Análise de Algoritmos

Slides de Paulo Feofiloff

[com erros do coelho e agora também da cris]

Heap

Um vetor $A[1 \ldots m]$ é um (max-)heap se

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

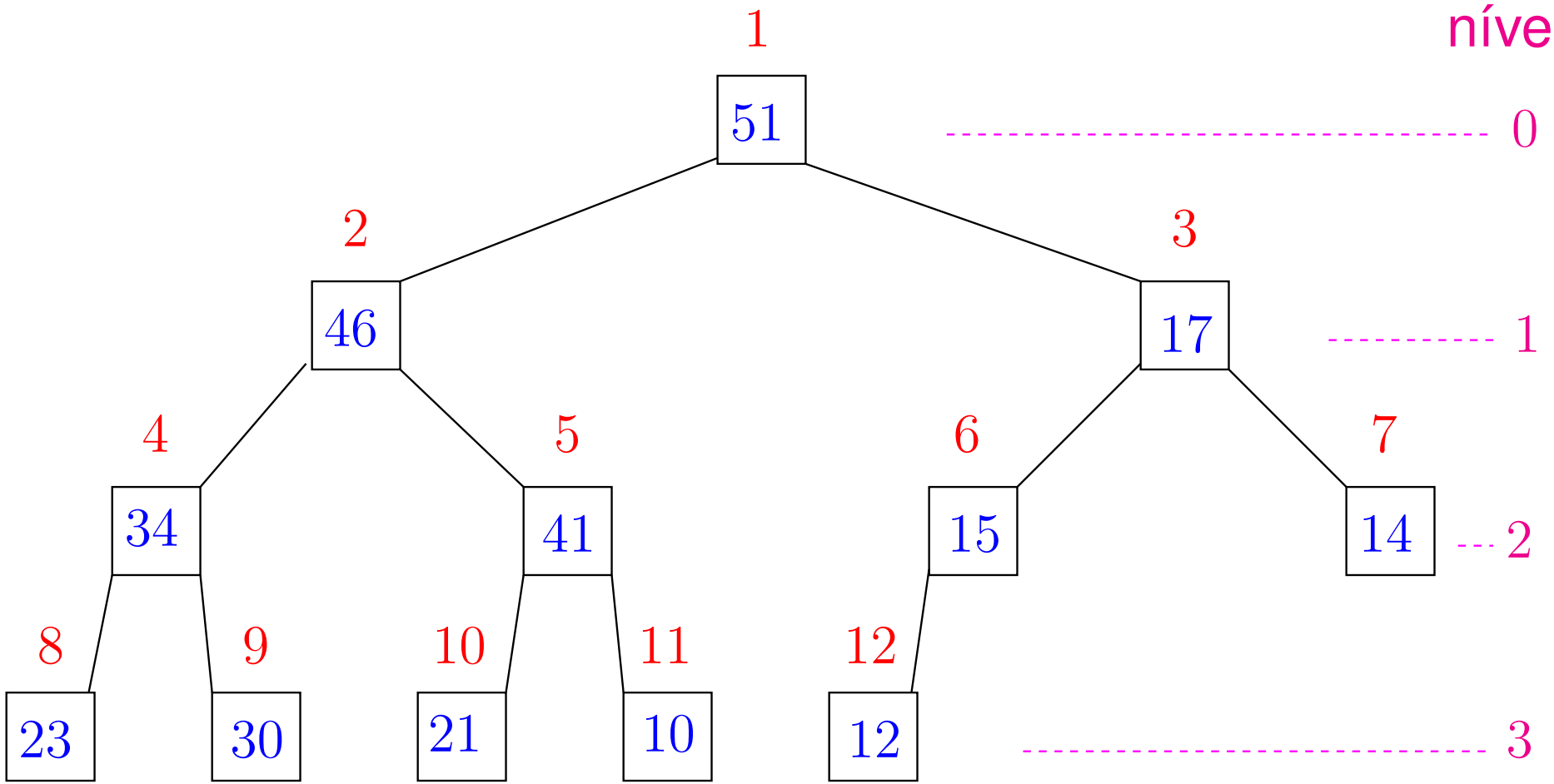
para todo $i = 2, 3, \ldots, m$.

De uma forma mais geral, $A[j \ldots m]$ 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



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

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)
    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)
```

Sobe-Heap

Exercício: Escreva uma função SOBE-HEAP (A, m, i) que recebe $A[1\,..\,m]$ e $i \geq 1$ tais que $A[1\,..\,m]$ é um heap exceto pela condição $A[\lfloor i/2 \rfloor] \geq A[i]$ que pode estar violada, e rearranja A de modo que passe a ser um heap.

Sua função deve ter complexidade $O(\lg m)$.

Sobe-Heap

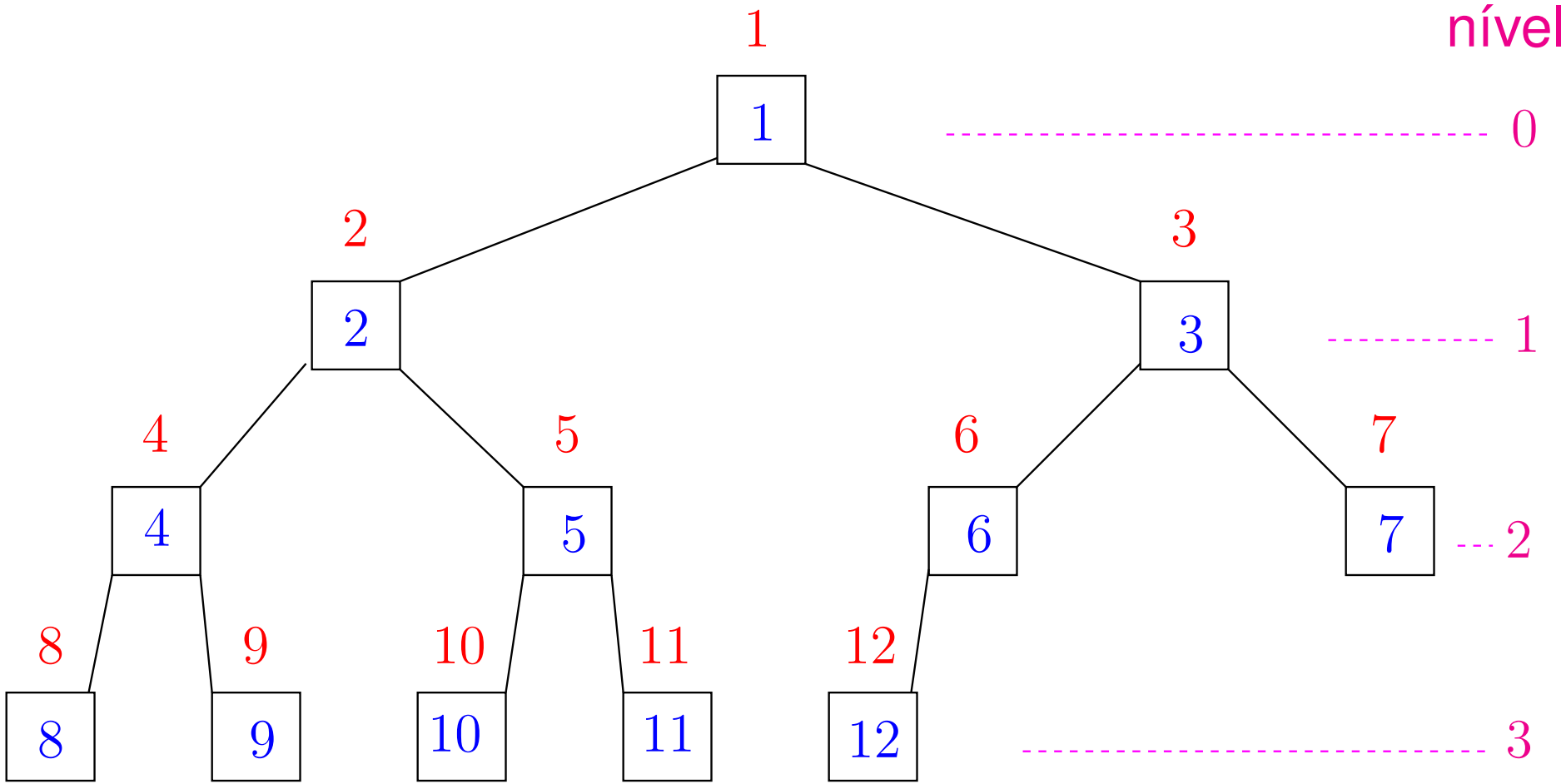
Exercício: Escreva uma função SOBE-HEAP (A, m, i) que recebe $A[1\,..\,m]$ e $i \geq 1$ tais que $A[1\,..\,m]$ é um heap exceto pela condição $A[\lfloor i/2 \rfloor] \geq A[i]$ que pode estar violada, e rearranja A de modo que passe a ser um heap.

Sua função deve ter complexidade $O(\lg m)$.

```
CONSTRÓI-HEAP (A, m)
1    para i ← 2 até n faça
2            SOBE-HEAP (A, i, i)
```

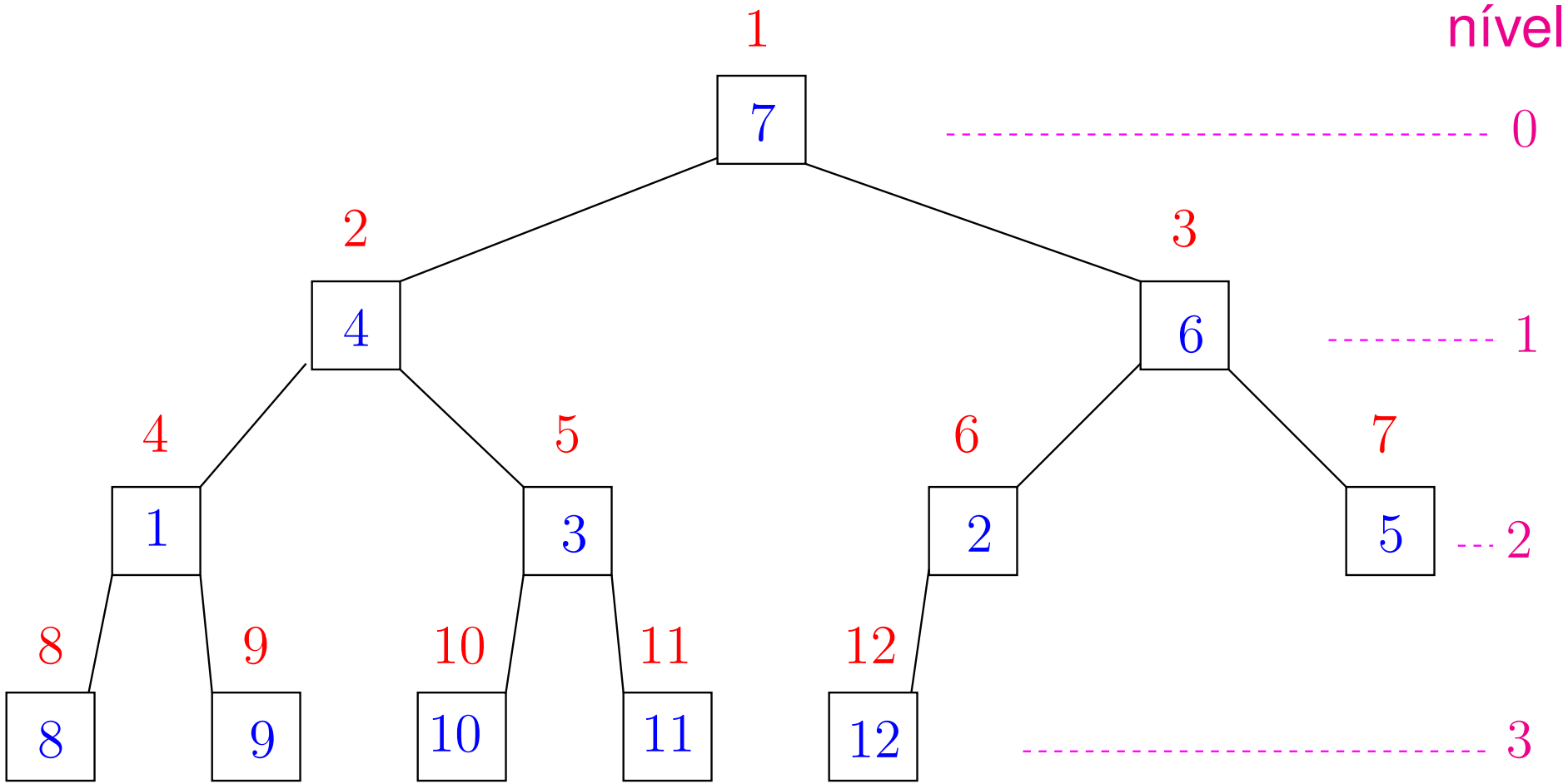
Qual é a complexidade dessa implementação do CONSTRÓI-HEAP?

Exemplo



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

Exemplo



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

Construção de um heap

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

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

Construção de um heap

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

CONSTRÓI-HEAP (A, n)
1 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
2 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 ????.

Construção de um heap

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

CONSTRÓI-HEAP (A, n)
1 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
2 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 $\frac{n}{2}\,\mathrm{O}(\lg n) = \mathrm{O}(n \lg n)$.

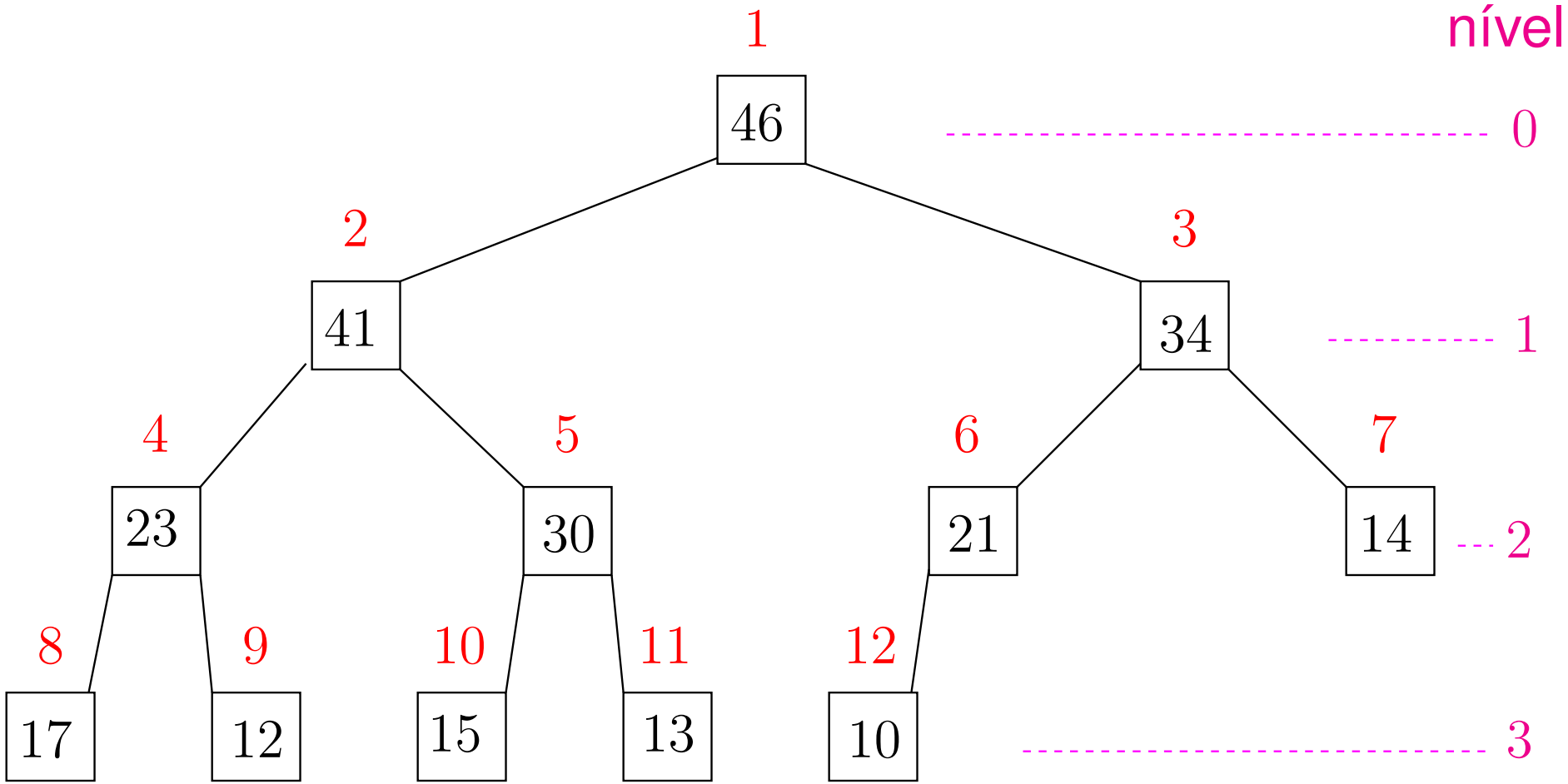
Análise mais cuidadosa: $T(n)$ é $\mathrm{O}(n)$.

Heap sort

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

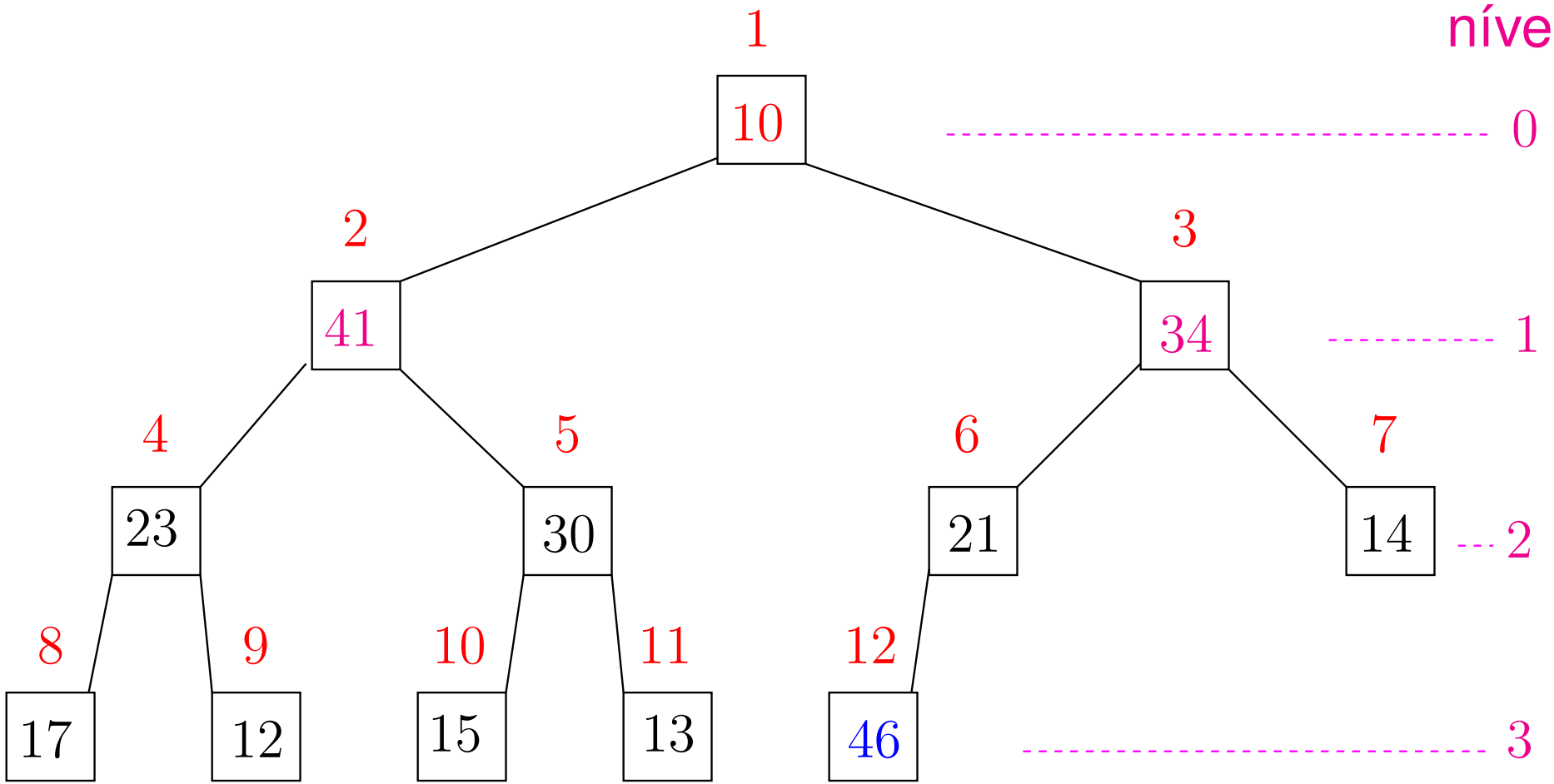
```
HEAPSORT (A, n)
    CONSTRÓI-HEAP (A, n)        ▷ pré-processamento
    m ← n
    para i ← n decrescendo até 2 faça
        A[1] ↔ A[i]
        m ← m − 1
        DESCE-HEAP (A, m, 1)
```

Heap sort



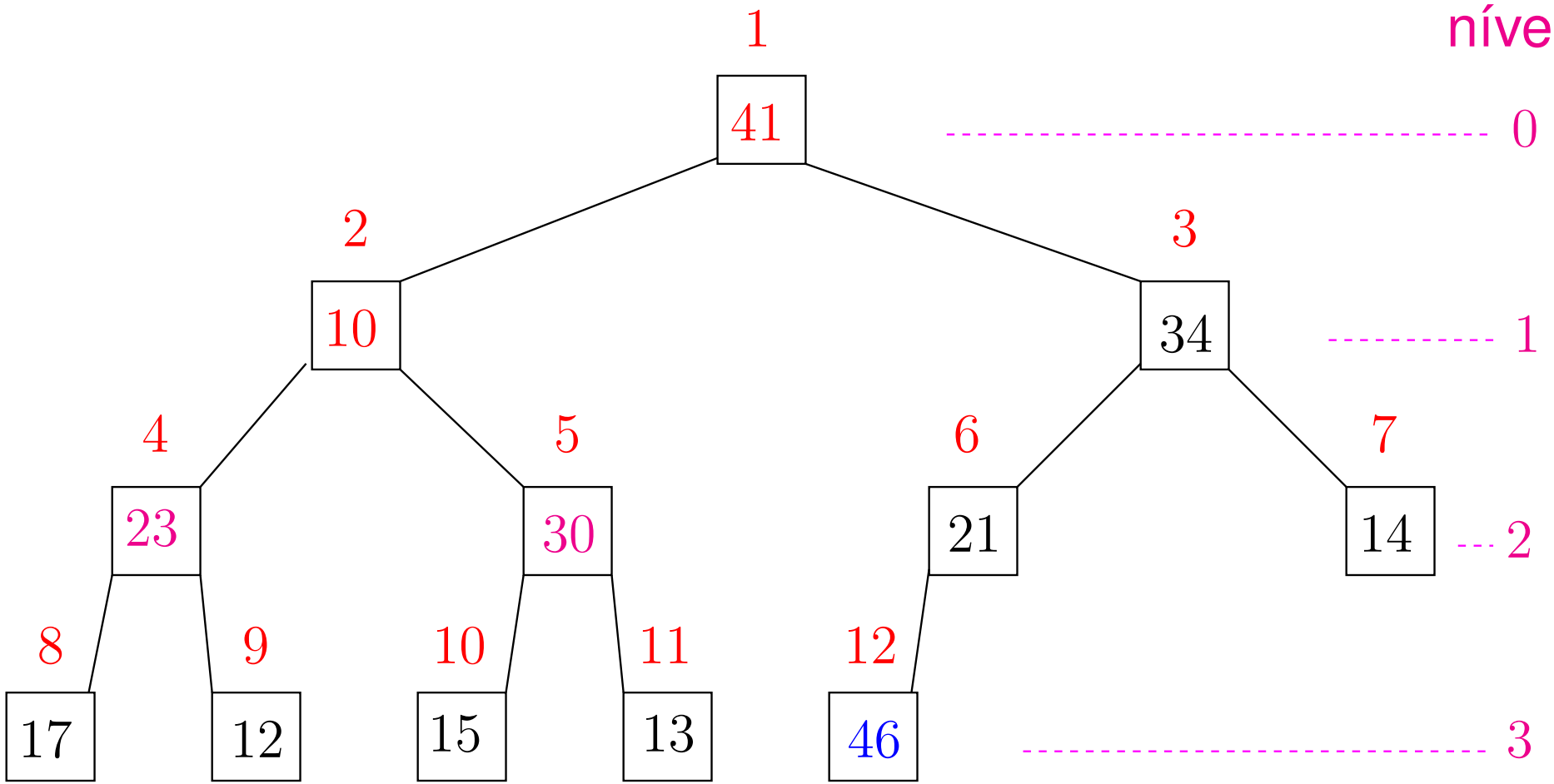
1	2	3	4	5	6	7	8	9	10	11	12
46	41	34	23	30	21	14	17	12	15	13	10

Heap sort



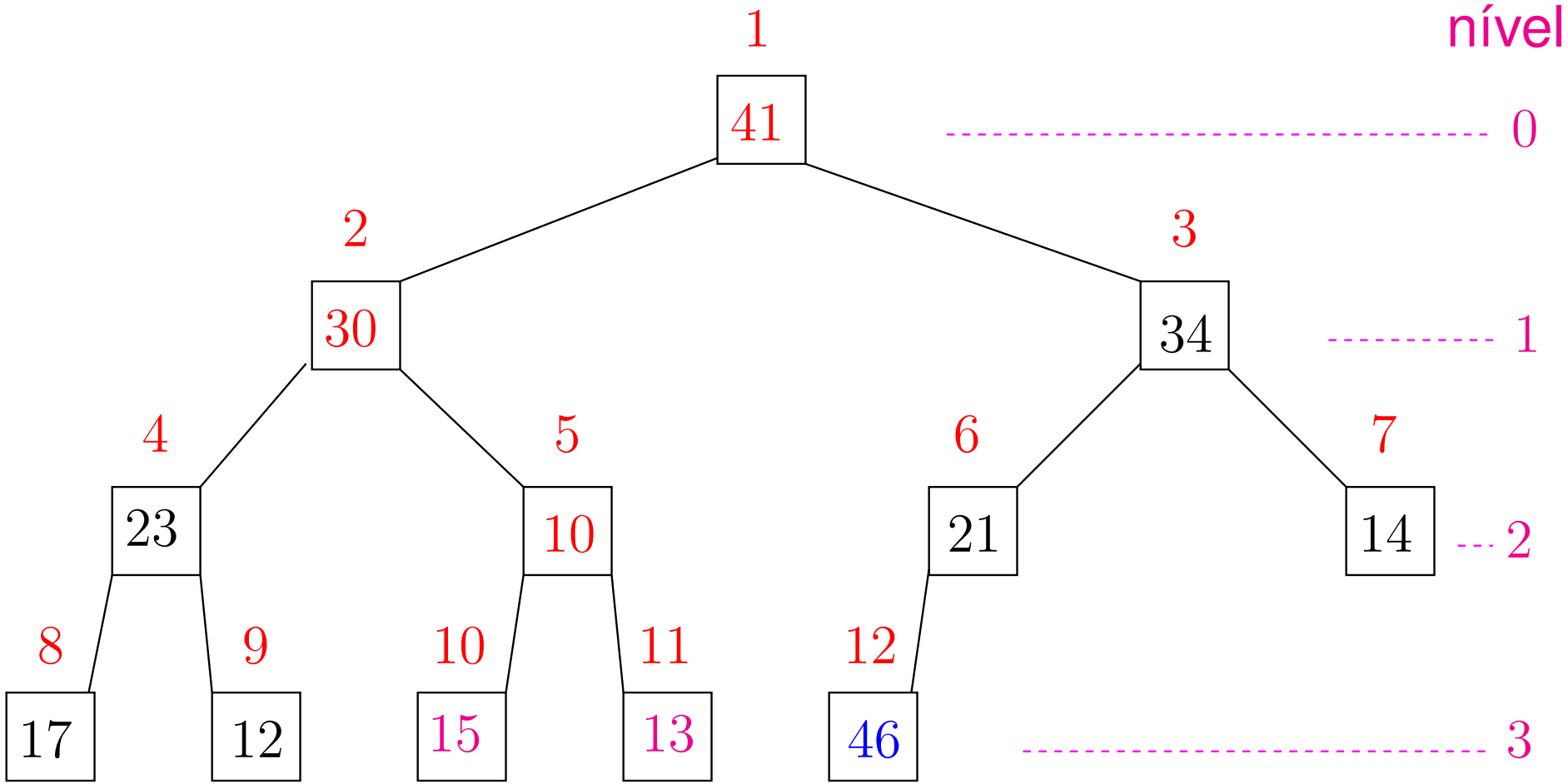
1	2	3	4	5	6	7	8	9	10	11	12
10	41	34	23	30	21	14	17	12	15	13	46

Heap sort



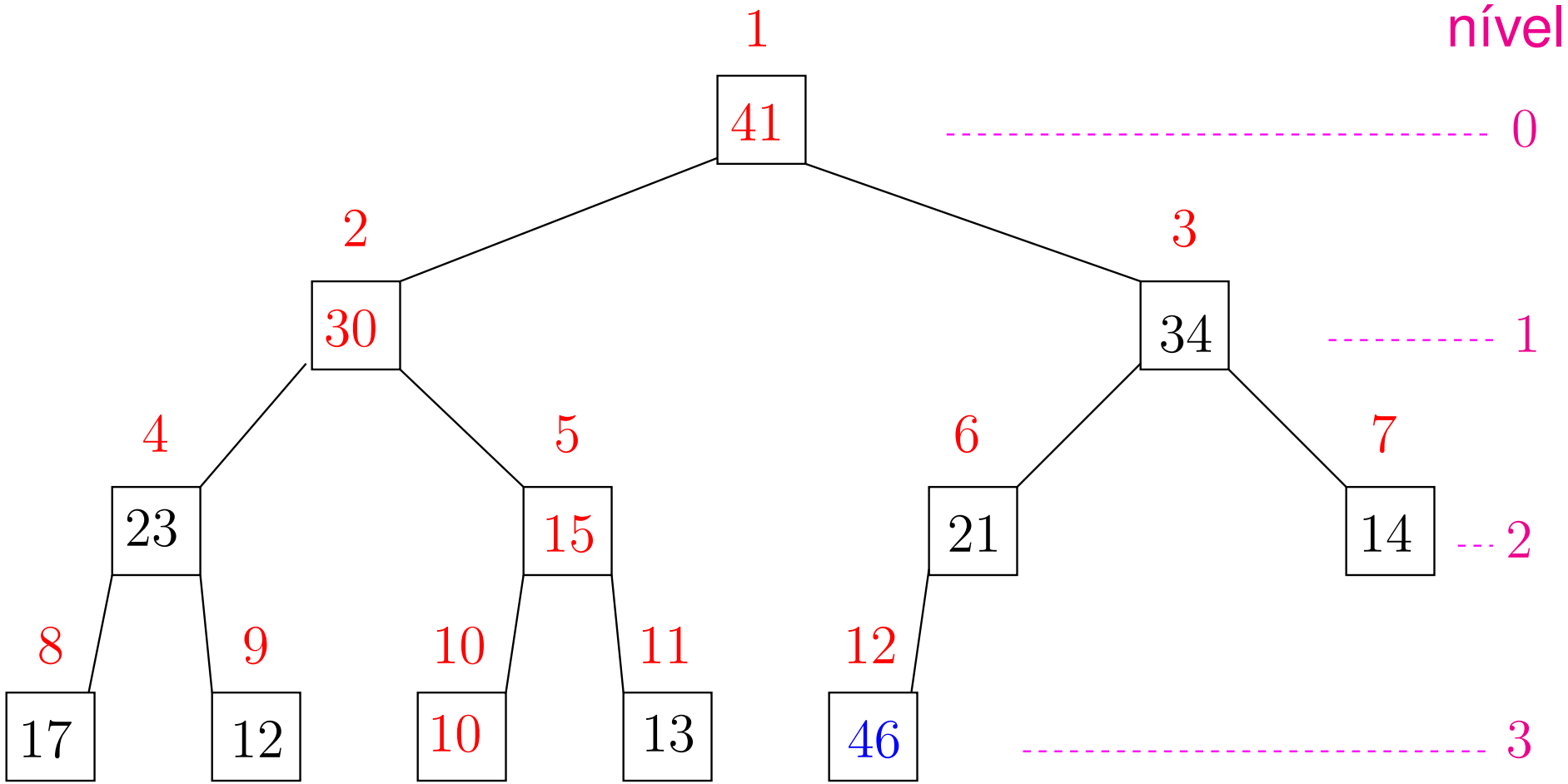
1	2	3	4	5	6	7	8	9	10	11	12
41	10	34	23	30	21	14	17	12	15	13	46

Heap sort



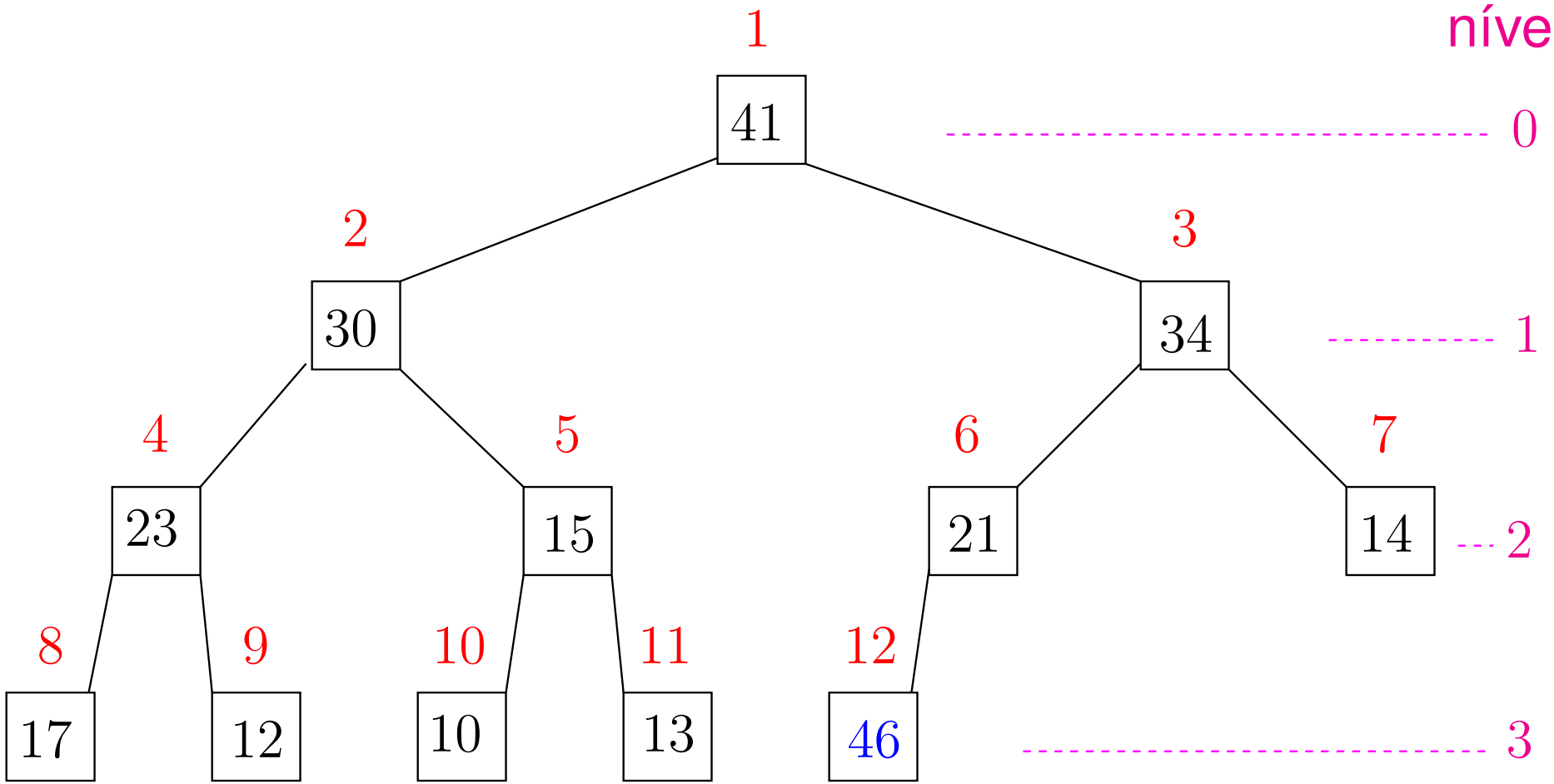
1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	10	21	14	17	12	15	13	46

Heap sort



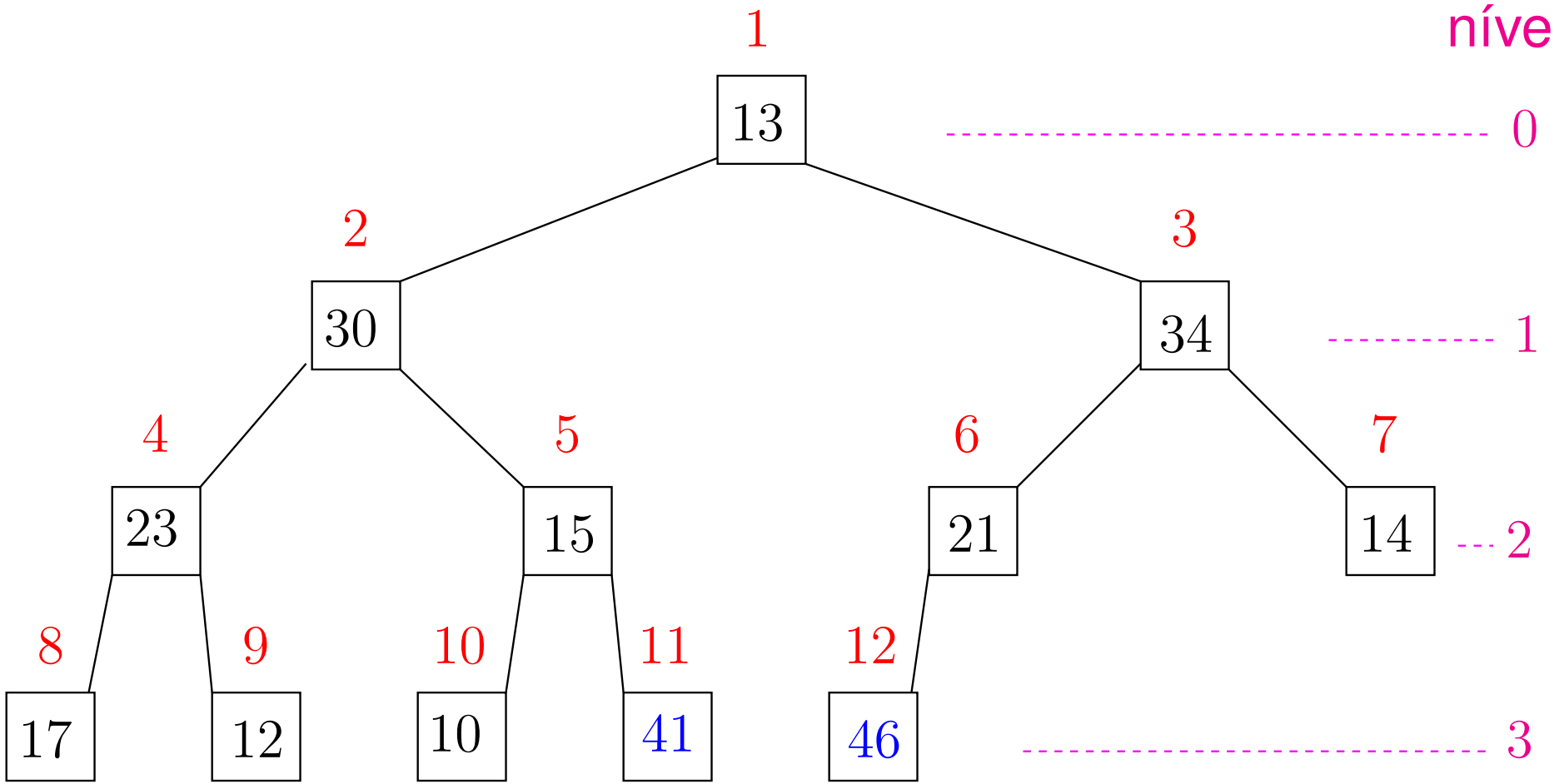
1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	15	21	14	17	12	10	13	46

Heap sort



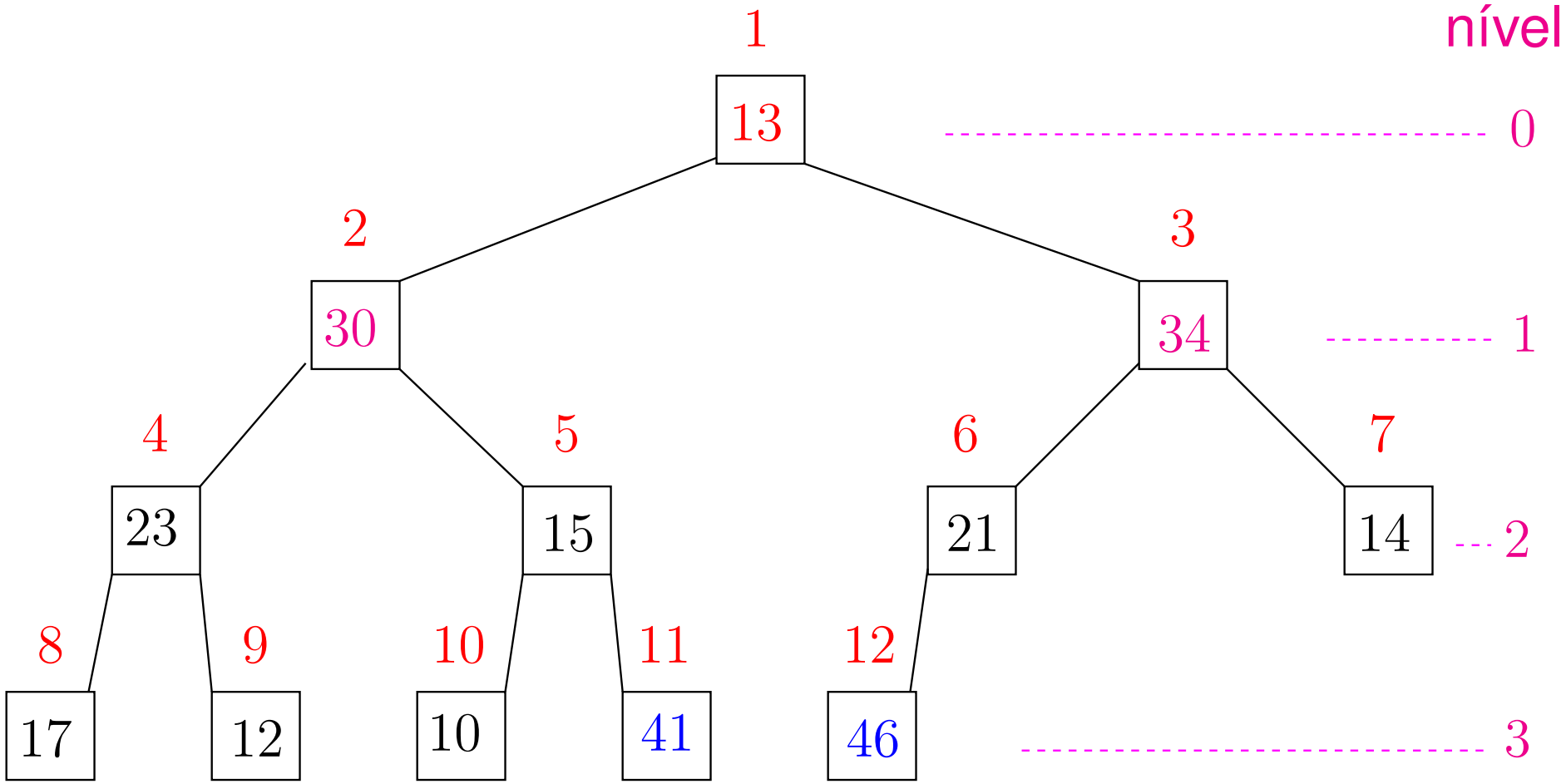
1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	15	21	14	17	12	10	13	46

Heap sort



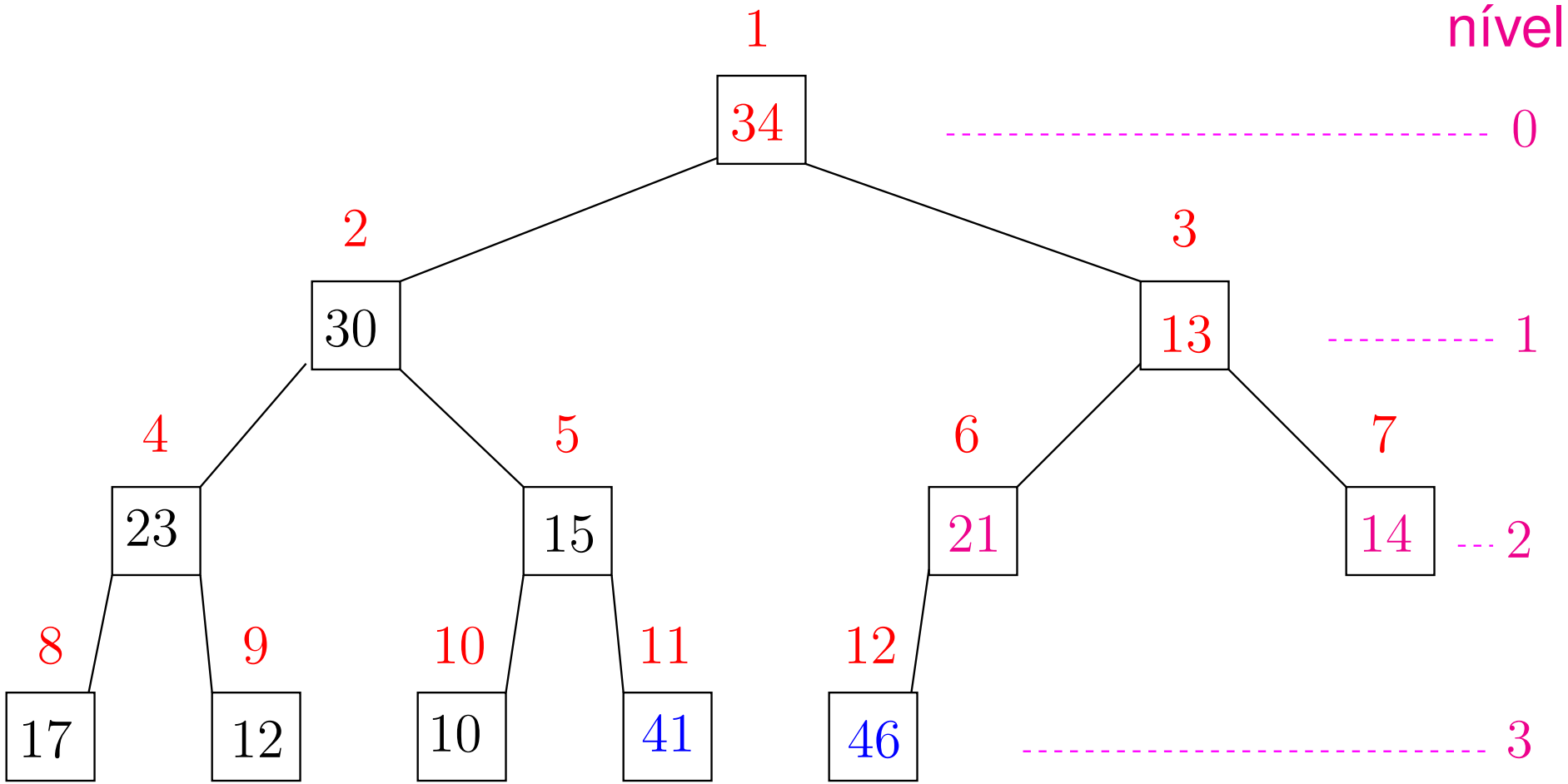
1	2	3	4	5	6	7	8	9	10	11	12
13	30	34	23	15	21	14	17	12	10	41	46

Heap sort



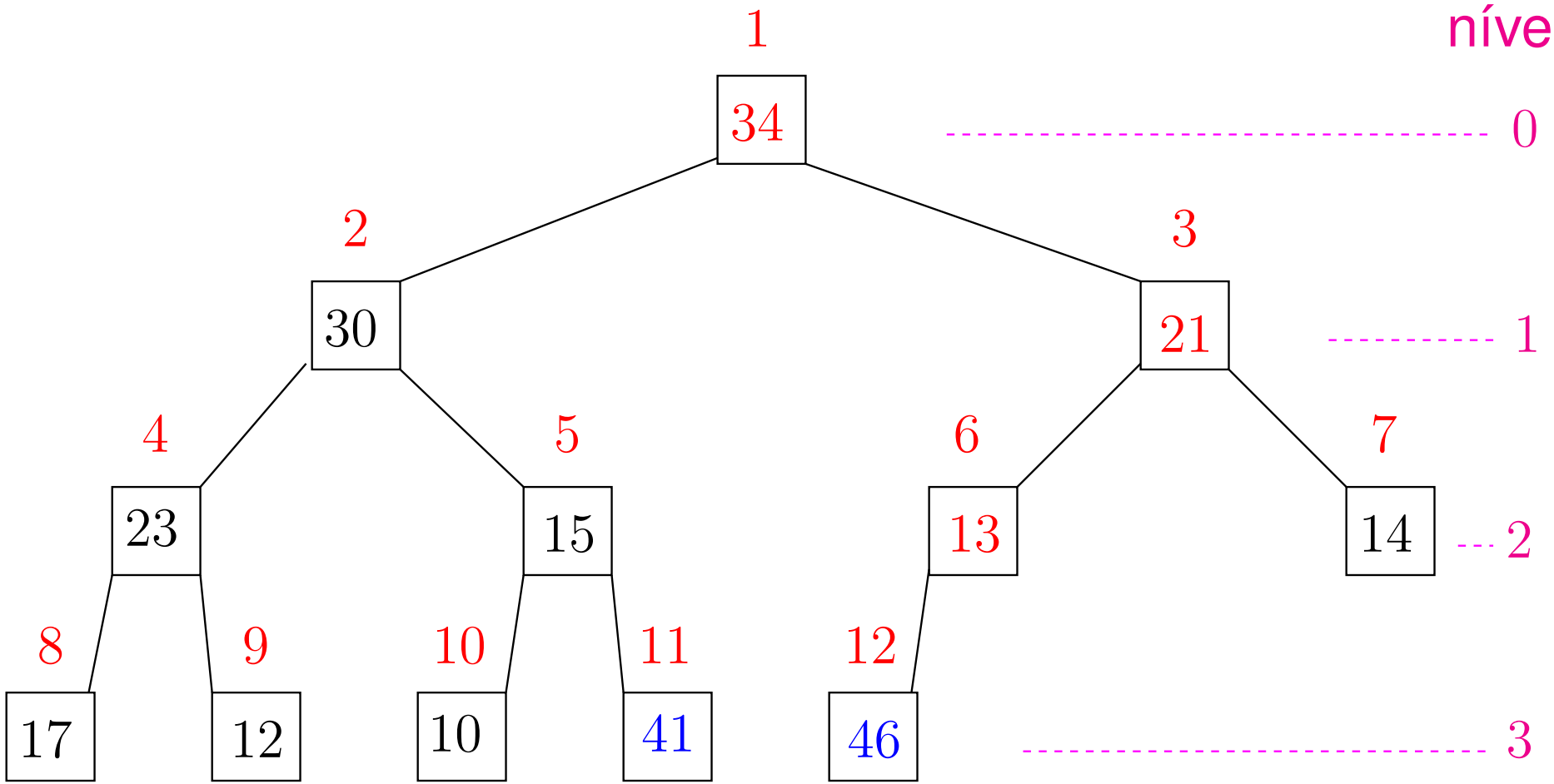
1	2	3	4	5	6	7	8	9	10	11	12
13	30	34	23	15	21	14	17	12	10	41	46

Heap sort



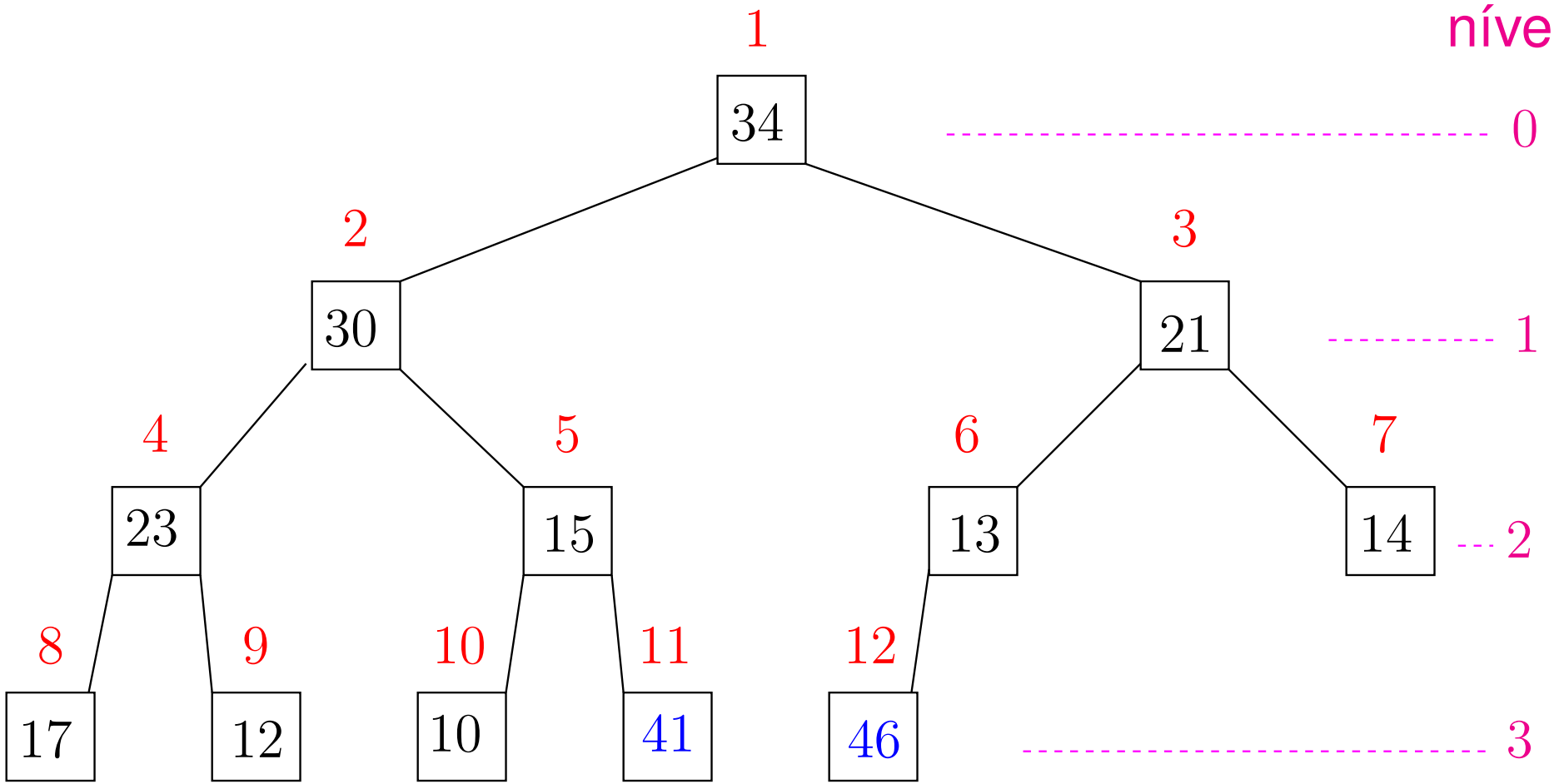
1	2	3	4	5	6	7	8	9	10	11	12
34	30	13	23	15	21	14	17	12	10	41	46

Heap sort



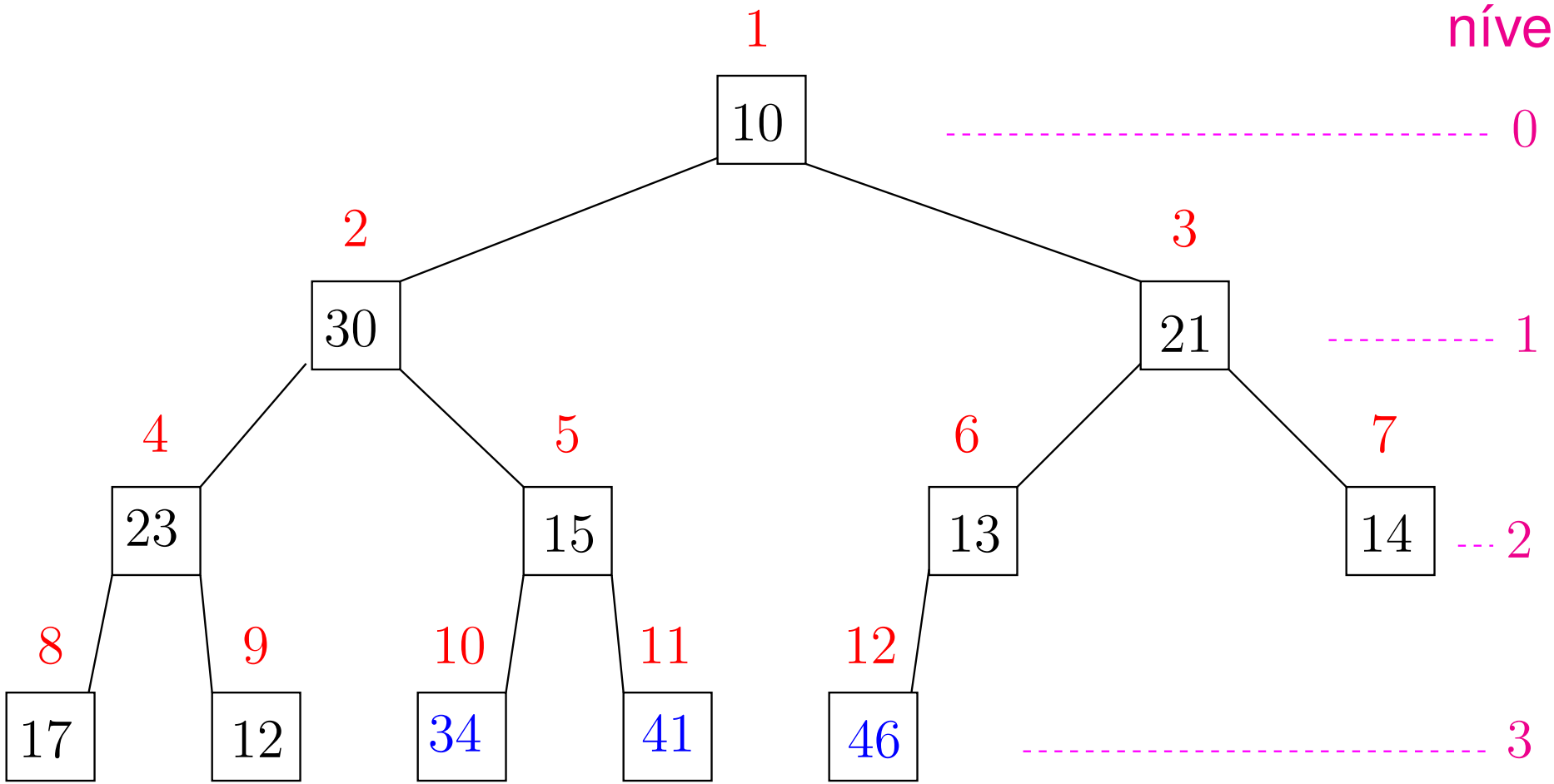
1	2	3	4	5	6	7	8	9	10	11	12
34	30	21	23	15	13	14	17	12	10	41	46

Heap sort



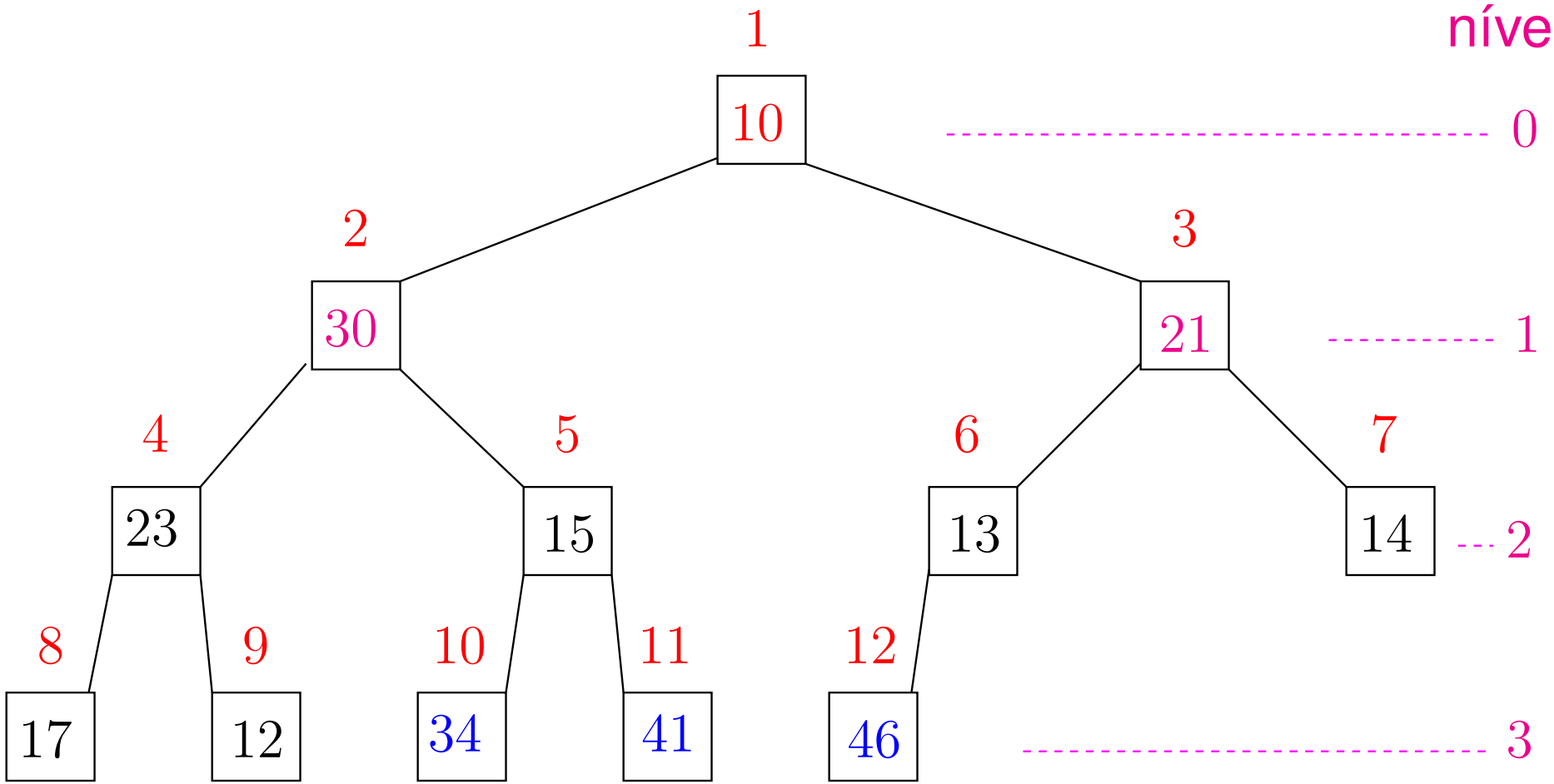
1	2	3	4	5	6	7	8	9	10	11	12
34	30	21	23	15	13	14	17	12	10	41	46

Heap sort



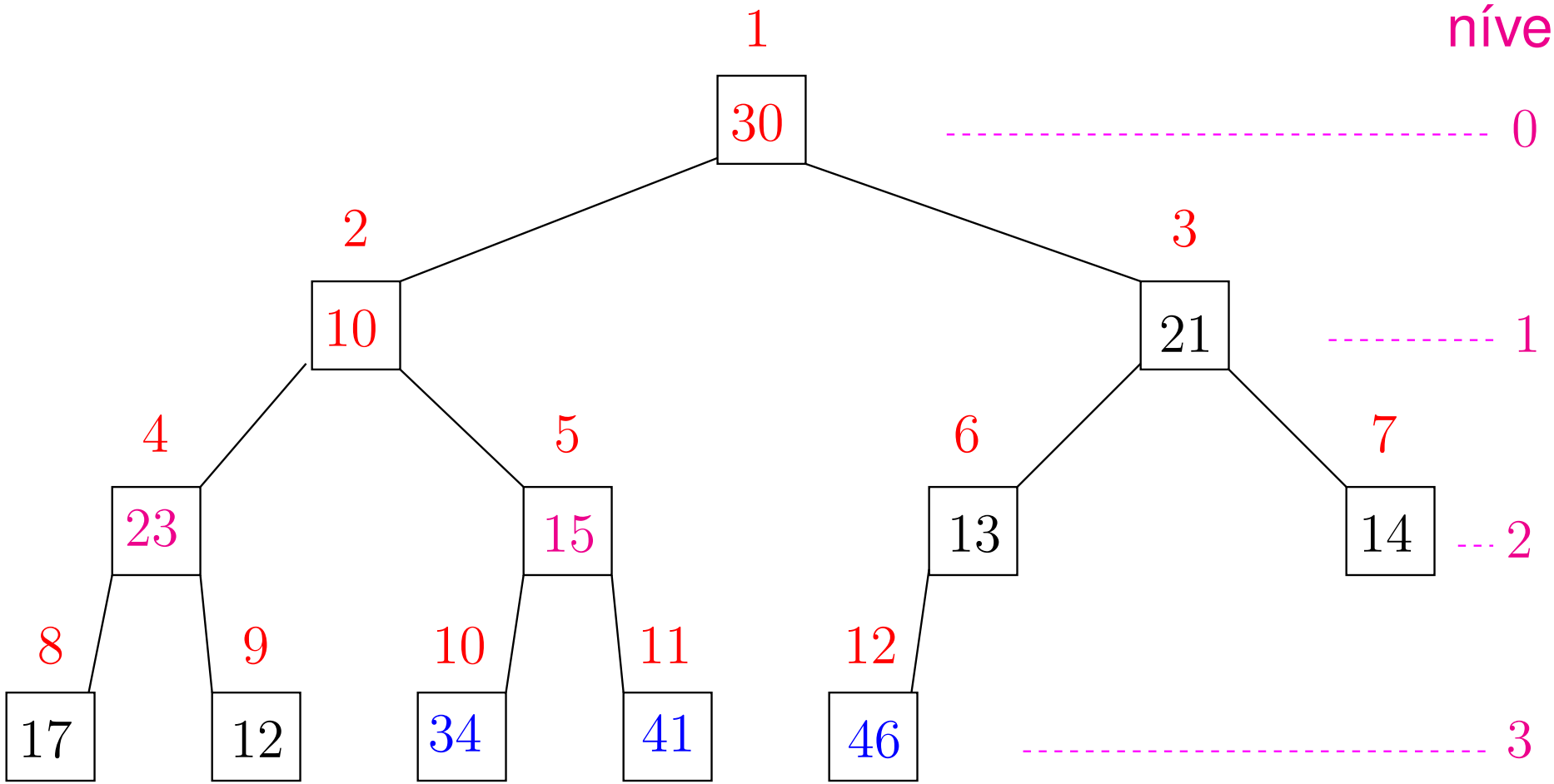
1	2	3	4	5	6	7	8	9	10	11	12
10	30	21	23	15	13	14	17	12	34	41	46

Heap sort



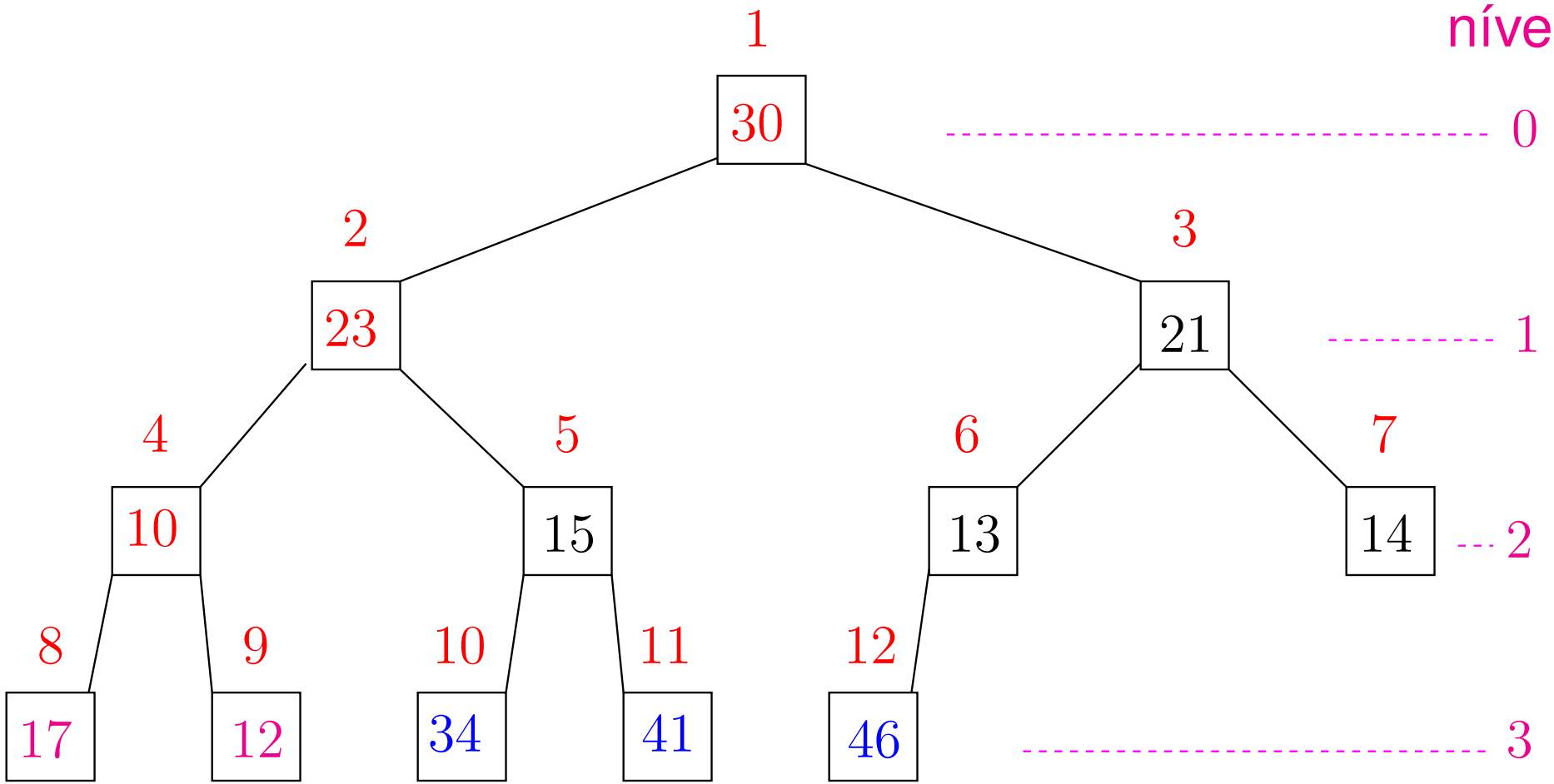
1	2	3	4	5	6	7	8	9	10	11	12
10	30	21	23	15	13	14	17	12	34	41	46

Heap sort



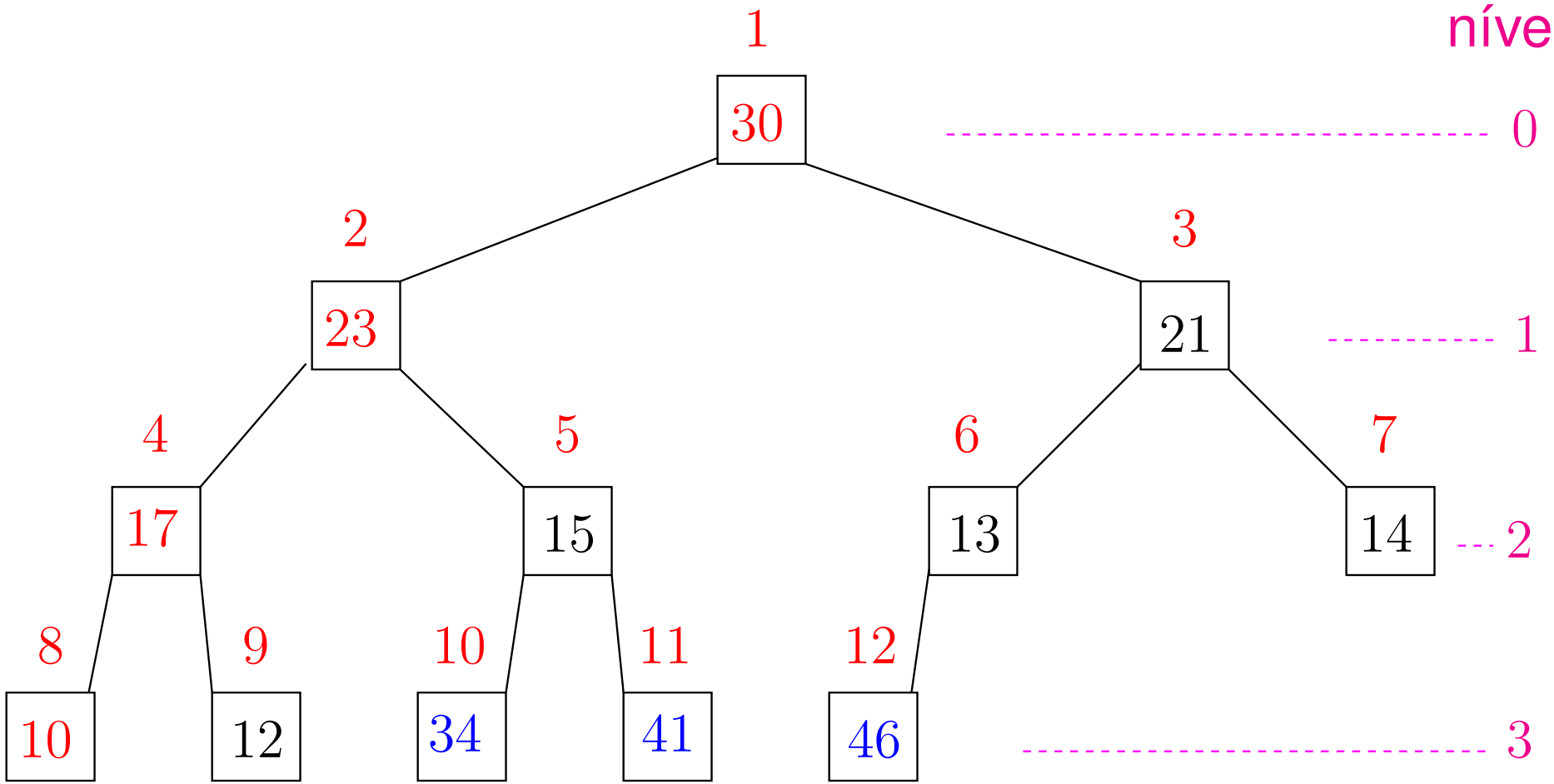
1	2	3	4	5	6	7	8	9	10	11	12
30	10	21	23	15	13	14	17	12	34	41	46

Heap sort



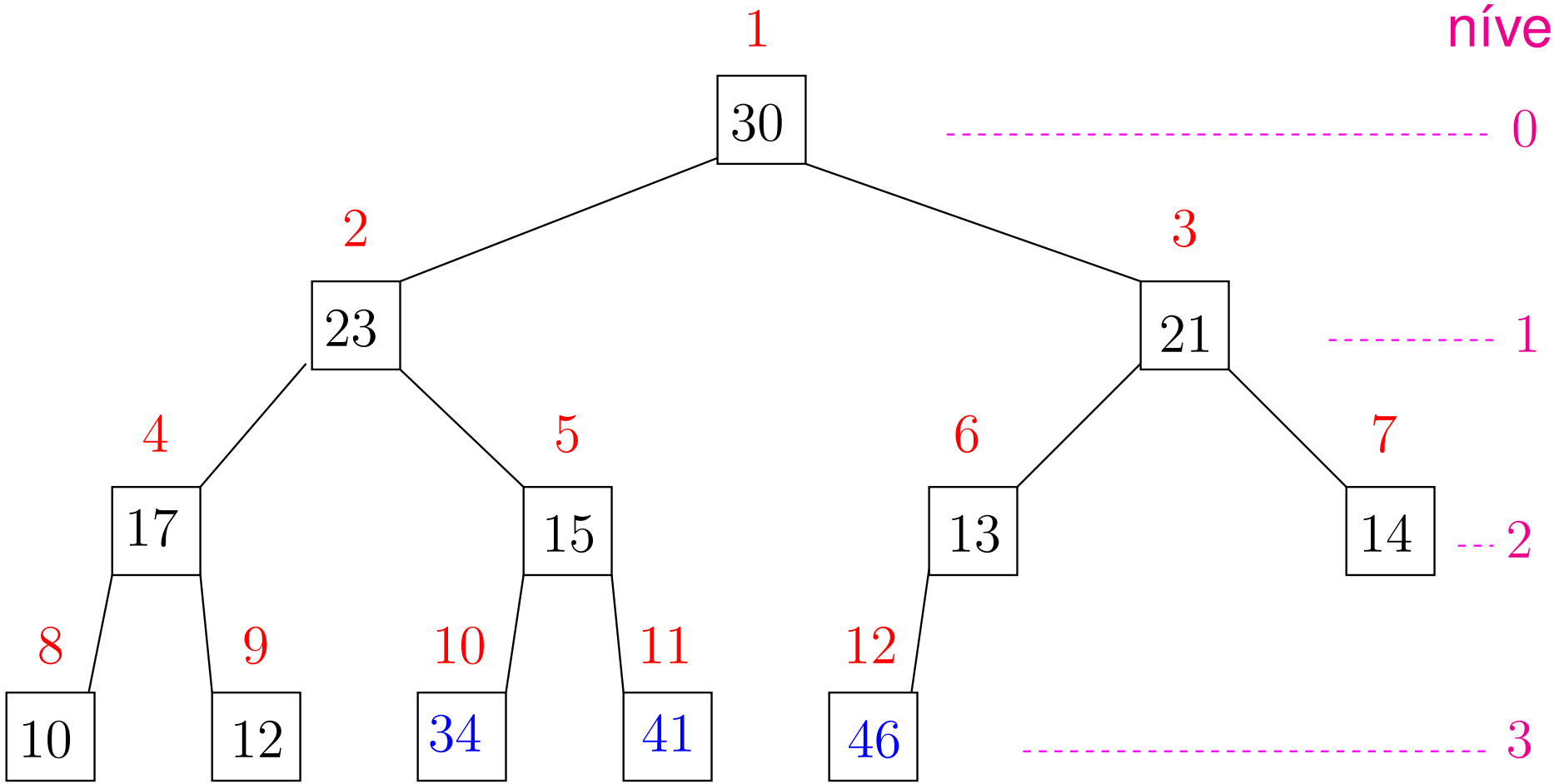
1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	10	15	13	14	17	12	34	41	46

Heap sort



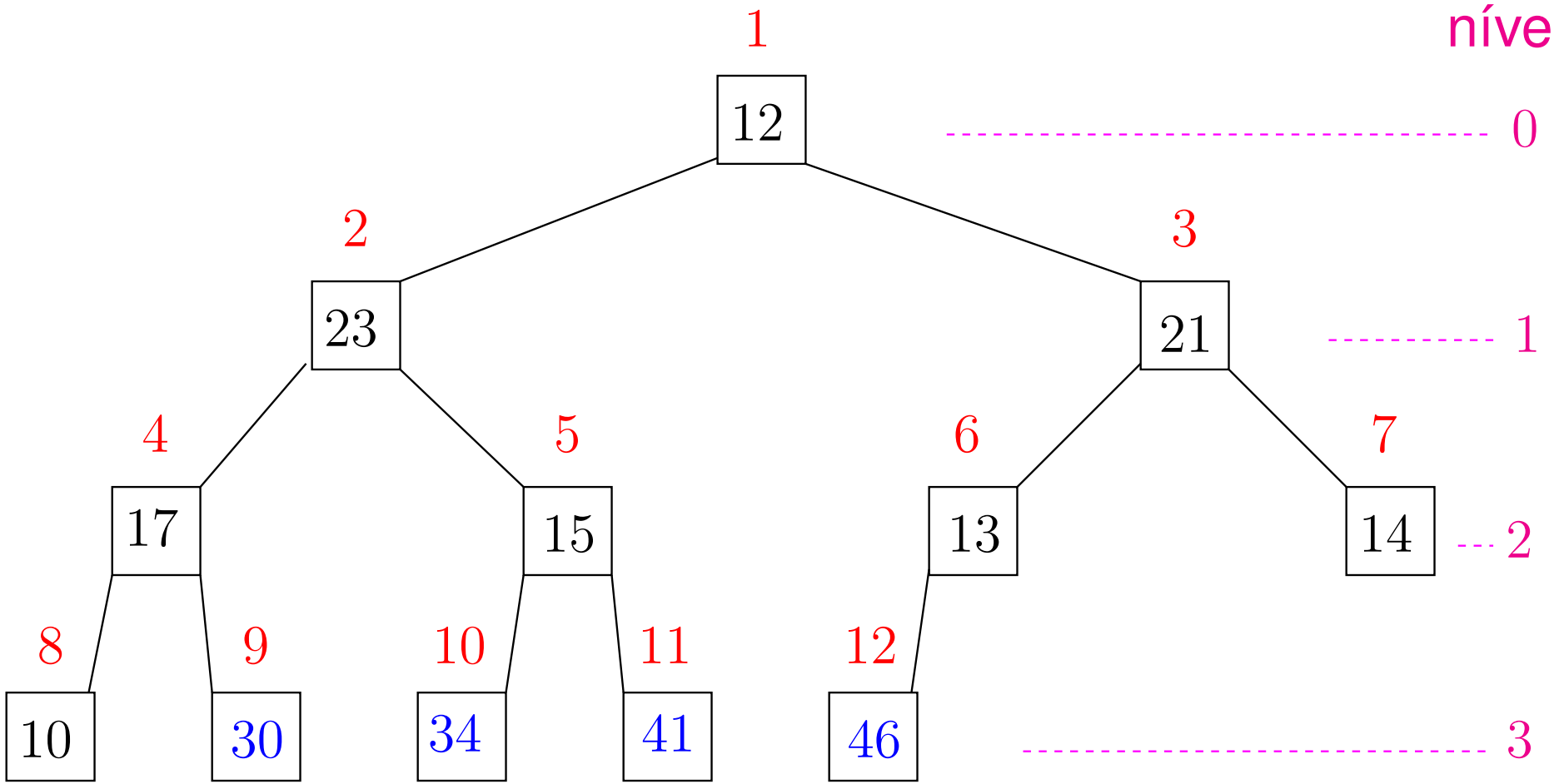
1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	17	15	13	14	10	12	34	41	46

Heap sort



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



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

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	$=$	$\Theta(n)$
6	$=$	$n\mathrm{O}(\lg n)$
total	$=$	$n\mathrm{O}(\lg n) + \Theta(4n+1) = \mathrm{O}(n \lg n)$

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

Exercícios

Exercício 9.A

A altura de i em $A[1\,..\,m]$ 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 9.B

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

Exercício 9.C

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

Exercício 9.D

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

Exercício 9.E

Considere um heap $A[1\,..\,m]$; 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 9.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 $\mathrm{O}(\log k)$.
(Curiosidade: Essa é a recorrência do DESCE-HEAP (A, m, i) se interpretarmos k como sendo o número de nós na subárvore com raiz i).

Exercício 9.G

Escreva uma versão iterativa do algoritmo DESCE-HEAP. Faça uma análise do consumo de tempo do algoritmo.

Mais exercícios ainda

Exercício 9.H

Discuta a seguinte variante do algoritmo DESCE-HEAP:

```
D-H (A, m, i)
    e ← 2i
    d ← 2i + 1
    se e ≤ m e A[e] > A[i]
            então A[i] ↔ A[e]
                    D-H (A, m, e)
    se d ≤ m e A[d] > A[i]
            então A[i] ↔ A[d]
                    D-H (A, m, d)
```

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 $\mathrm{O}(n \lg n)$.

Ordenação: limite inferior

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

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

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 $\mathrm{O}(n \lg n)$.

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 $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

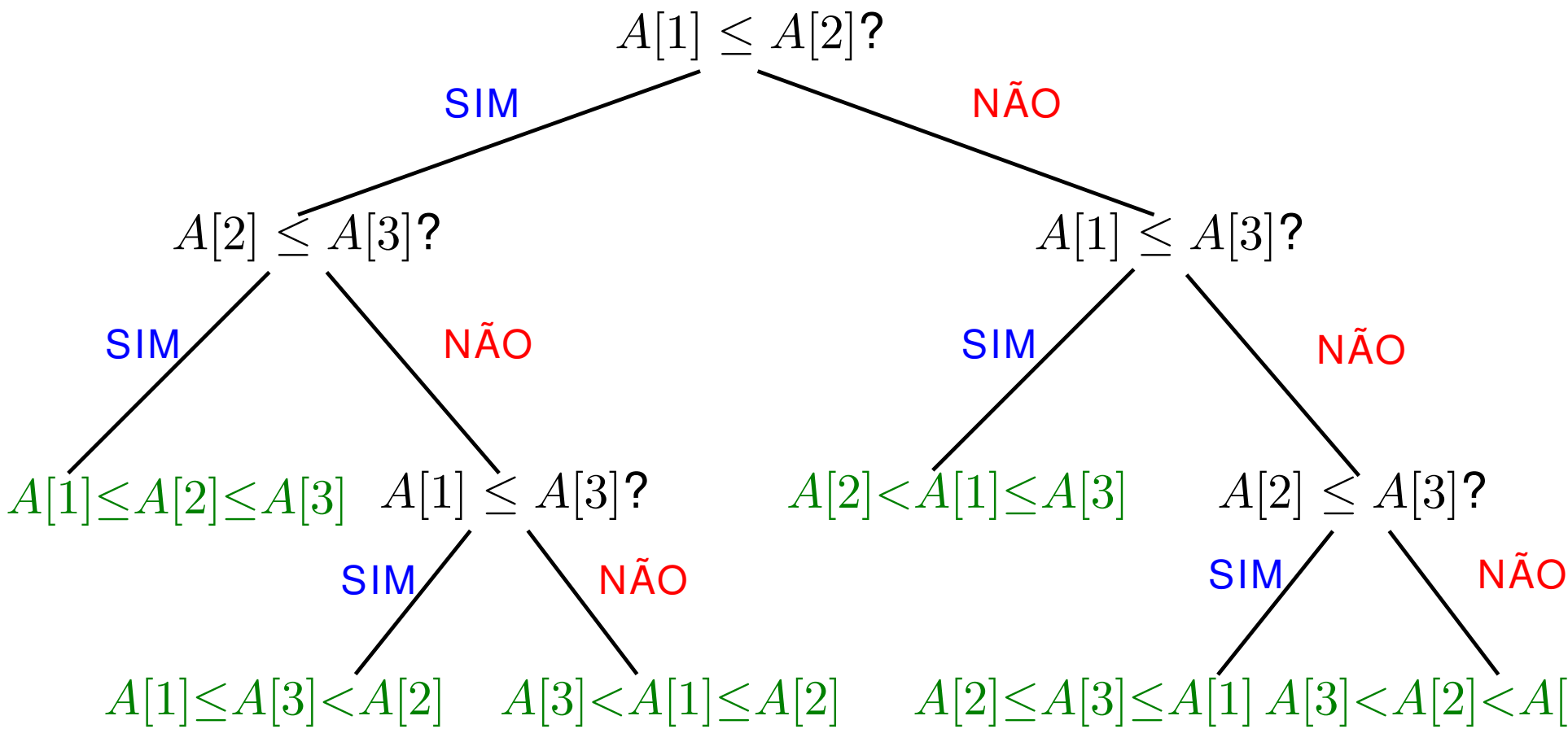
NÃO, se o algoritmo é baseado em comparações.

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 $1, \ldots, n$ 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 $1, \ldots, n$ 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, $h \geq 1$.
O número de folhas de uma árvore de altura h é a soma do número de folhas de suas sub-árvores; que têm altura $\leq h - 1$. Logo, o número de folhas de uma árvore de altura h é não superior a

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

Limite inferior

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

$$(n!)^2 \;=\; \prod_{i=0}^{n-1} (n-i)(i+1) \;\geq\; \prod_{i=1}^{n} n \;=\; n^n$$

Portanto,

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

Conclusão

Todo algoritmo de ordenação baseado em comparações faz

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

comparações no pior caso

Exercícios

Exercício 16.A
Desenhe a árvore de decisão para o SELECTION-SORT aplicado a $A[1\,..\,3]$ com todos os elementos distintos.

Exercício 16.B [CLRS 8.1-1]
Qual o menor profundidade (= menor nível) que uma folha pode ter em uma árvore de decisão que descreve um algoritmo de ordenação baseado em comparações?

Exercício 16.C [CLRS 8.1-2]
Mostre que $\lg(n!) = \Omega(n \lg n)$ sem usar a fórmula de Stirling. Sugestão: Calcule $\sum_{k=n/2}^{n} \lg k$. Use as técnicas de CLRS A.2.