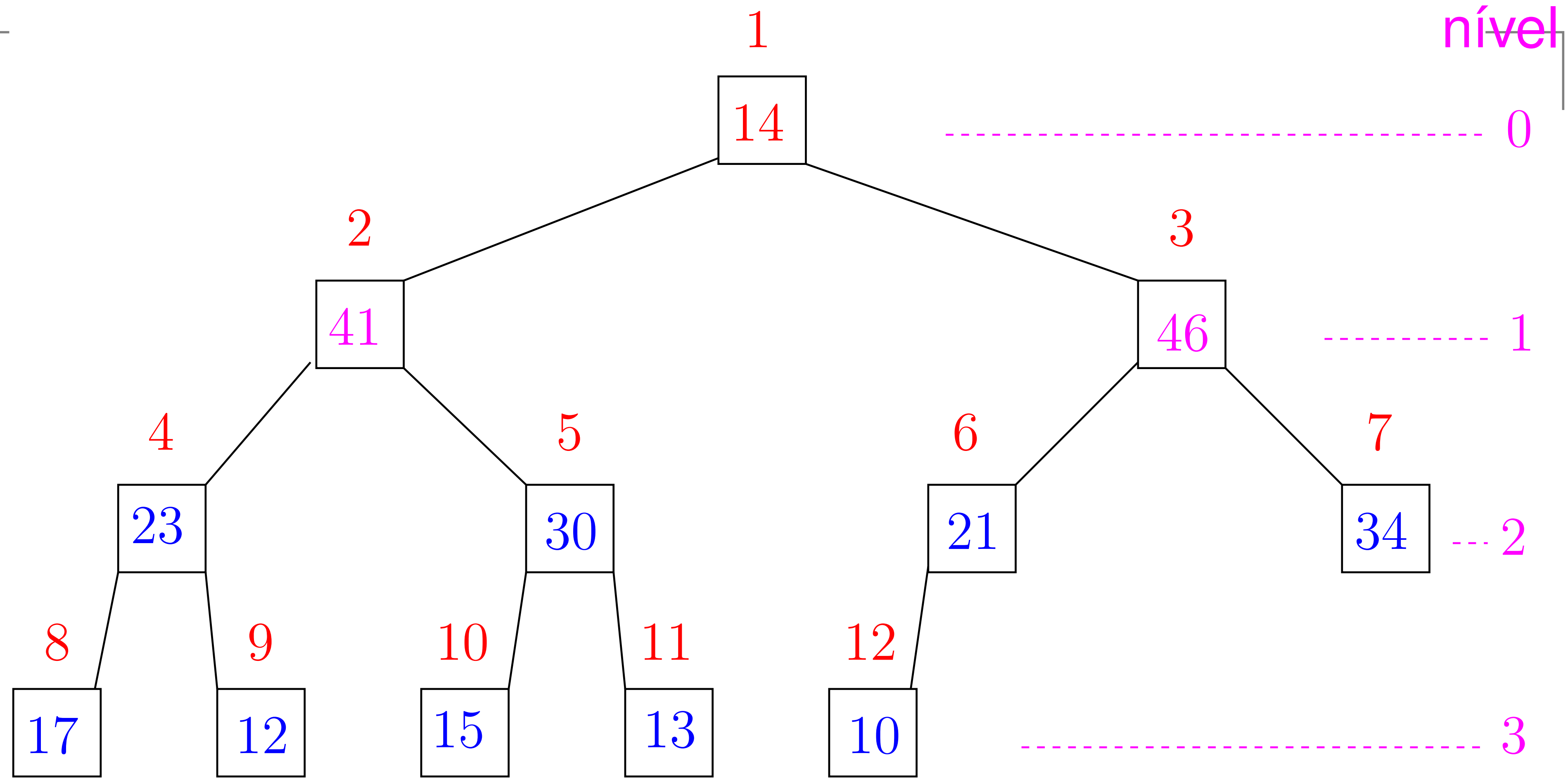
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	13	21	34	17	12	15	30	10

Construção de um max-heap



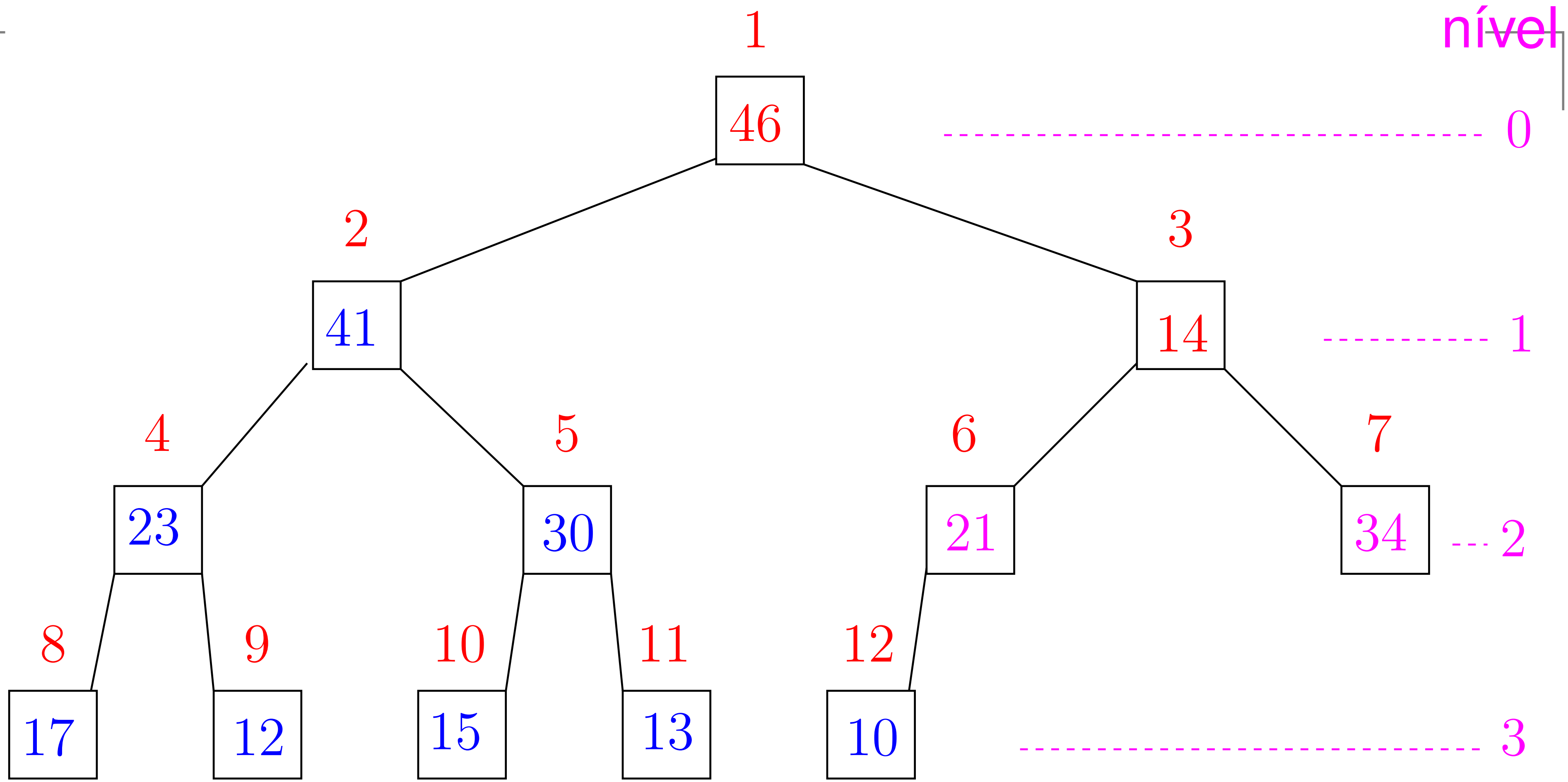
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	30	21	34	17	12	15	13	10

Construção de um max-heap



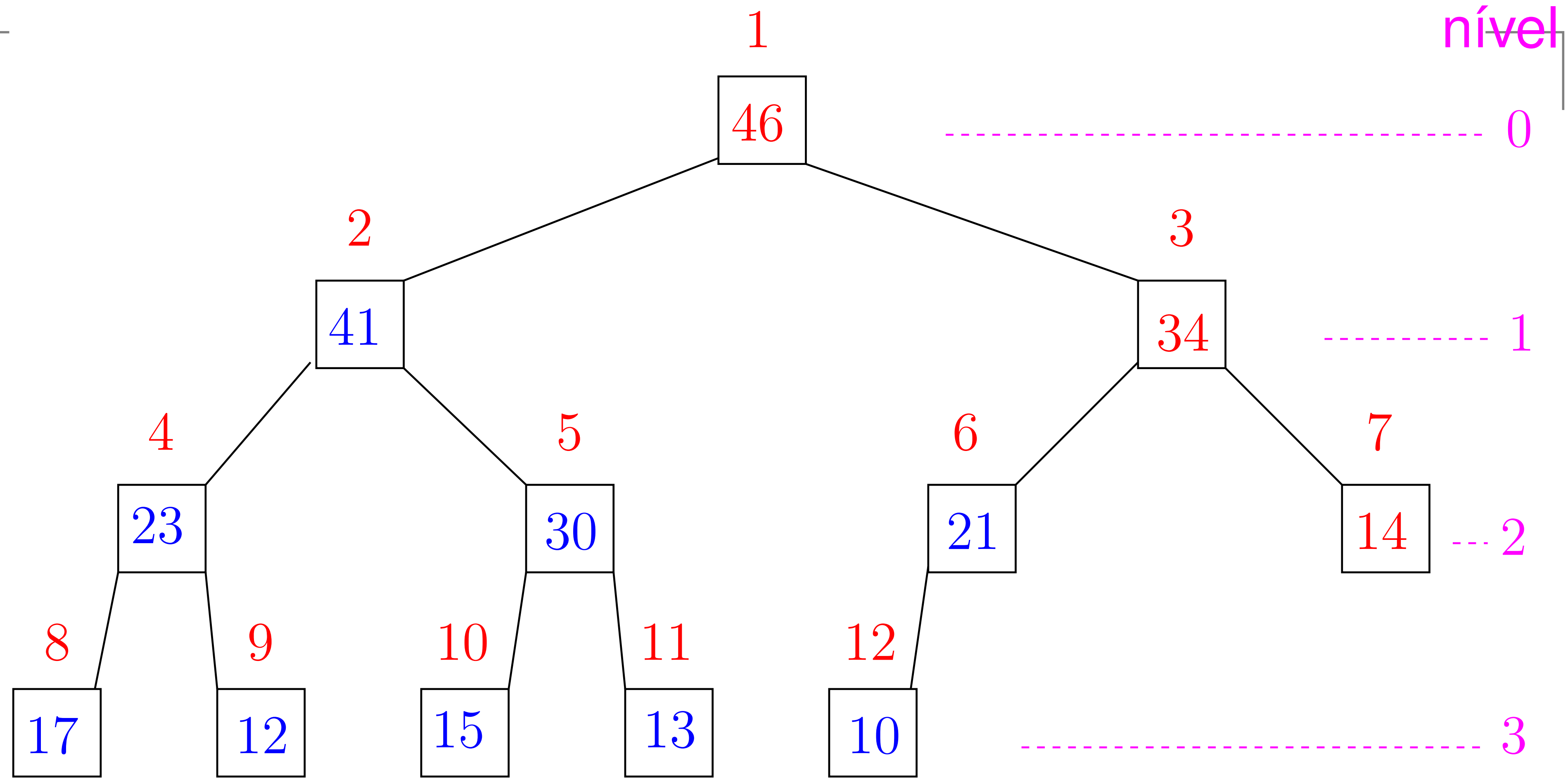
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	30	21	34	17	12	15	13	10

Construção de um max-heap



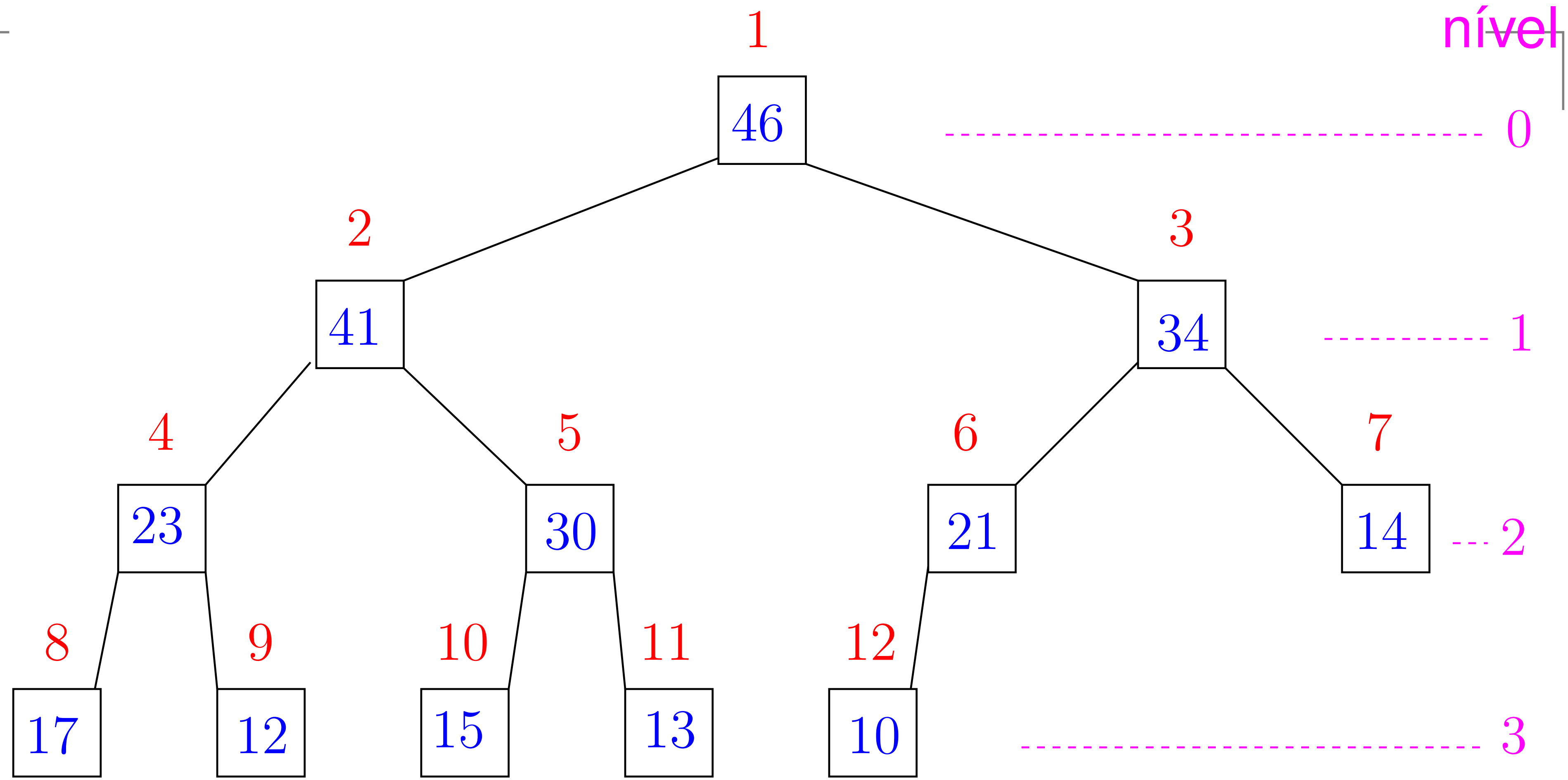
1	2	3	4	5	6	7	8	9	10	11	12
46	41	14	23	30	21	34	17	12	15	13	10

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 um vetor $A[1\,.\,.\,n]$ e rearranja A para que seja max-heap.

BUILD-MAX-HEAP (A, n)
2 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
3 MAX-HEAPIFY (A, n, i)

Relação invariante:

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

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

Construção de um max-heap

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

BUILD-MAX-HEAP (A, n)
2 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
3 MAX-HEAPIFY (A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i+1, \ldots, n$ são raízes de max-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: O consumo de MAX-HEAPIFY (A, n, i) é proporcional a $h = \lfloor \lg \frac{n}{i} \rfloor$. Logo,

$$\begin{aligned}
T(n) &\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil h \\
&\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h}\, h \quad \text{(Exercício 12.C)} \\
&\leq n \left(\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{\lfloor \lg n \rfloor}{2^{\lfloor \lg n \rfloor}} \right) \\
&< n \, \frac{1/2}{(1-1/2)^2} \\
&= 2n.
\end{aligned}$$

$T(n)$ é $\mathrm{O}(n)$

Prova: O consumo de tempo de MAX-HEAPIFY (A, n, i) é a $\mathrm{O}(h) = \mathrm{O}(\lfloor \lg \frac{n}{i} \rfloor)$. Logo,

$$
\begin{aligned}
T(n) &= \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \mathrm{O}(h) \\
&= \mathrm{O}\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) \qquad \text{(Exercício 12.C)} \\
&= \mathrm{O}\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) \\
&= \mathrm{O}\left(n \frac{1/2}{(1 - 1/2)^2} \right) \\
&= \mathrm{O}(2n) = \mathrm{O}(n)
\end{aligned}
$$

Algumas séries

Para todo número real x, $|x| < 1$, temos que $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$.

Para todo número real x, $|x| < 1$, temos que

$$\sum_{i=1}^{\infty} i\,x^i = \frac{x}{(1-x)^2}$$

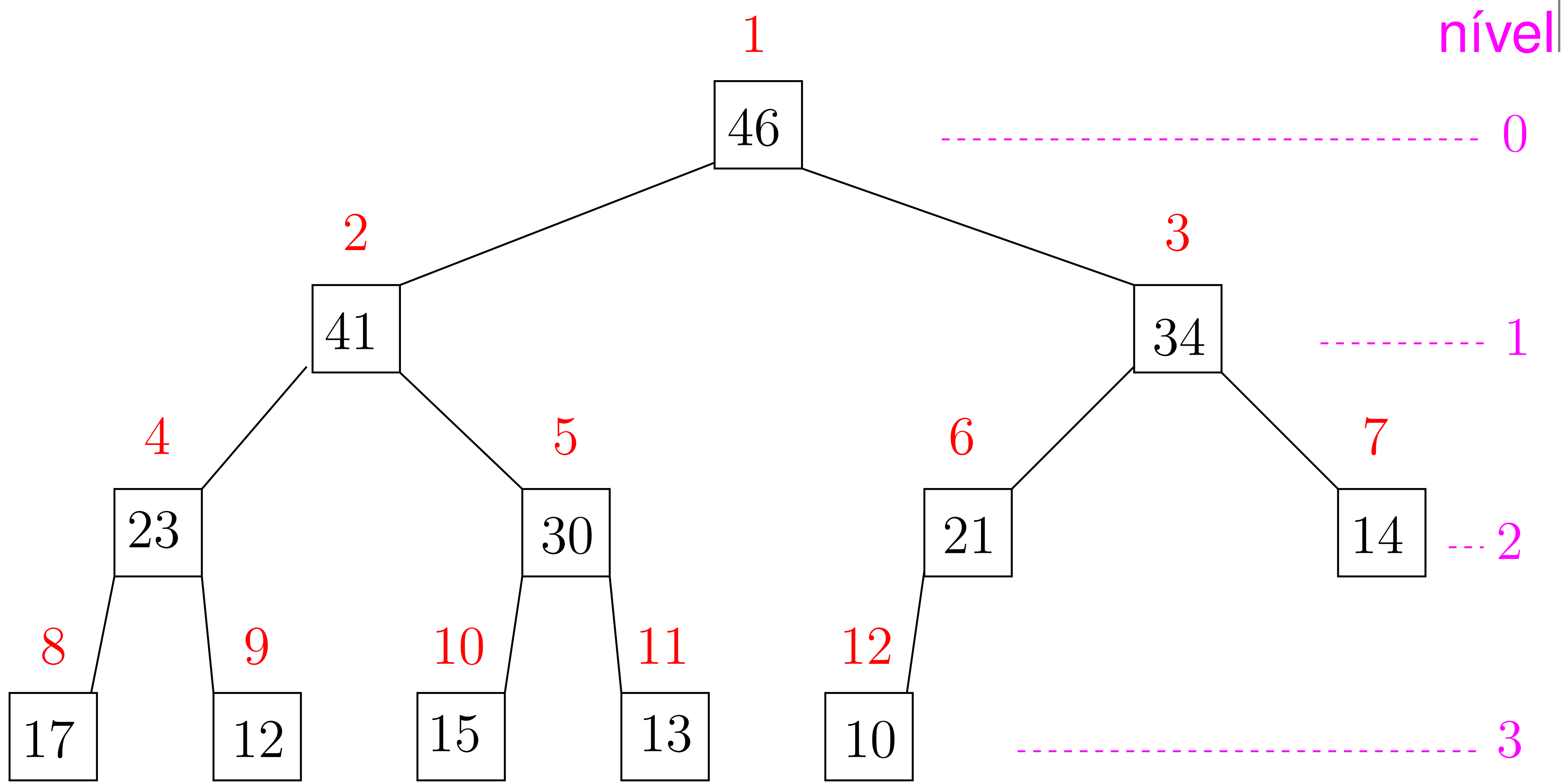
Prova:

$$\begin{aligned}
\sum_{i=1}^{\infty} i\,x^i &= \sum_{i=1}^{\infty} x^i + \sum_{i=2}^{\infty} x^i + \cdots + \sum_{i=k}^{\infty} x^i + \cdots \\
&= \frac{x}{1-x} + \frac{x^2}{1-x} + \cdots + \frac{x^k}{1-x} + \cdots \\
&= \frac{x}{1-x}(x^0 + x^1 + x^2 + \cdots + x^k + \cdots) = \frac{x}{(1-x)^2}.
\end{aligned}$$

Conclusão

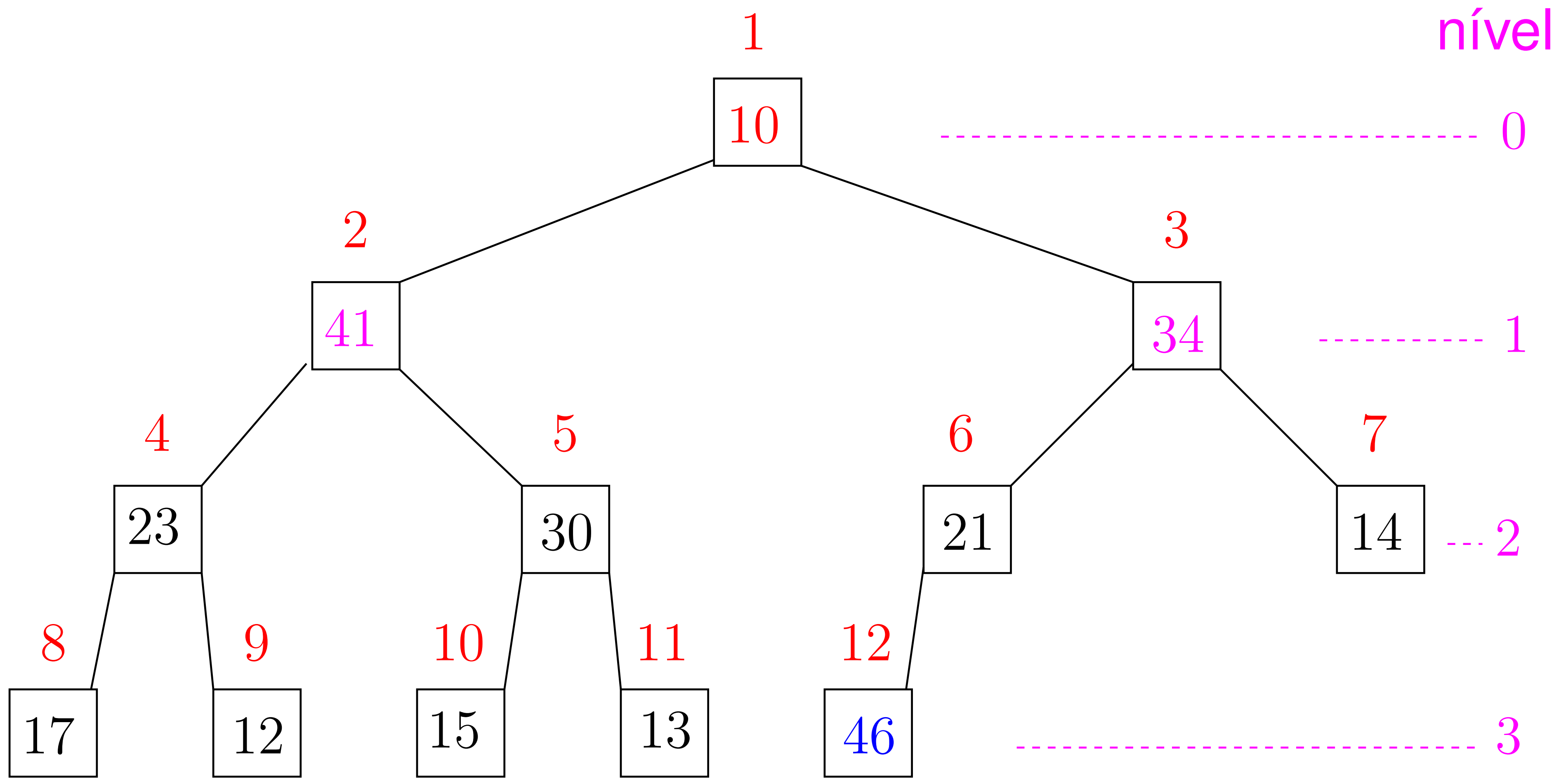
O consumo de tempo do algoritmo
BUILD-MAX-HEAP é $\Theta(n)$.

Heapsort



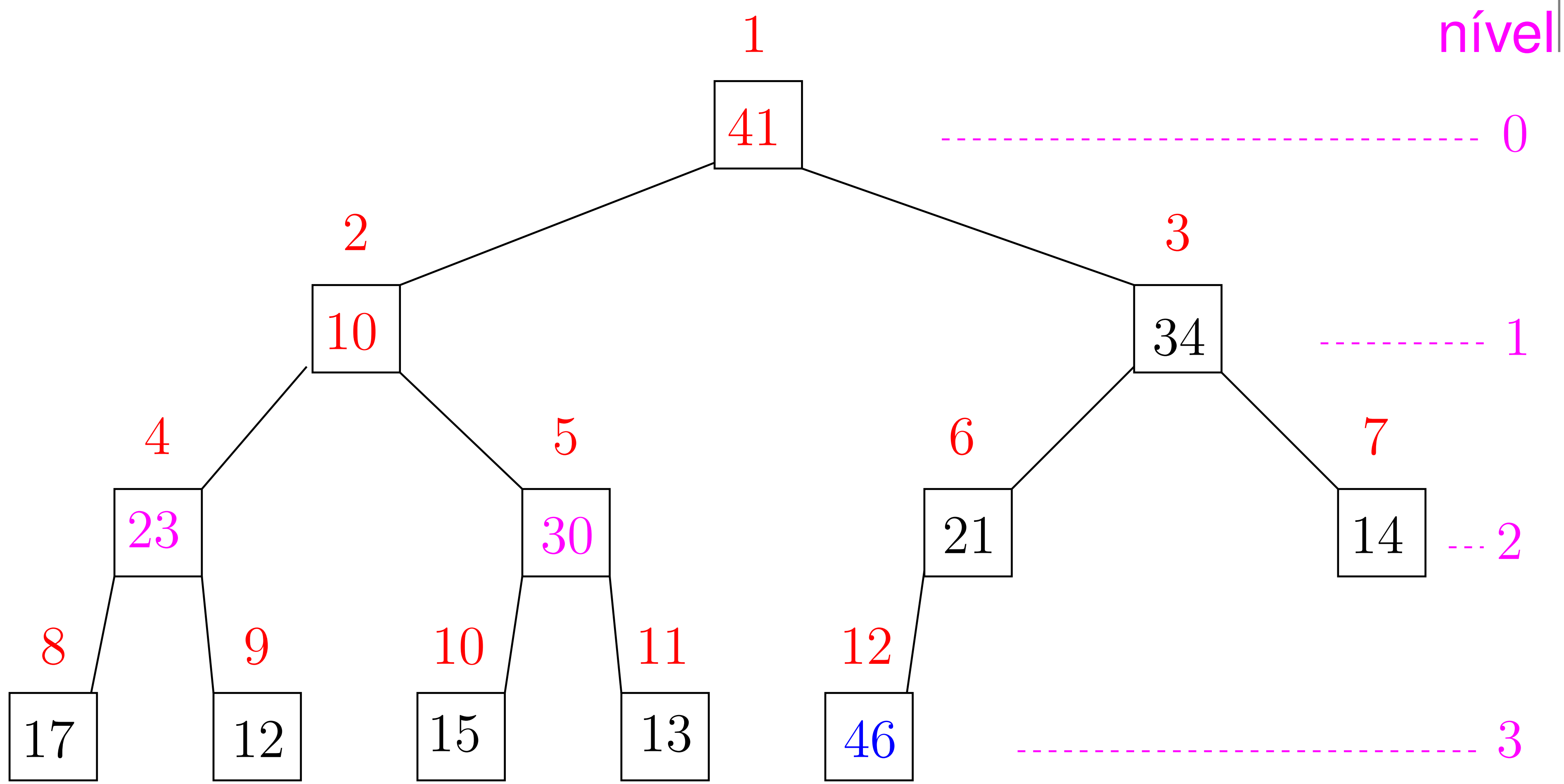
1	2	3	4	5	6	7	8	9	10	11	12
46	41	34	23	30	21	14	17	12	15	13	10

Heapsort



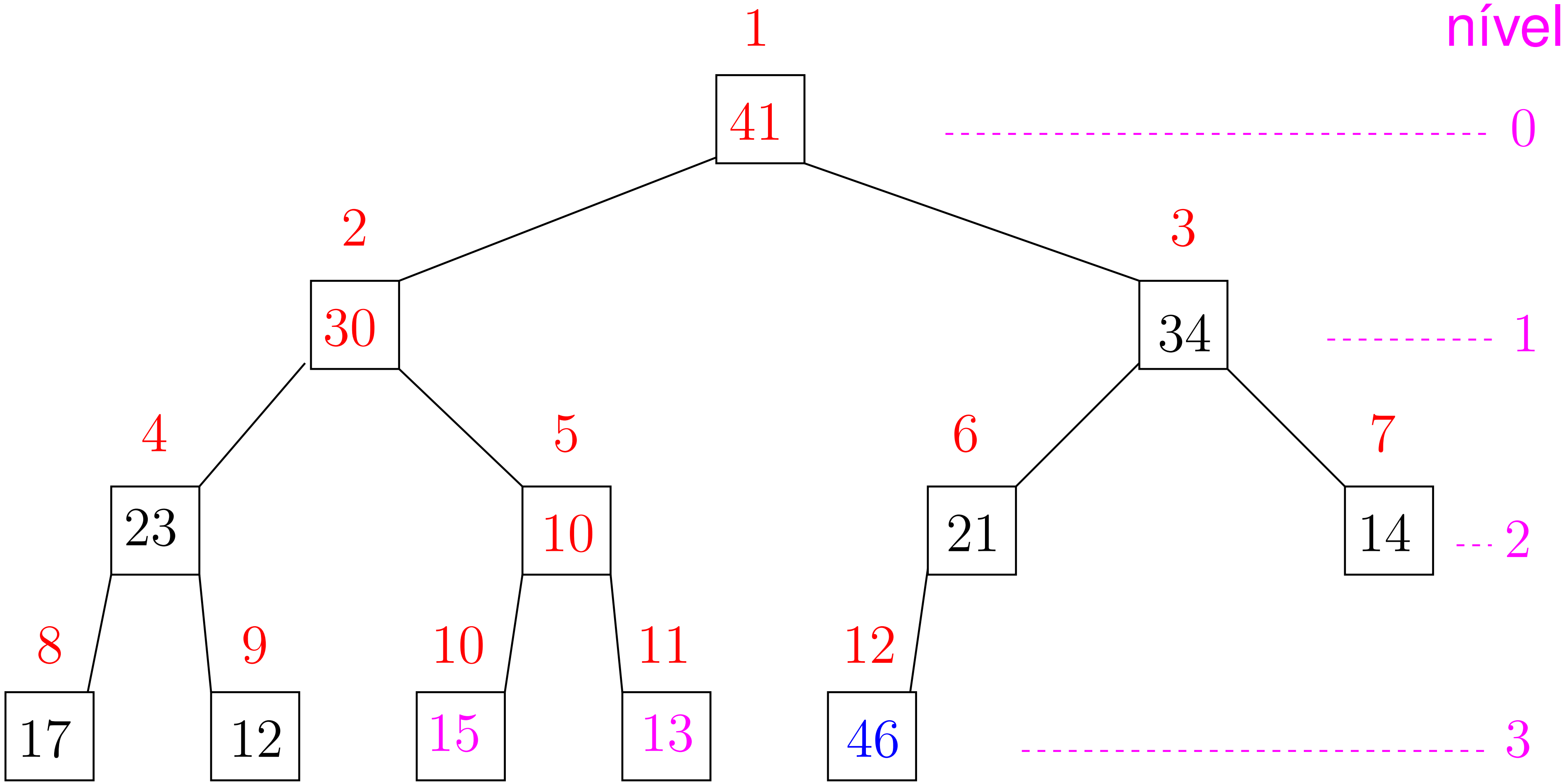
1	2	3	4	5	6	7	8	9	10	11	12
10	41	34	23	30	21	14	17	12	15	13	46

Heapsort



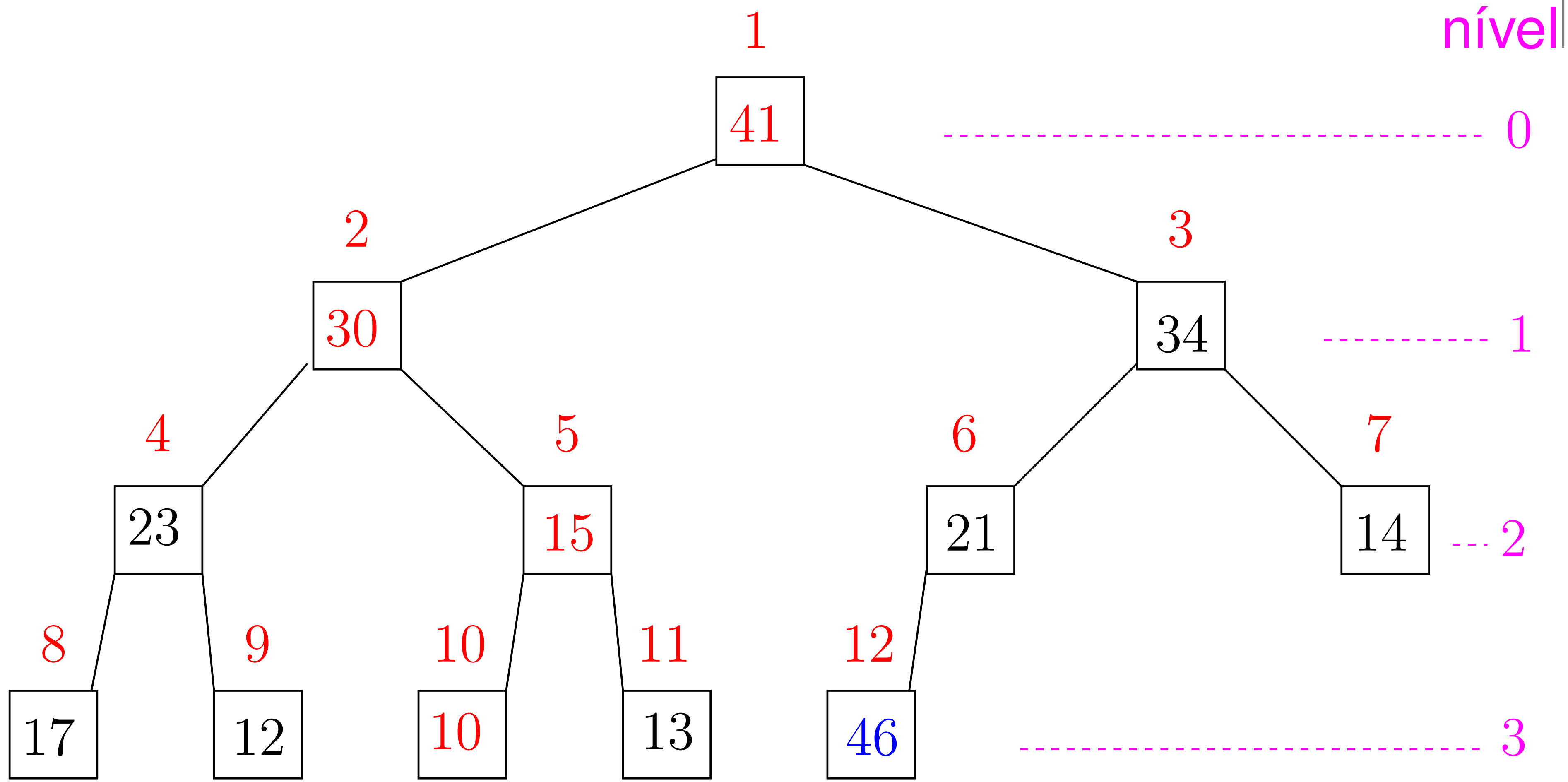
1	2	3	4	5	6	7	8	9	10	11	12
41	10	34	23	30	21	14	17	12	15	13	46

Heapsort



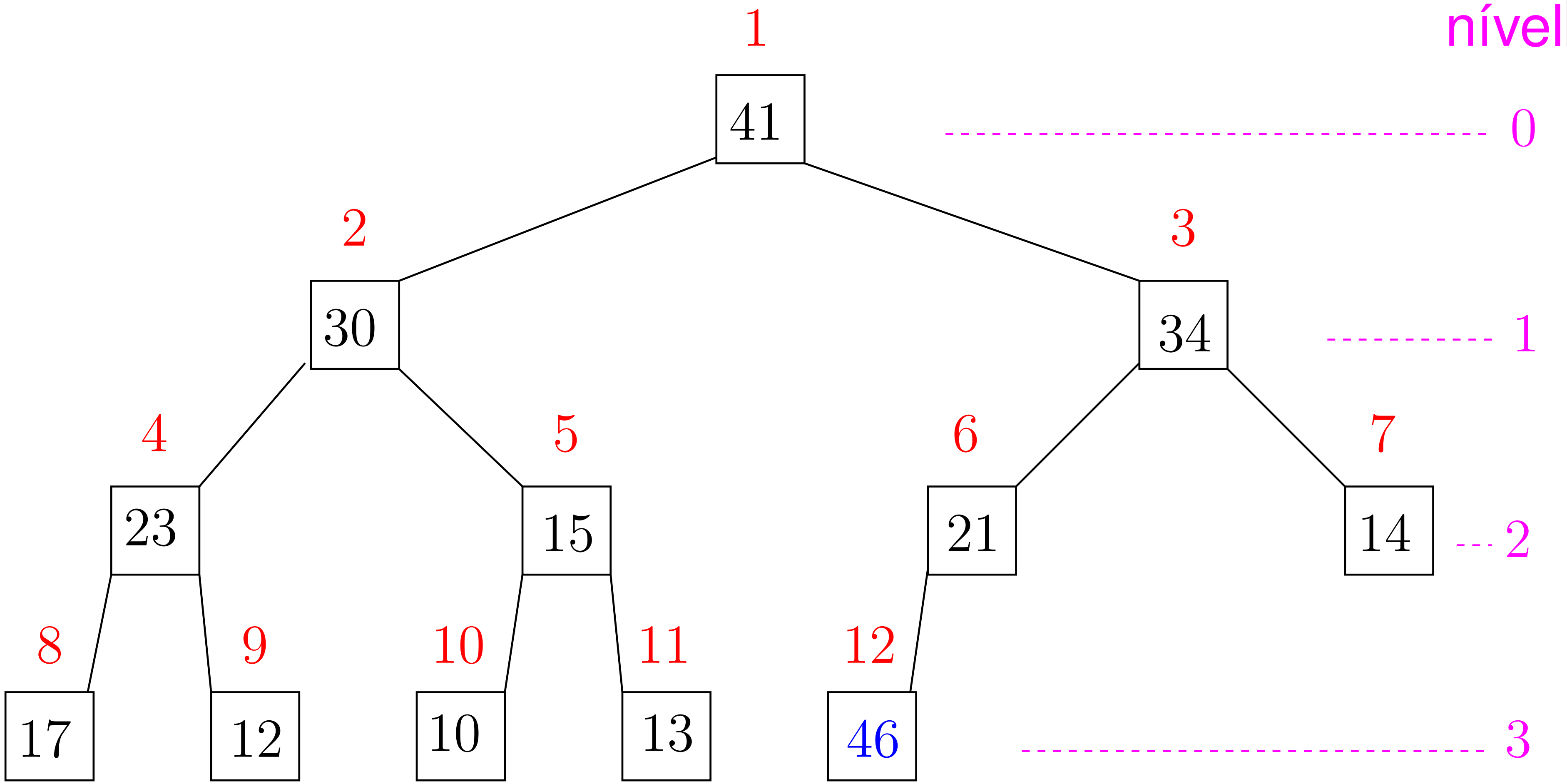
1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	10	21	14	17	12	15	13	46

Heapsort



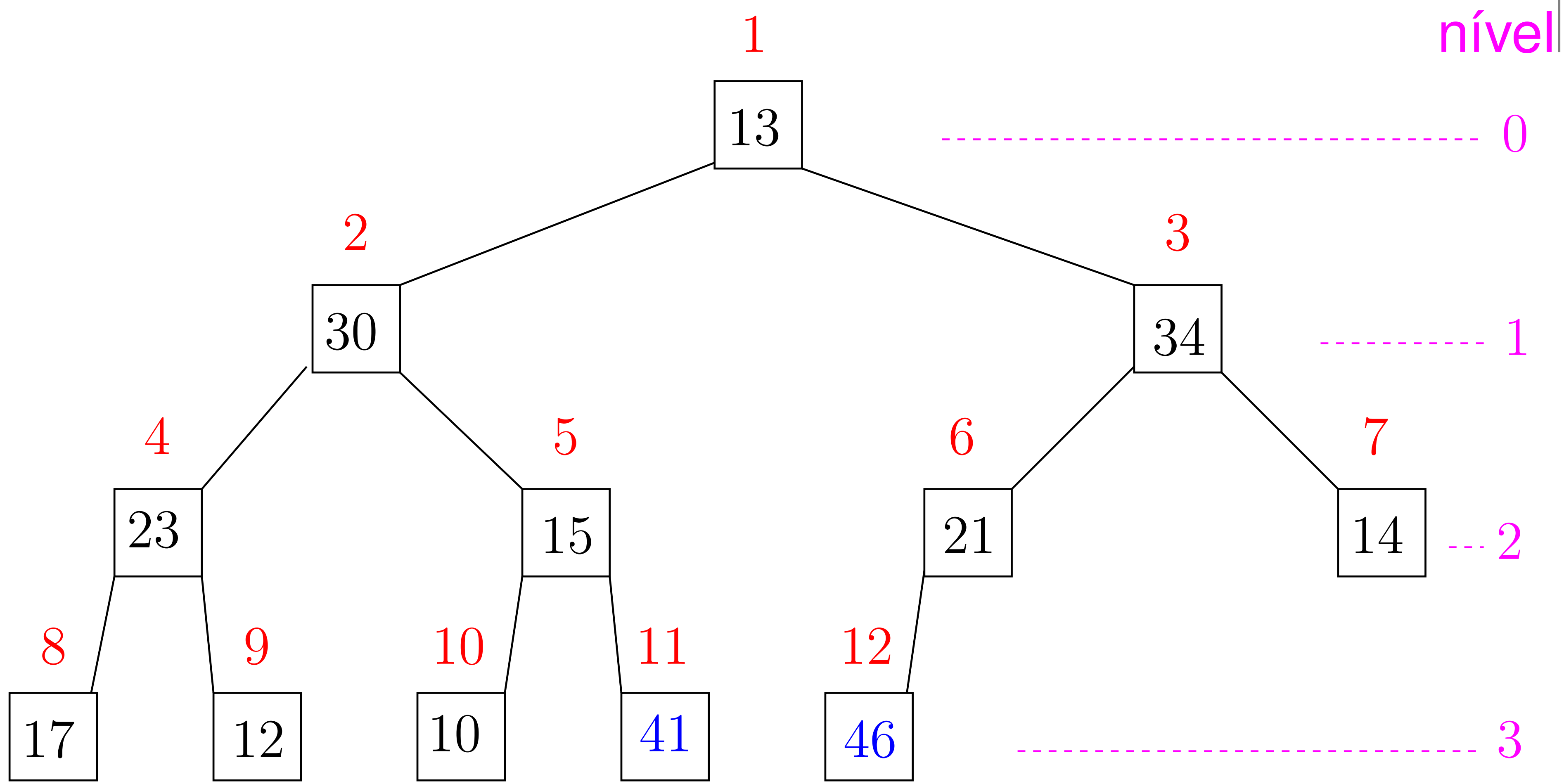
1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	15	21	14	17	12	10	13	46

Heapsort



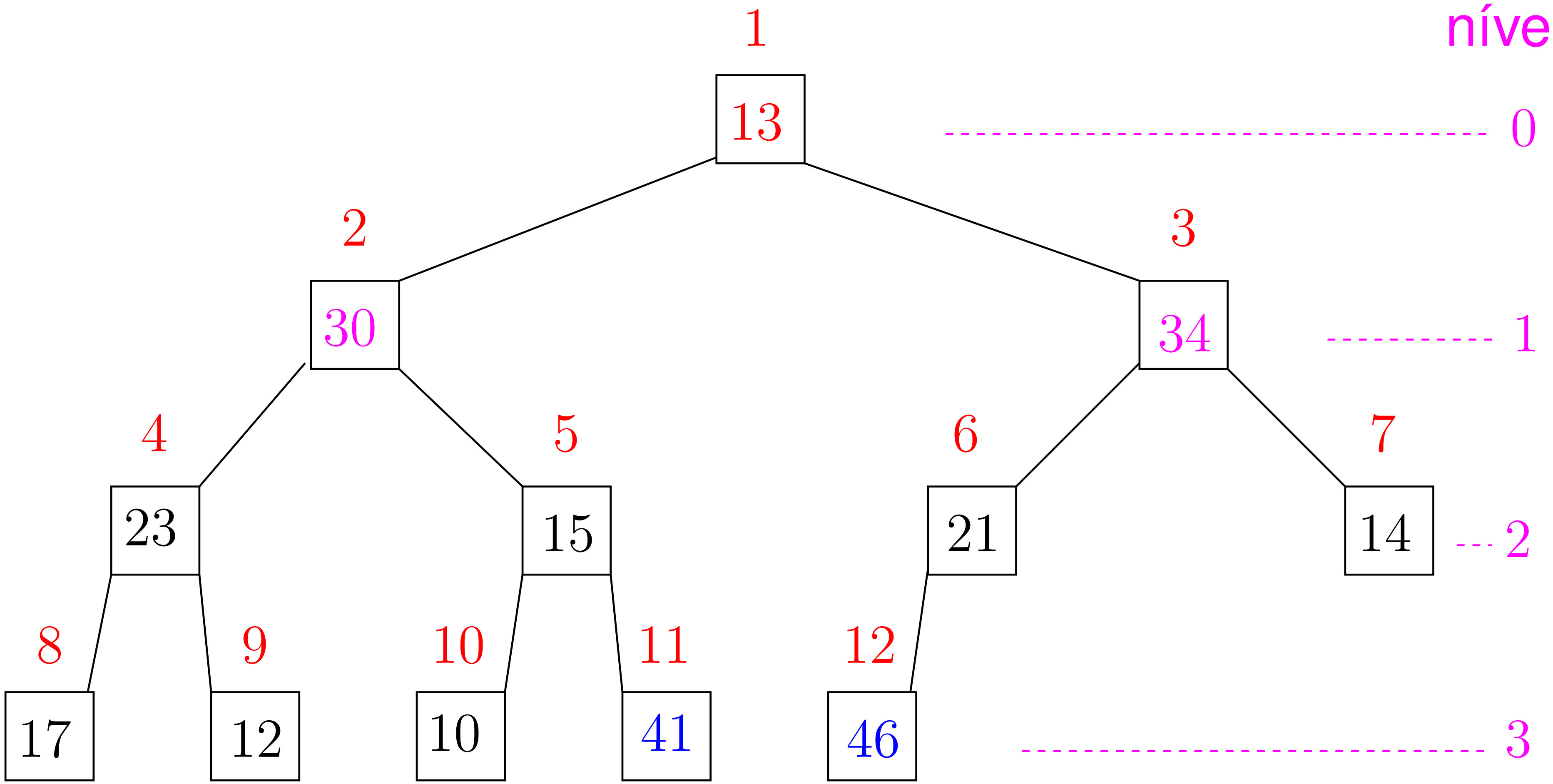
1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	15	21	14	17	12	10	13	46

Heapsort



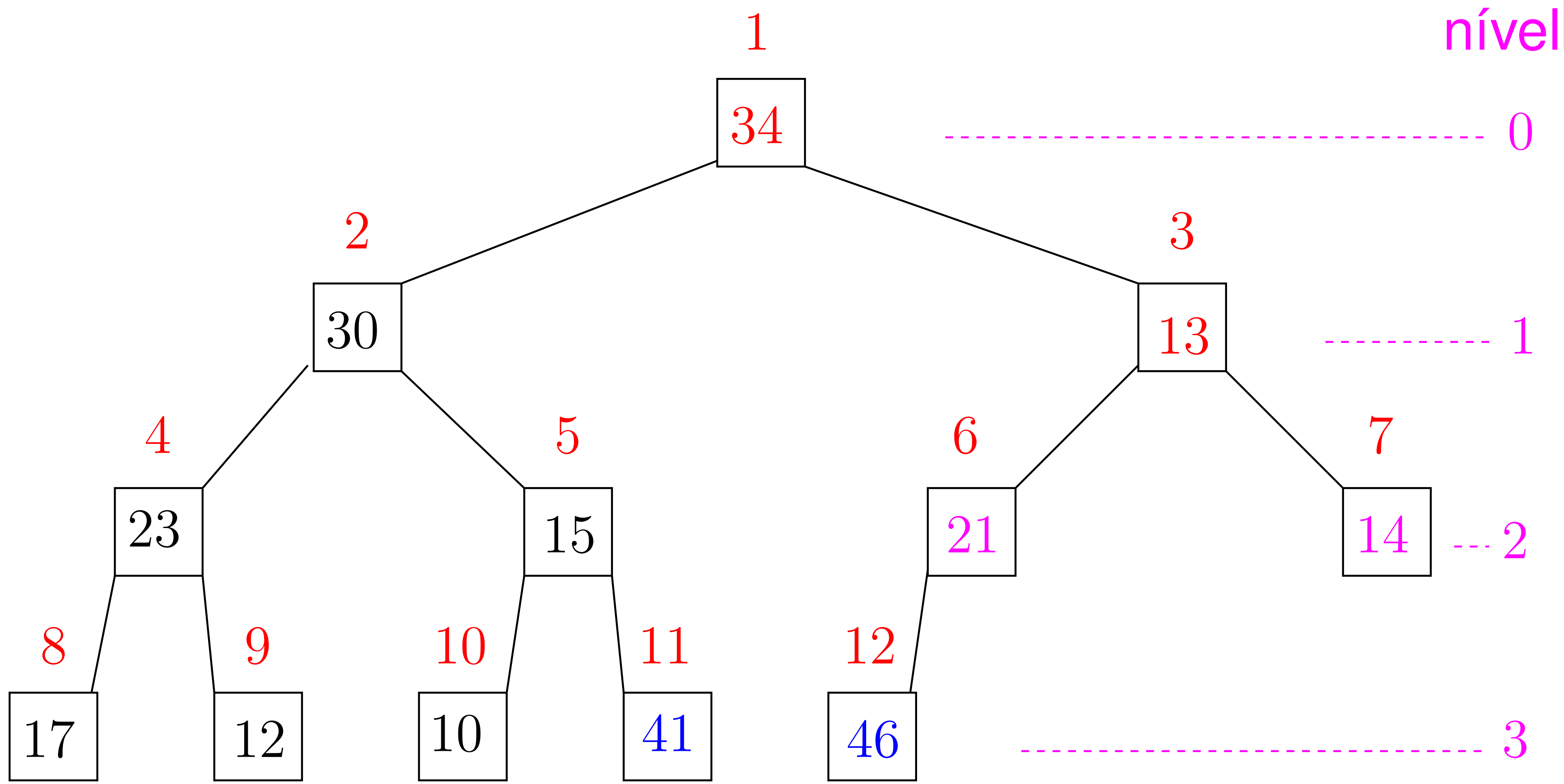
1	2	3	4	5	6	7	8	9	10	11	12
13	30	34	23	15	21	14	17	12	10	41	46

Heapsort



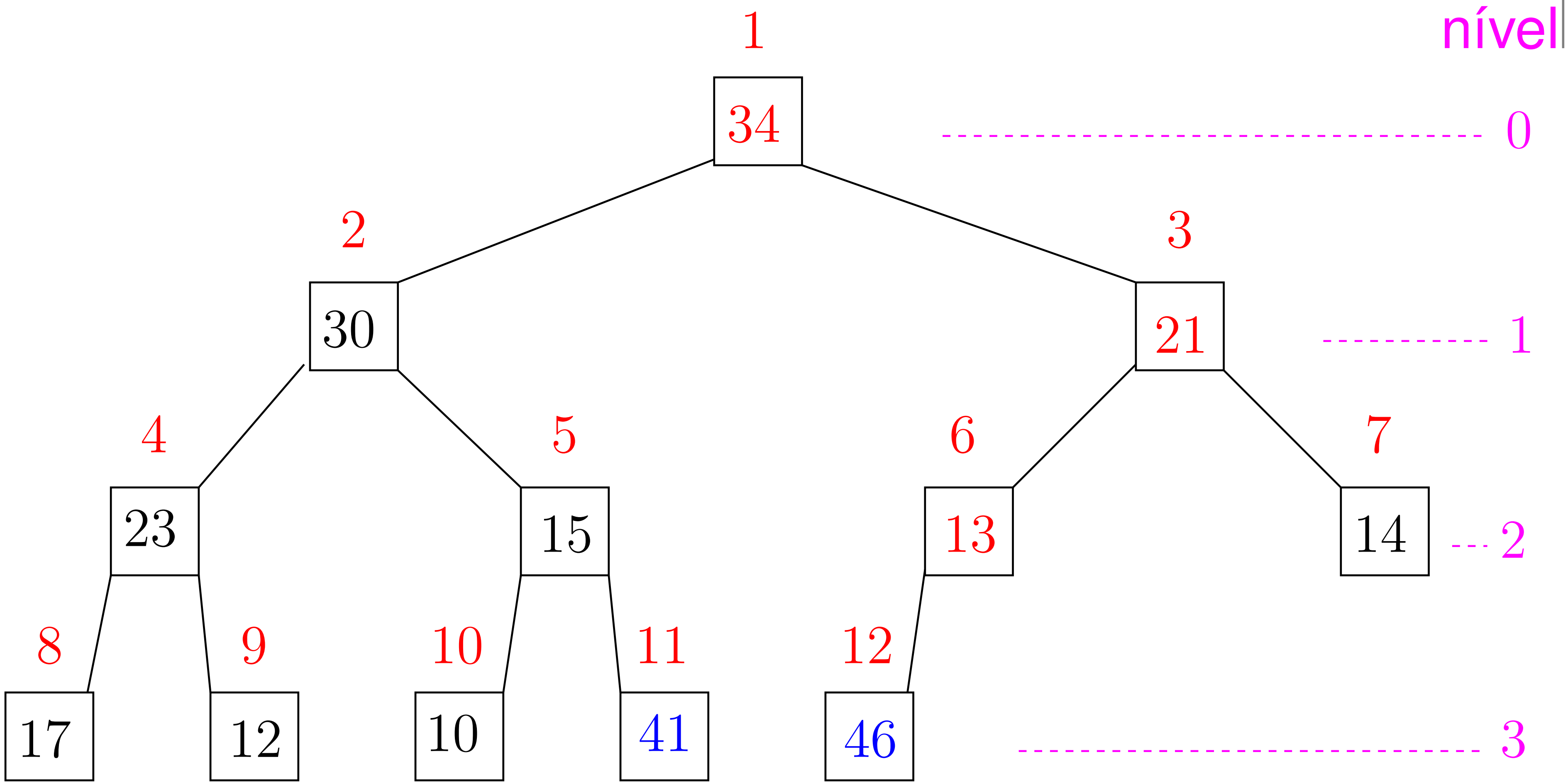
1	2	3	4	5	6	7	8	9	10	11	12
13	30	34	23	15	21	14	17	12	10	41	46

Heapsort



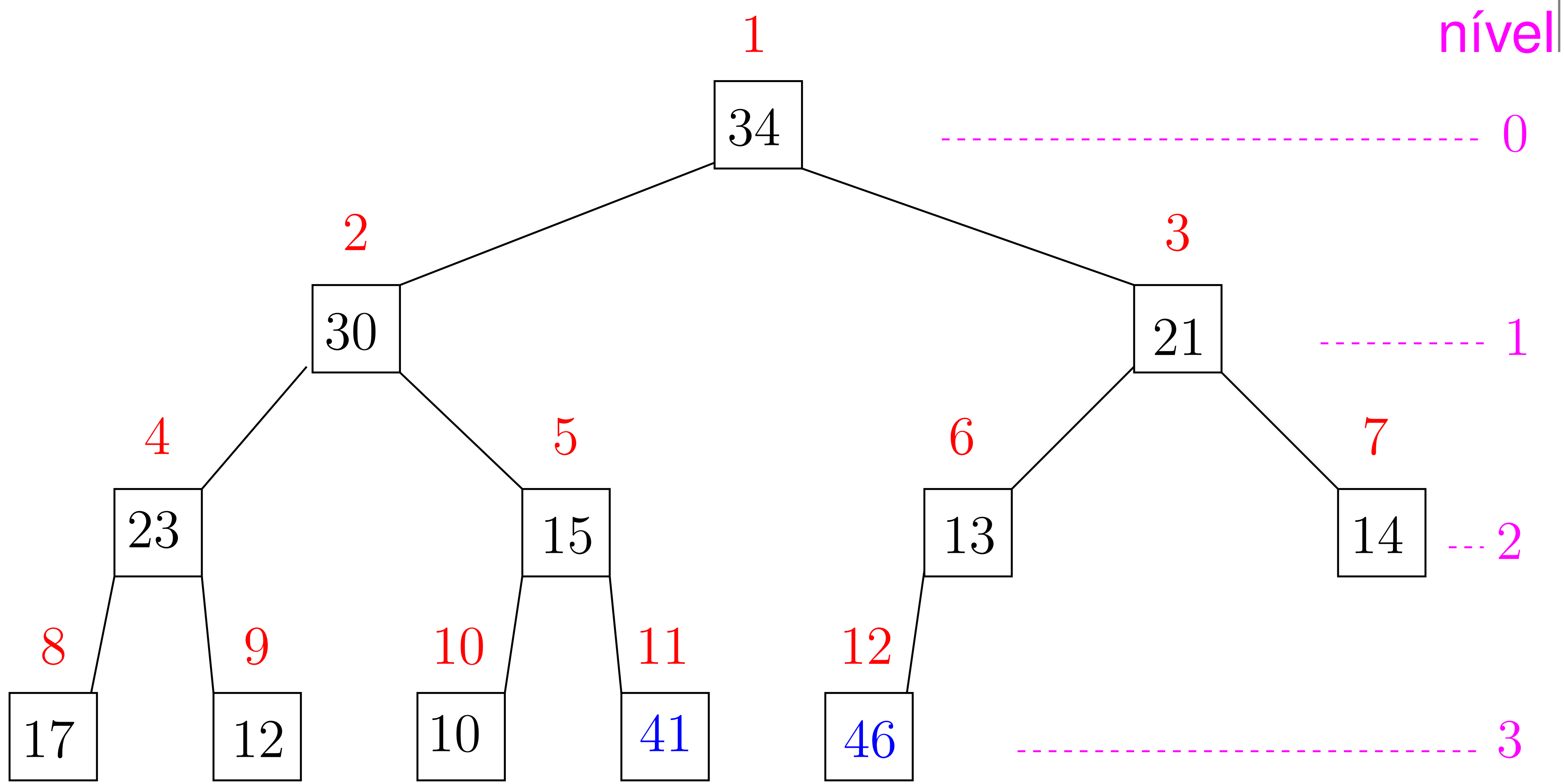
1	2	3	4	5	6	7	8	9	10	11	12
34	30	13	23	15	21	14	17	12	10	41	46

Heapsort



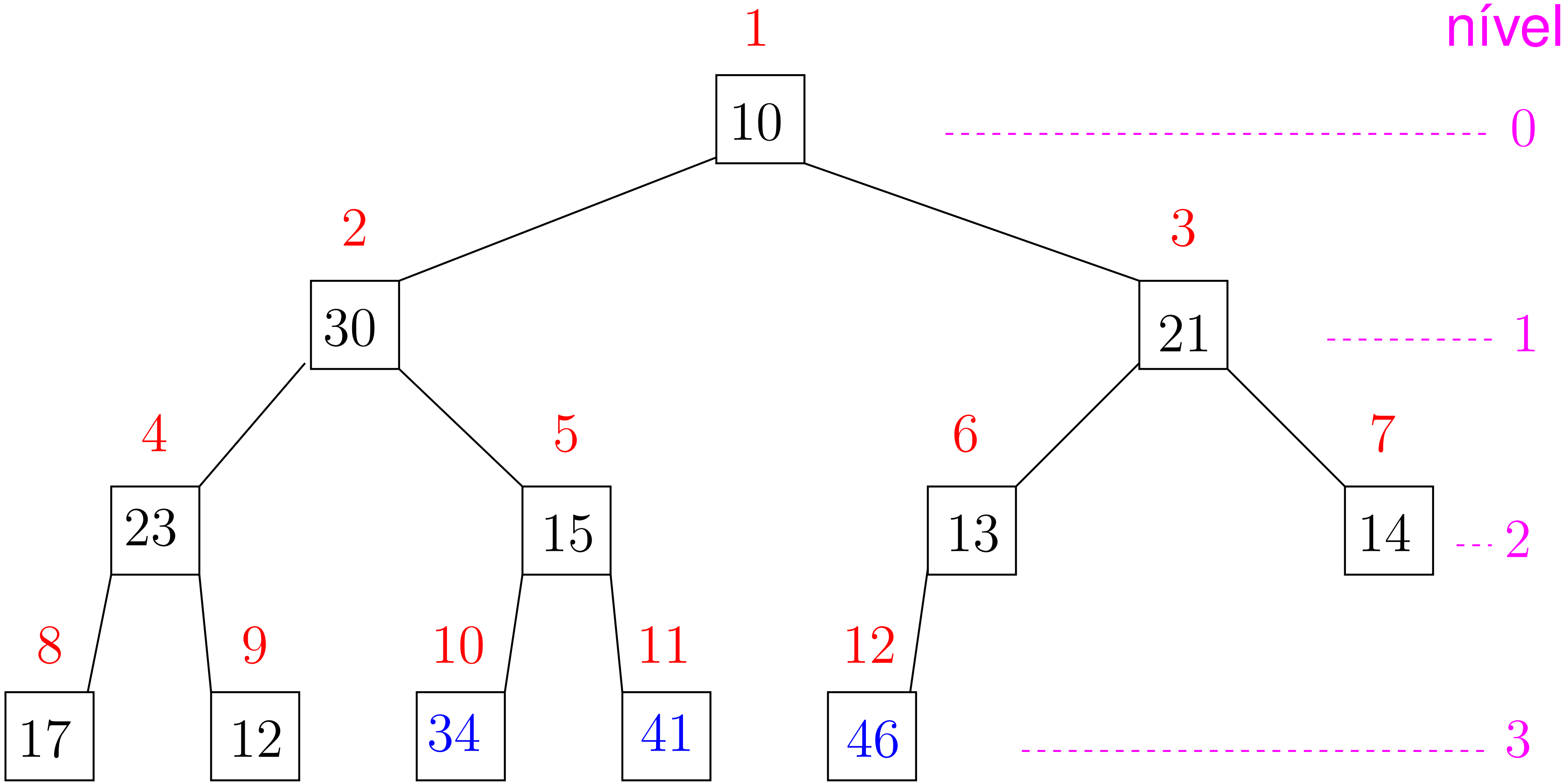
1	2	3	4	5	6	7	8	9	10	11	12
34	30	21	23	15	13	14	17	12	10	41	46

Heapsort



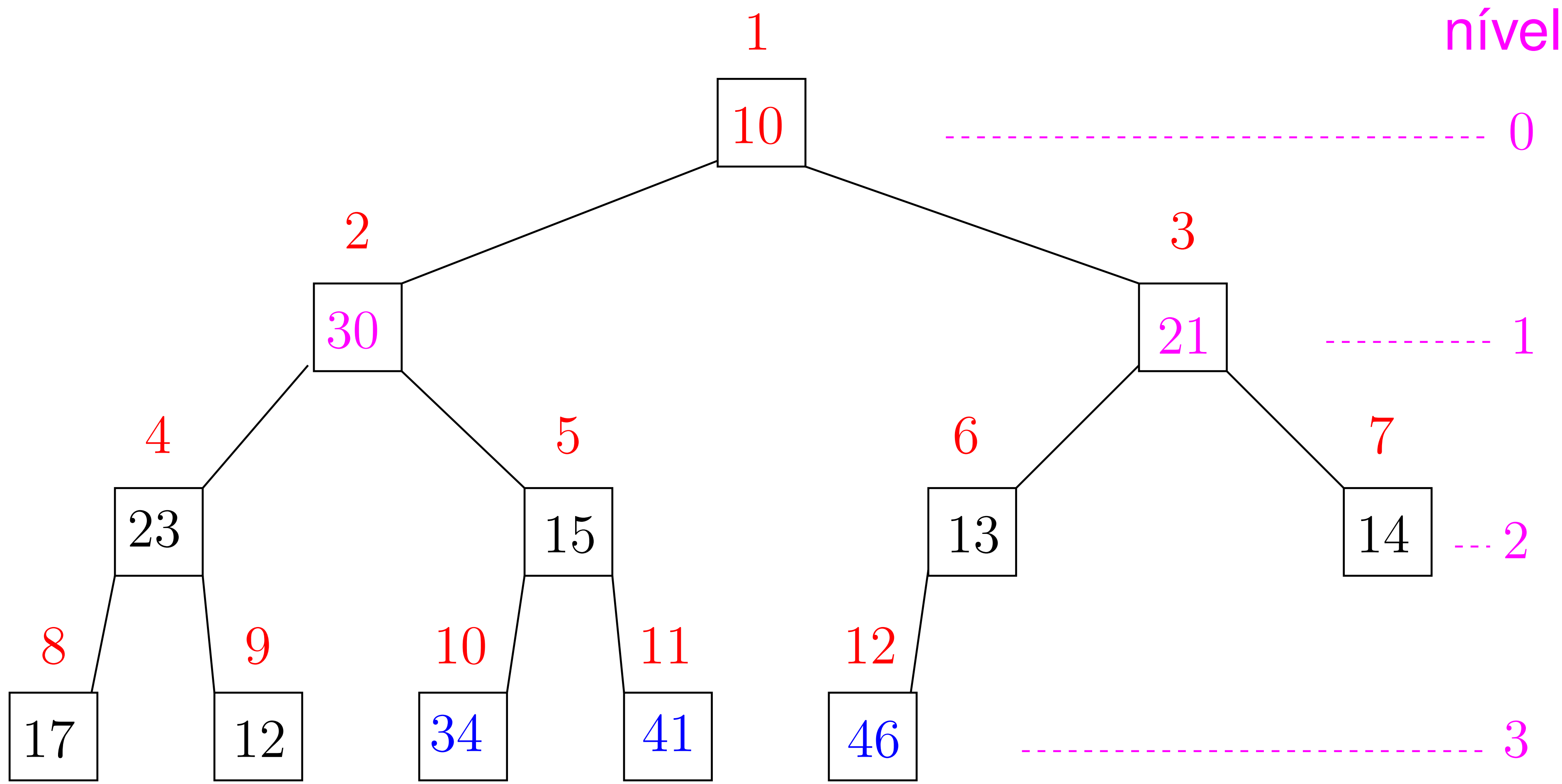
1	2	3	4	5	6	7	8	9	10	11	12
34	30	21	23	15	13	14	17	12	10	41	46

Heapsort



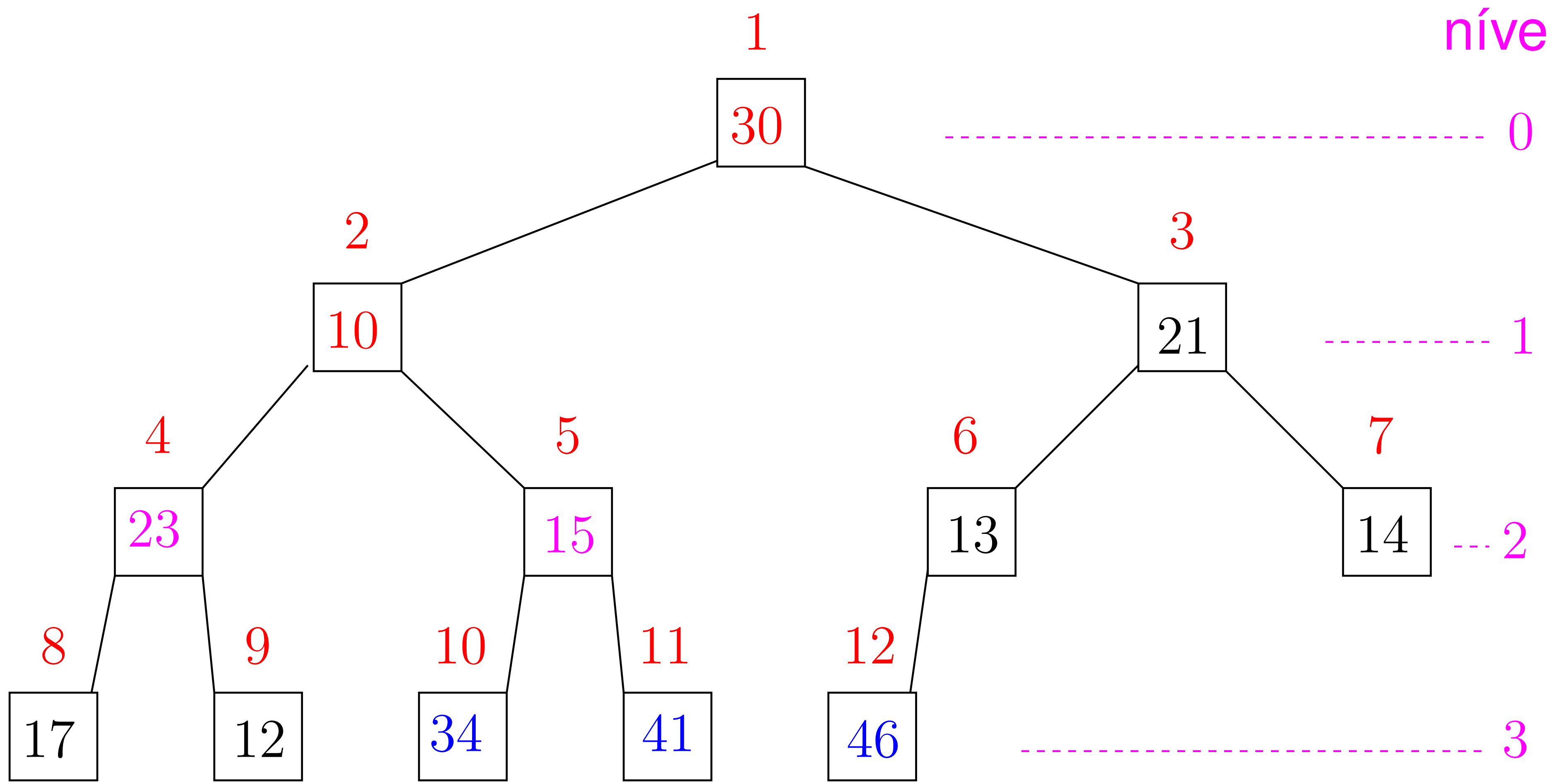
1	2	3	4	5	6	7	8	9	10	11	12
10	30	21	23	15	13	14	17	12	34	41	46

Heapsort



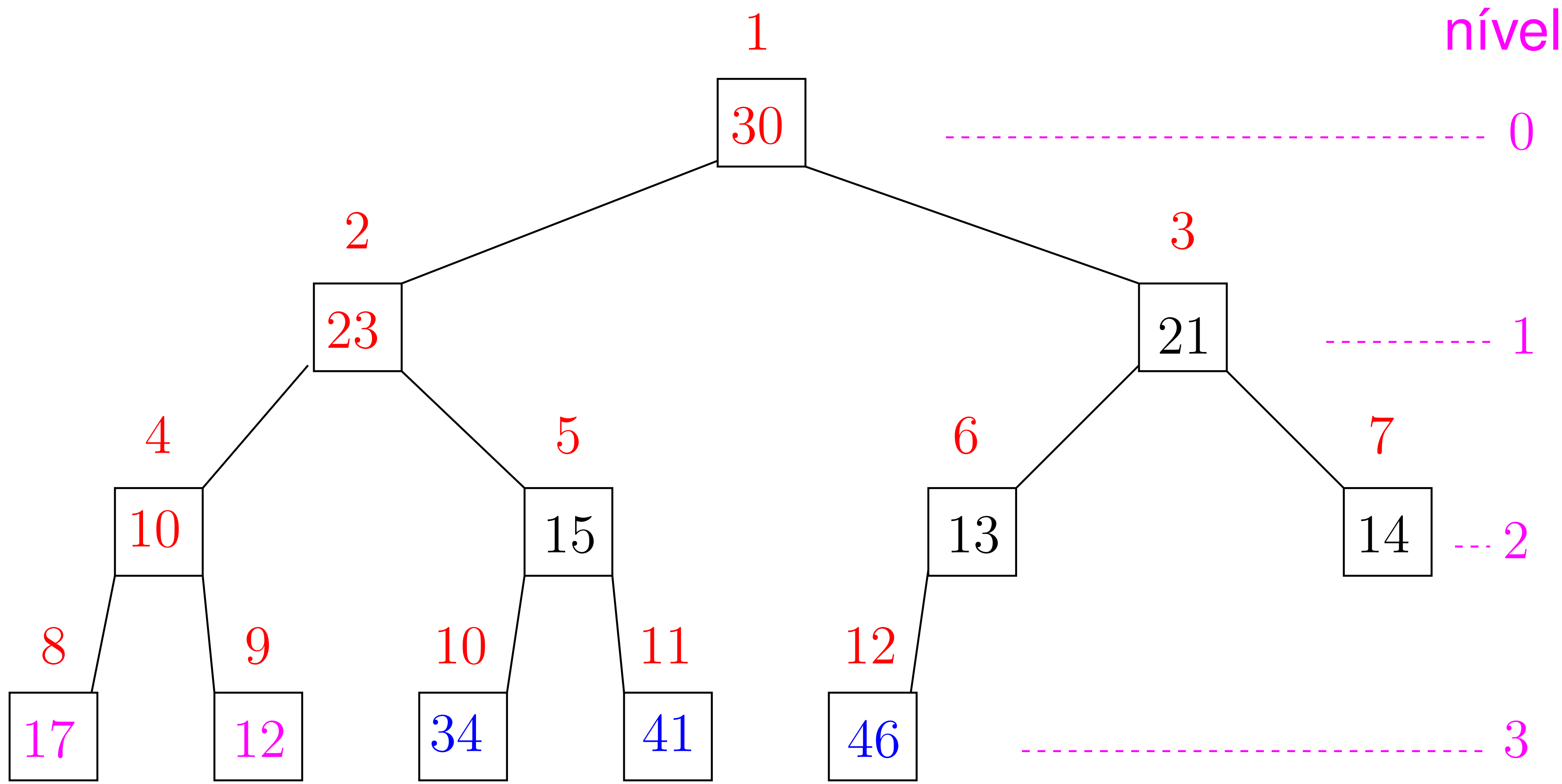
1	2	3	4	5	6	7	8	9	10	11	12
10	30	21	23	15	13	14	17	12	34	41	46

Heapsort



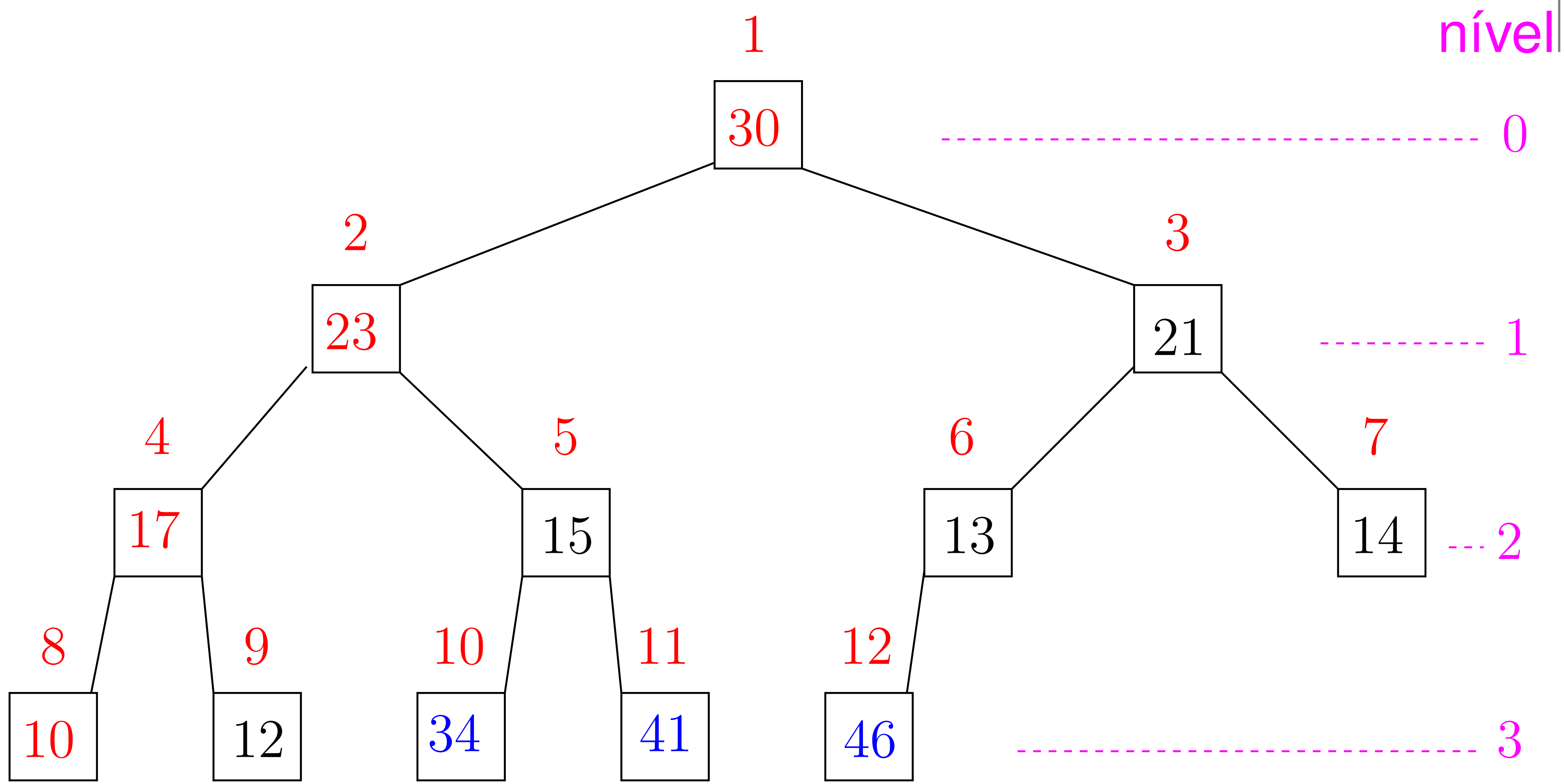
1	2	3	4	5	6	7	8	9	10	11	12
30	10	21	23	15	13	14	17	12	34	41	46

Heapsort



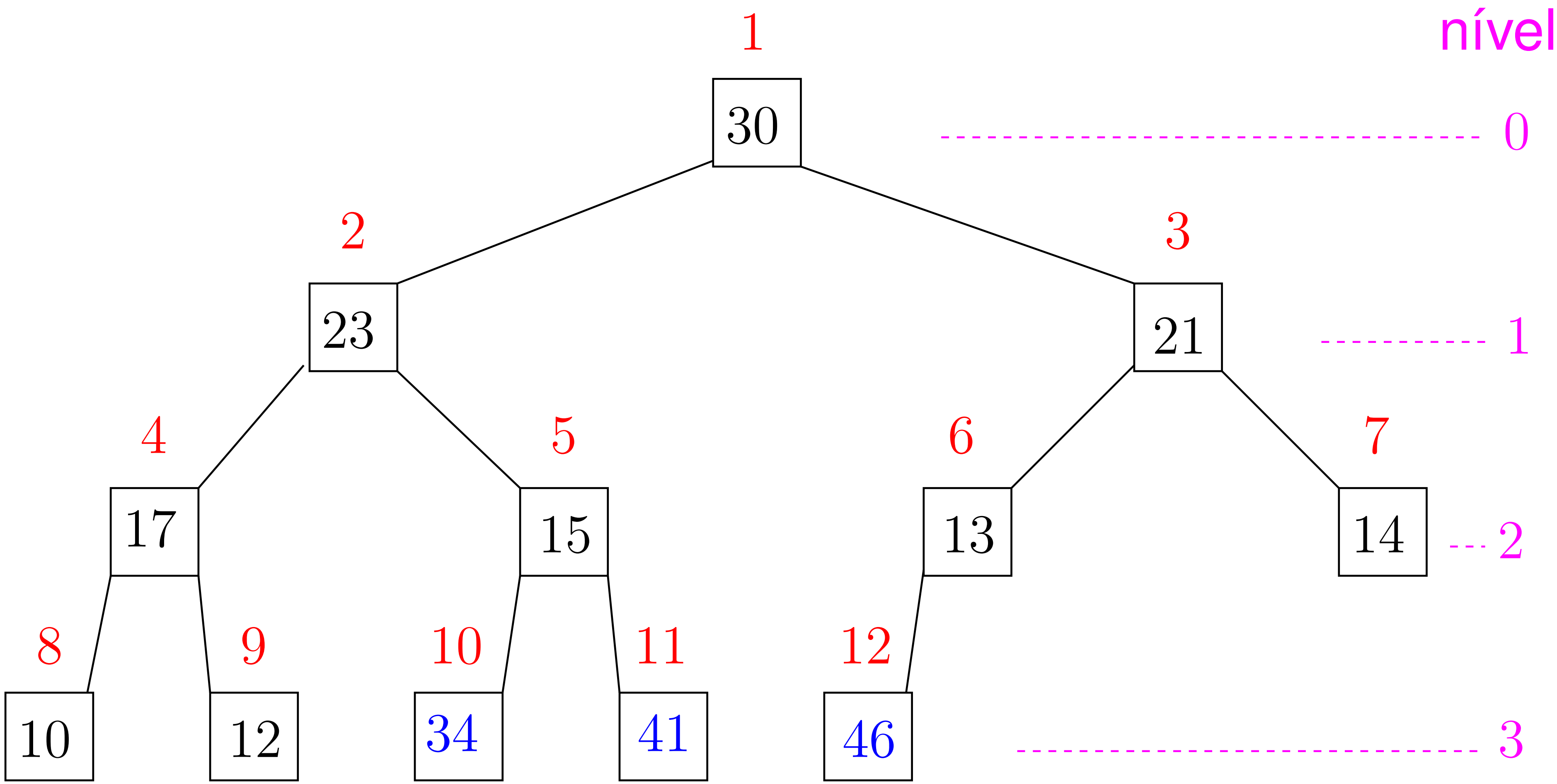
1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	10	15	13	14	17	12	34	41	46

Heapsort



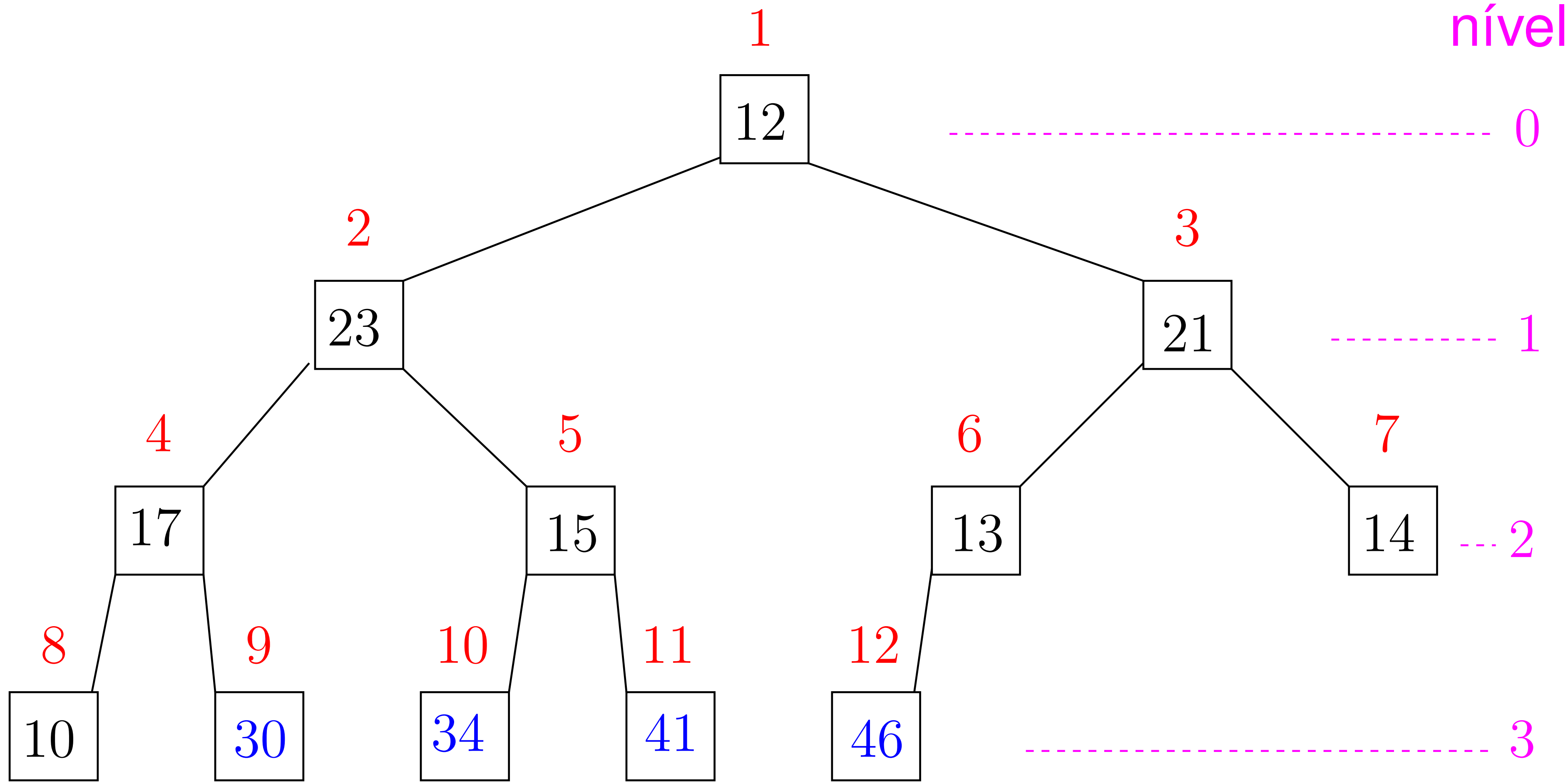
1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	17	15	13	14	10	12	34	41	46

Heapsort



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

Heapsort



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

Heapsort

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

HEAPSORT (A, n)
0 BUILD-MAX-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 MAX-HEAPIFY $(A, m, 1)$

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 max-heap.

Consumo de tempo

linha		todas as execuções da linha
0	$=$	$\Theta(n)$
1	$=$	$\Theta(1)$
2	$=$	$\Theta(n)$
3	$=$	$\Theta(n)$
4	$=$	$\Theta(n)$
6	$=$	$n\mathrm{O}(\lg n)$
total	$=$	$n\mathrm{O}(\lg n) + \Theta(4n+1) = \mathrm{O}(n \lg n)$

Conclusão

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

Um pouco de experimentação

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

Os códigos estão compilados com o gcc versão 2.7.2.1 e opção de compilação -O2.

Algoritmos implementados:

bub	`bubblesort`
bub2	`bubblesort_2`
shkr	`shakersort`
sele	ORDENA-POR-SELEÇÃO
ins	ORDENA-POR-INSERÇÃO
insS	`inserção Sedgewick`
insB	`inserção binária`
shell	`shellsort`

Crescente

n	bub	bub2	shkr	sele	ins	insS	insB	shell
256	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
512	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1024	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00
2048	0.04	0.00	0.00	0.04	0.00	0.00	0.00	0.00
4096	0.18	0.00	0.00	0.14	0.00	0.00	0.00	0.00
8192	0.90	0.00	0.00	0.71	0.00	0.00	0.01	0.00
16384	3.81	0.00	0.00	3.04	0.00	0.01	0.00	0.00
32768	15.48	0.00	0.00	12.32	0.00	0.00	0.01	0.02
65536	62.43	0.01	0.00	49.44	0.00	0.01	0.04	0.05
131072	266.90	0.01	0.00	214.86	0.01	0.01	0.09	0.11

Decrescente

n	bub	bub2	shkr	sele	ins	insS	insB
256	0.00	0.01	0.00	0.00	0.00	0.00	0.00
512	0.00	0.01	0.00	0.01	0.00	0.00	0.00
1024	0.01	0.02	0.02	0.01	0.00	0.01	0.01
2048	0.06	0.08	0.07	0.03	0.03	0.02	0.03
4096	0.28	0.29	0.28	0.15	0.11	0.07	0.11
8192	1.30	1.25	1.20	0.71	0.75	0.56	0.78
16384	5.41	5.15	4.98	3.01	3.45	2.57	3.59
32768	23.59	22.19	20.56	13.28	14.56	11.21	14.56
65536	91.07	85.88	84.66	51.16	67.98	55.89	59.67
131072	361.67	339.09	341.04	204.23	244.55	207.27	222.73

Aleatório: média de 10

n	bub	bub2	shkr	sele	ins	insS	insB
256	0.00	0.00	0.00	0.00	0.00	0.00	0.00
512	0.01	0.01	0.01	0.00	0.00	0.00	0.00
1024	0.03	0.02	0.02	0.01	0.00	0.01	0.00
2048	0.09	0.11	0.07	0.03	0.02	0.01	0.01
4096	0.37	0.41	0.27	0.14	0.06	0.04	0.06
8192	1.70	1.77	1.12	0.71	0.29	0.22	0.30
16384	7.08	7.33	4.75	3.05	1.55	1.25	1.60
32768	28.54	29.32	19.30	12.33	6.88	5.61	7.08
65536	113.55	116.95	77.64	49.17	28.37	23.17	28.83
131072	493.08	506.67	323.34	224.09	121.01	105.44	115.89

Mais experimentação

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

Os códigos estão compilados com o gcc versão 2.7.2.1 e opção de compilação -O2.

Algoritmos implementados:

`shell`	`shellsort` origina
`merge_r`	MERGE-SORT recursivo
`merge_i`	MERGE-SORT iterativo
`heap`	HEAPSORT

Crescente

n	shell	merge_r	merge_i	heap
4096	0.01	0.00	0.01	0.00
8192	0.00	0.01	0.01	0.01
16384	0.01	0.02	0.02	0.01
32768	0.02	0.05	0.04	0.03
65536	0.05	0.10	0.10	0.07
131072	0.12	0.23	0.28	0.16

Decrescente

n	shell	merge_r	merge_i	heap
4096	0.00	0.01	0.00	0.00
8192	0.00	0.01	0.01	0.01
16384	0.02	0.02	0.02	0.01
32768	0.03	0.04	0.04	0.03
65536	0.07	0.09	0.09	0.07
131072	0.15	0.21	0.28	0.16

Aleatório: média de 10

n	shell	merge_r	merge_i	heap
4096	0.01	0.01	0.00	0.01
8192	0.01	0.01	0.01	0.01
16384	0.03	0.02	0.02	0.02
32768	0.07	0.05	0.05	0.04
65536	0.16	0.11	0.11	0.10
131072	0.38	0.25	0.33	0.23
262144	0.92	0.54	0.73	0.54
524288	2.13	1.18	1.55	1.31
1048576	4.97	2.52	3.32	3.06
2097152	11.38	5.42	6.96	7.07
4194304	26.50	11.40	14.46	15.84
8388608	61.68	24.35	29.76	35.85

Exercícios

Exercício 12.A
A altura de i em $A[1 \mathrel{.\,.} m]$ é o comprimento da mais longa seqüência da forma

$$\langle \text{filho}(i), \text{filho}(\text{filho}(i)), \text{filho}(\text{filho}(\text{filho}(i))), \ldots \rangle$$

onde $\text{filho}(i)$ vale $2i$ ou $2i+1$. Mostre que a altura de i é $\lfloor \lg \frac{m}{i} \rfloor$.
É verdade que $\lfloor \lg \frac{m}{i} \rfloor = \lfloor \lg m \rfloor - \lfloor \lg i \rfloor$?

Exercício 12.B
Mostre que um heap $A[1 \mathrel{.\,.} m]$ tem no máximo $\lceil m/2^{h+1} \rceil$ nós com altura h.

Exercício 12.C
Mostre que $\lceil m/2^{h+1} \rceil \leq m/2^h$ quando $h \leq \lfloor \lg m \rfloor$.

Exercício 12.D
Mostre que um heap $A[1 \mathrel{.\,.} m]$ tem no mínimo $\lfloor m/2^{h+1} \rfloor$ nós com altura h.

Exercício 12.E
Considere um heap $A[1 \mathrel{.\,.} m]$; a raiz do heap é o elemento de índice 1. Seja m' o número de elementos do "sub-heap esquerdo", cuja raiz é o elemento de índice 2. Seja m'' o número de elementos do "sub-heap direito", cuja raiz é o elemento de índice 3. Mostre que

$$m'' \leq m' < 2m/3 \,.$$

Mais execícios

Exercício 12.F
Mostre que a solução da recorrência

$$\begin{aligned} T(1) &= 1 \\ T(k) &\leq T(2k/3) + 5 \quad \text{para } k \geq 2 \end{aligned}$$

é $\mathrm{O}(\log k)$. Mais geral: mostre que se $T(k) = T(2k/3) + \mathrm{O}(1)$ então $T(k)$ é $\mathrm{O}(\log k)$. (Curiosidade: Essa é a recorrência do MAX-HEAPIFY (A, m, i) se interpretarmos k como sendo o número de nós na subárvore com raiz i).

Exercício 12.G
Escreva uma versão iterativa do algoritmo MAX-HEAPIFY. Faça uma análise do consumo de tempo do algoritmo.

Mais exercícios ainda

Exercício 12.H

Discuta a seguinte variante do algoritmo MAX-HEAPIFY:

M-H (A, m, i)
    $e \leftarrow 2i$
    $d \leftarrow 2i + 1$
    se $e \leq m$ e $A[e] > A[i]$
        então $A[i] \leftrightarrow A[e]$
            M-H (A, m, e)
    se $d \leq m$ e $A[d] > A[i]$
        então $A[i] \leftrightarrow A[d]$
            M-H (A, m, d)