AULA 26

Ordenação: algoritmo Heapsort

PF 10

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

Ordenação

$\mathtt{v}[1\,..\,\mathtt{n}]$ é crescente se $\mathtt{v}[1] \leq \cdots \leq \mathtt{v}[\mathtt{n}]$.

Problema: Rearranjar um vetor $\mathtt{v}[1\,..\,\mathtt{n}]$ de modo 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

Heapsort

O Heapsort ilustra o uso de estruturas de dados no projeto de algoritmos eficientes.
Rearranjar um vetor $v[1\,..\,n]$ de modo que ele fique crescente.

Entra:

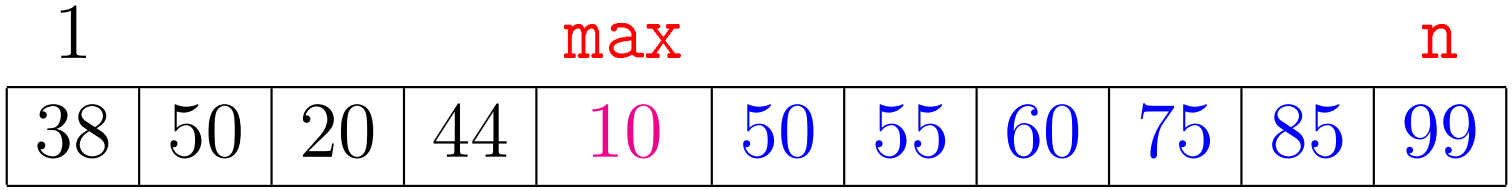
1										n
33	55	33	44	33	22	11	99	22	55	77

Sai:

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

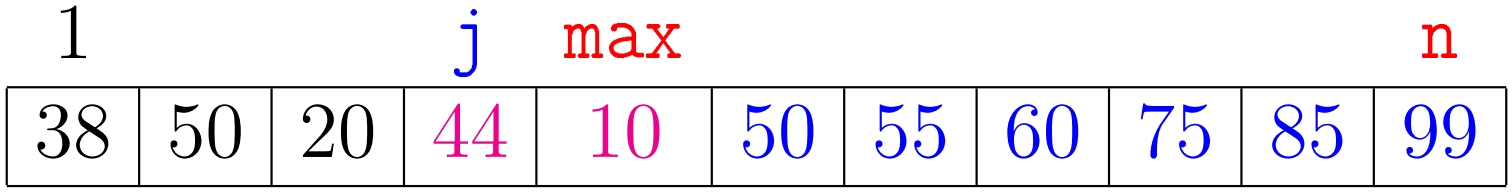
Ordenação por seleção

$\texttt{i} = 5$



Ordenação por seleção

$\texttt{i} = 5$



Ordenação por seleção

i = 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

i = 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

i = 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

i = 5

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

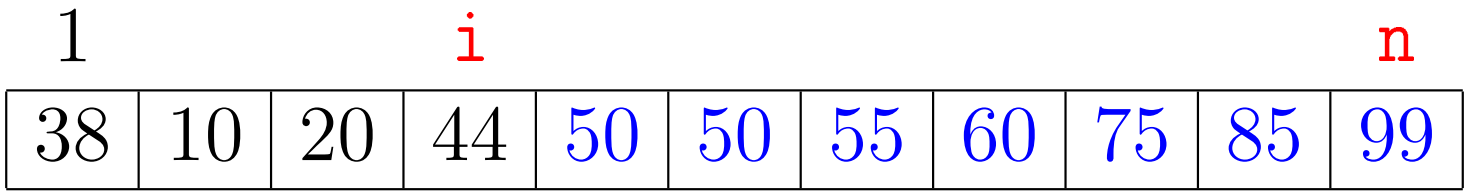
1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

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

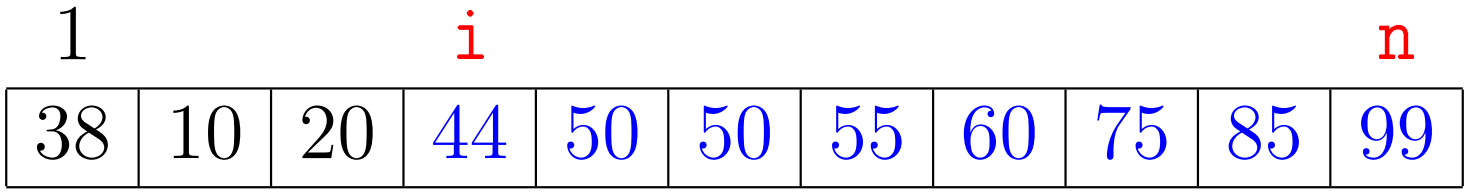
j	max									n
38	50	20	44	10	50	55	60	75	85	99

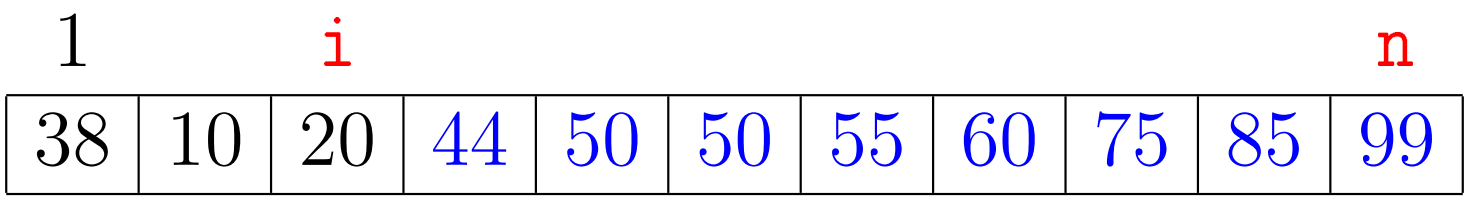
1	max									n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

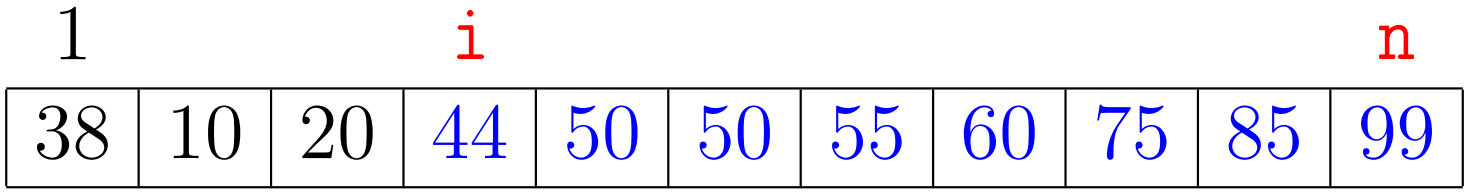


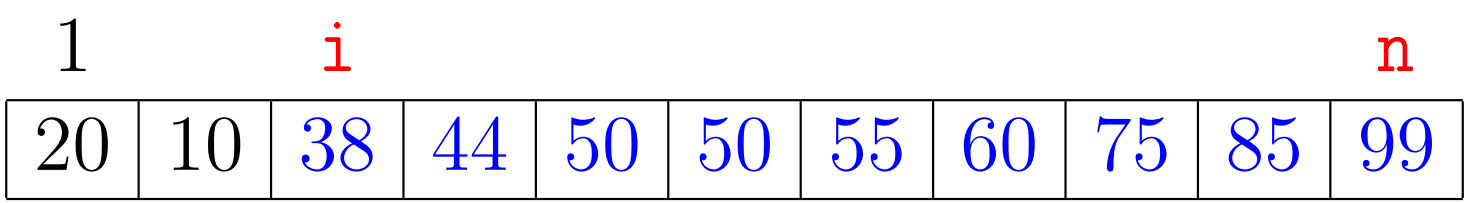
Ordenação por seleção

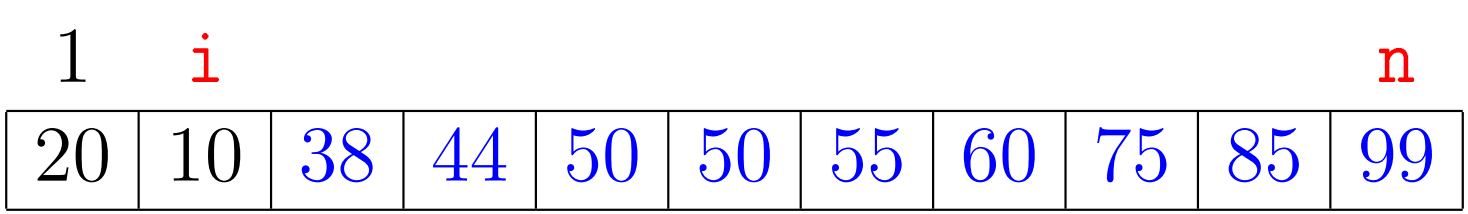




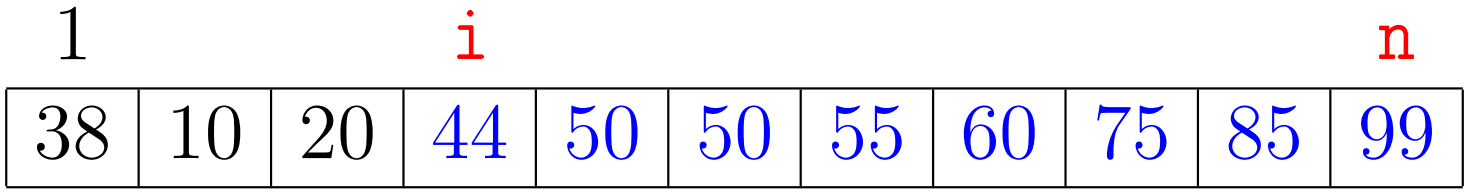
Ordenação por seleção

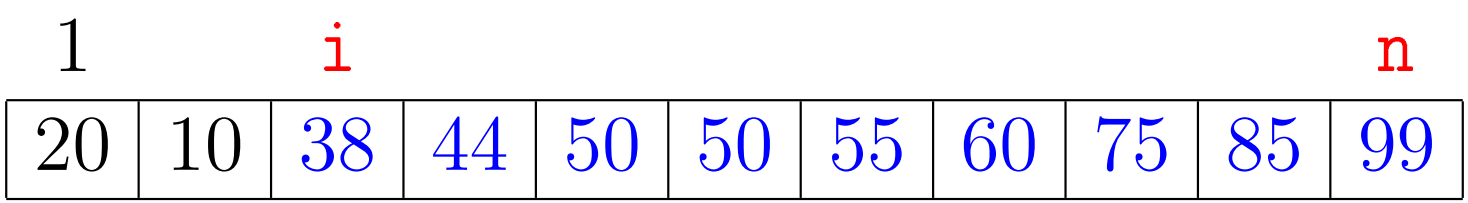


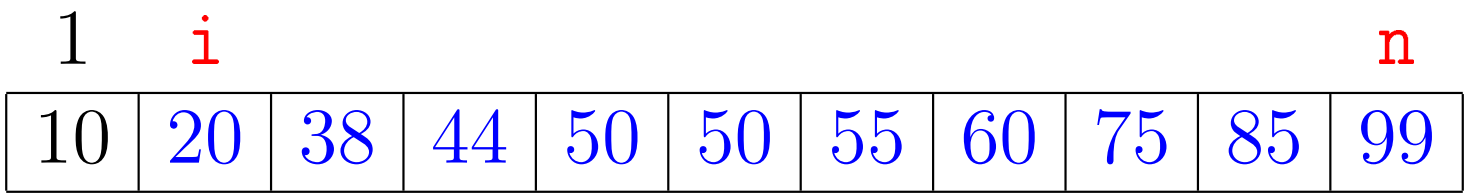


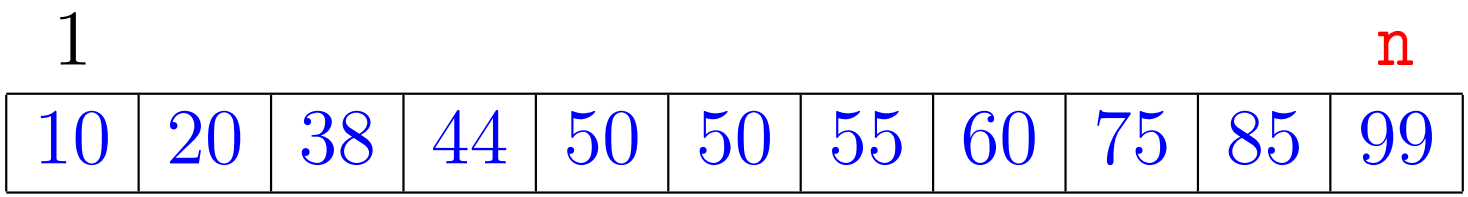


Ordenação por seleção









Função selecao

Algoritmo rearranja $v[0\,..\,n-1]$ em ordem crescente

```
void selecao (int n, int v[])
{
    int i, j, max, x;
1   for (i = n-1; /*B*/ i > 0; i--) {
2       max = i;
3       for (j = i-1; j >= 0; j--)
4           if (v[j] > v[max]) max = j;
5       x=v[i]; v[i]=v[max]; v[max]=x;
    }
}
```

Função selecao

Algoritmo rearranja v[1..n] em ordem crescente

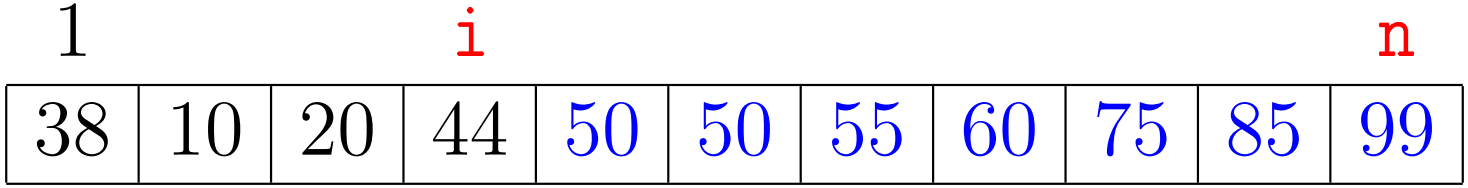
```
void selecao (int n, int v[])
{
    int i, j, max, x;
1   for (i = n; /*B*/ i >= 2; i--) {
2       max = i;
3       for (j = i-1; j >= 1; j--)
4           if (v[j] > v[max]) max = j;
5       x=v[i]; v[i]=v[max]; v[max]=x;
    }
}
```

Função `selecao`

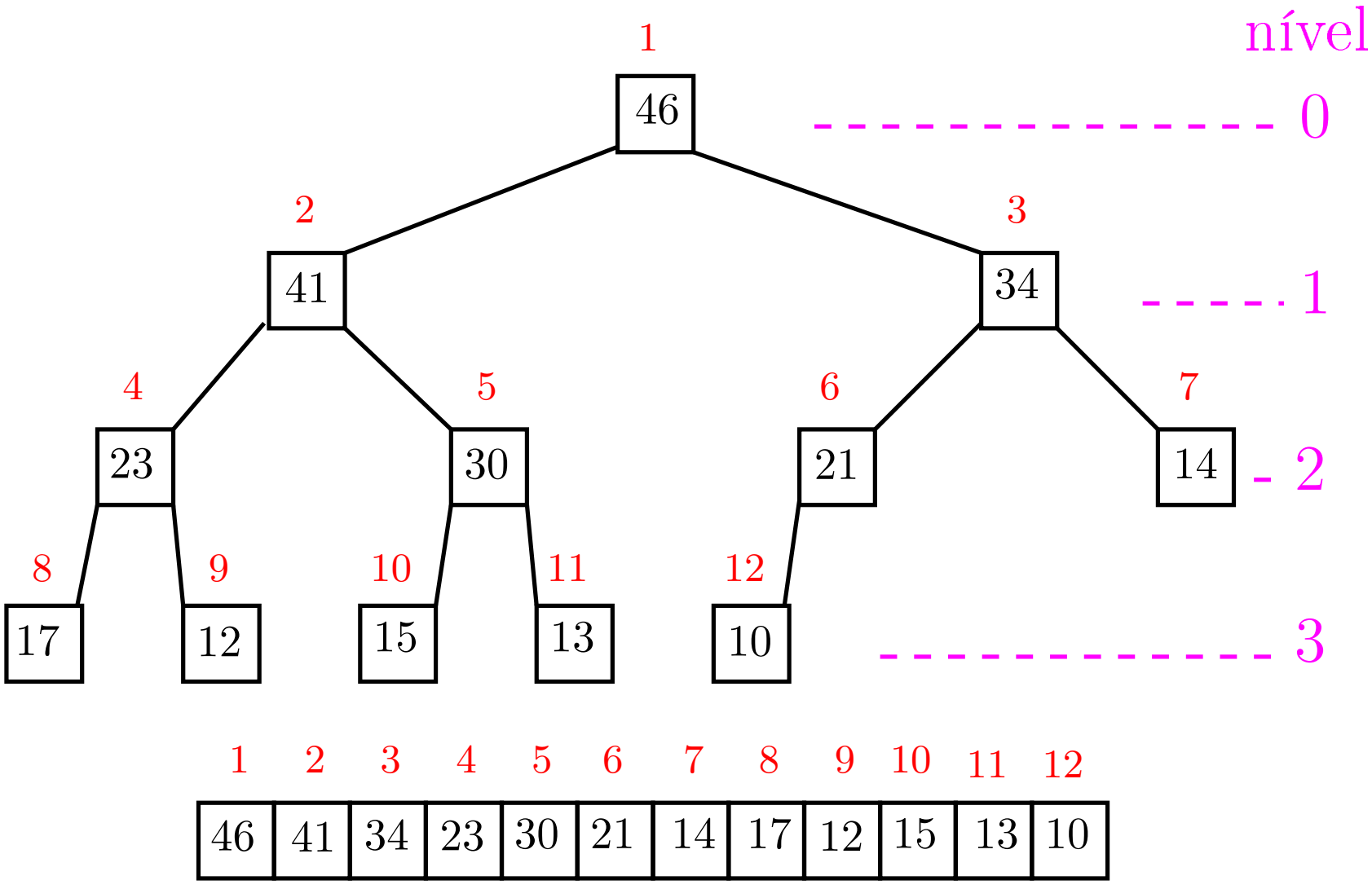
Relações invariantes: Em /*B*/ vale que:

(i0) $\texttt{v}[\texttt{i}+1\,..\,\texttt{n}]$ é crescente;

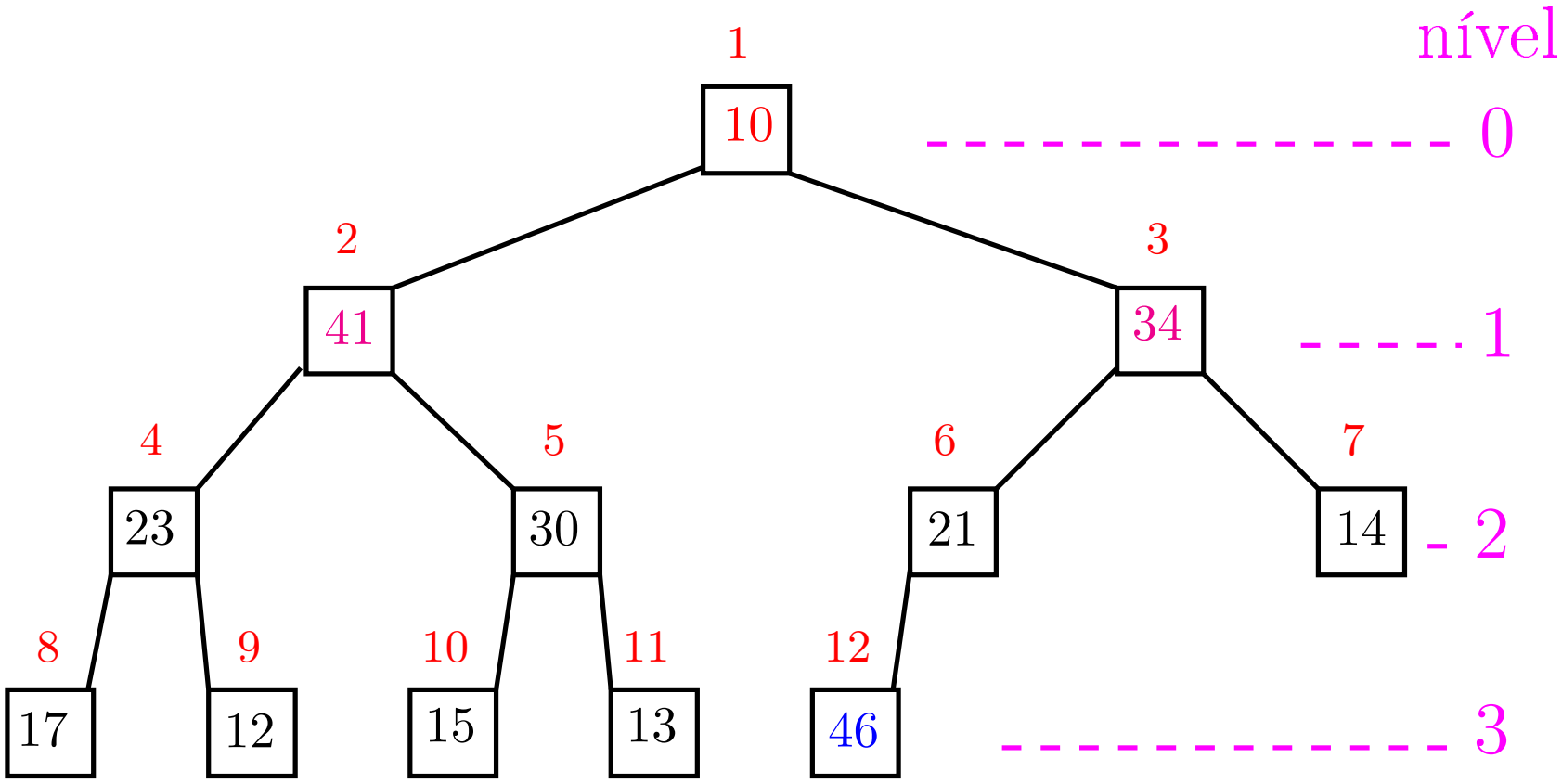
(i1) $\texttt{v}[1\,..\,\texttt{i}] \leq \texttt{v}[\texttt{i}+1]$;



Heapsort

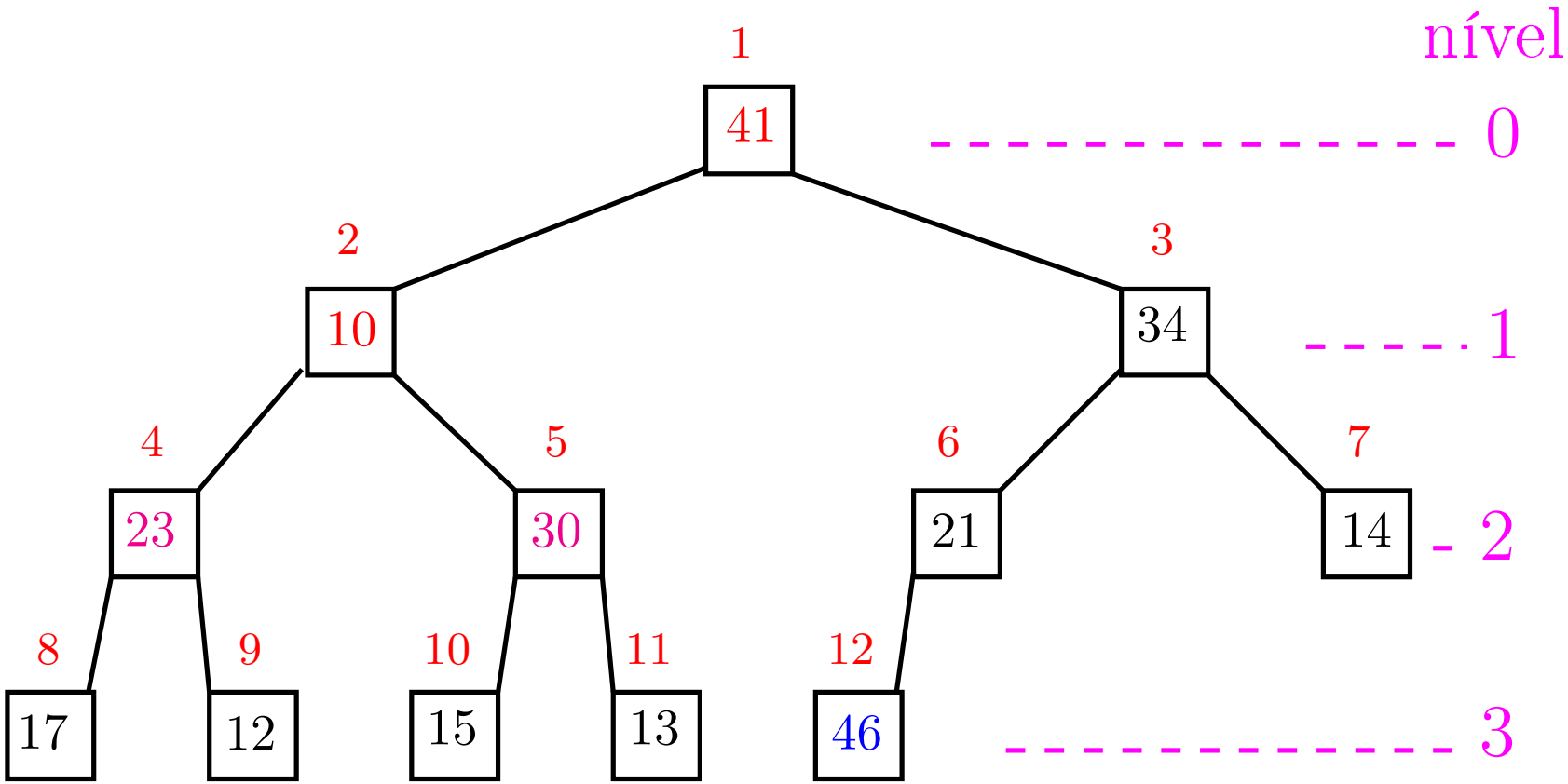

nível
1
46
0
2
41
3
34
1
4
23
5
30
6
21
7
14
2
8
17
9
12
10
15
11
13
12
10
3
1 2 3 4 5 6 7 8 9 10 11 12
46 41 34 23 30 21 14 17 12 15 13 10

Heapsort



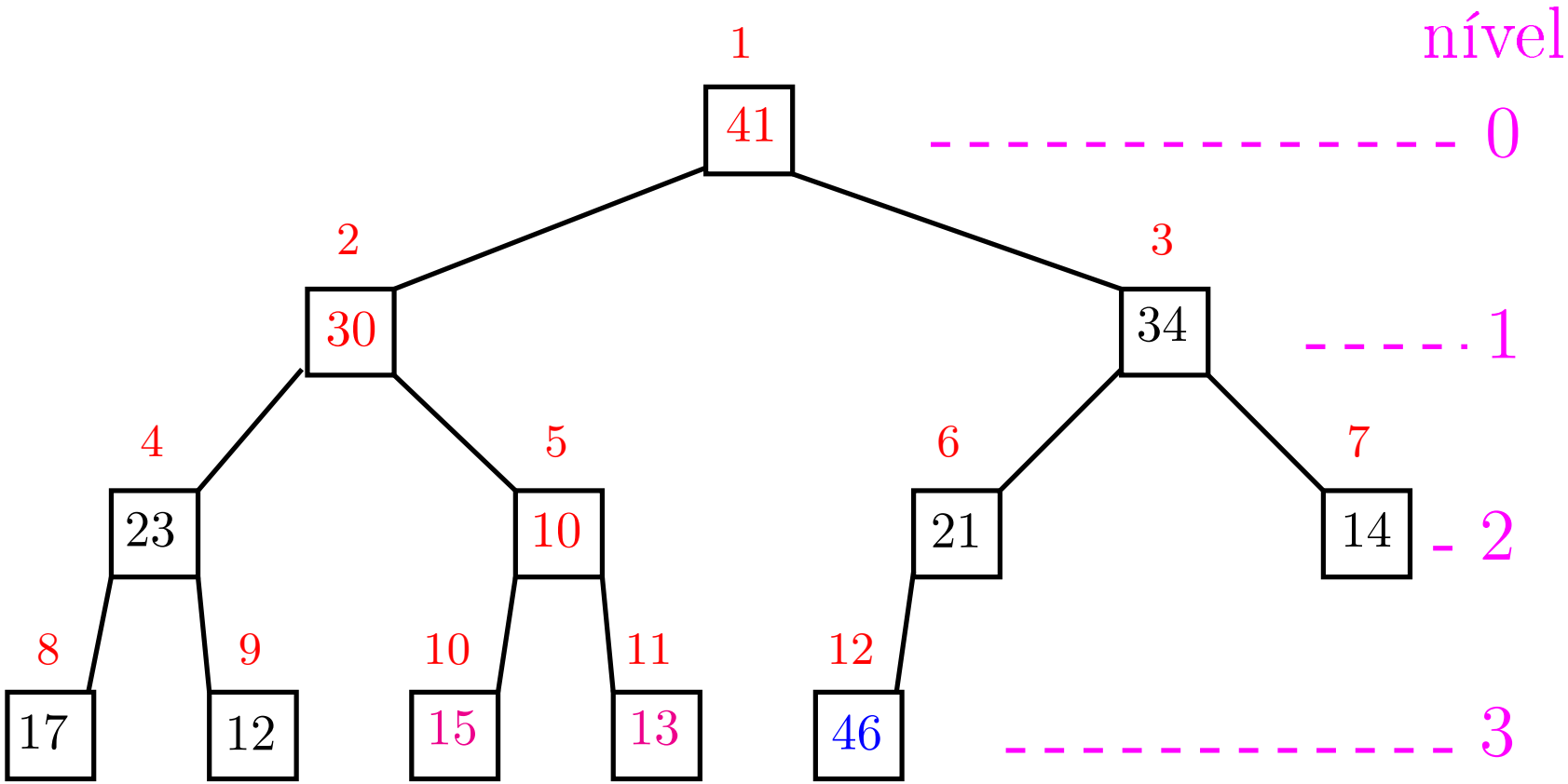
1	2	3	4	5	6	7	8	9	10	11	12
10	41	34	23	30	21	14	17	12	15	13	46

Heapsort



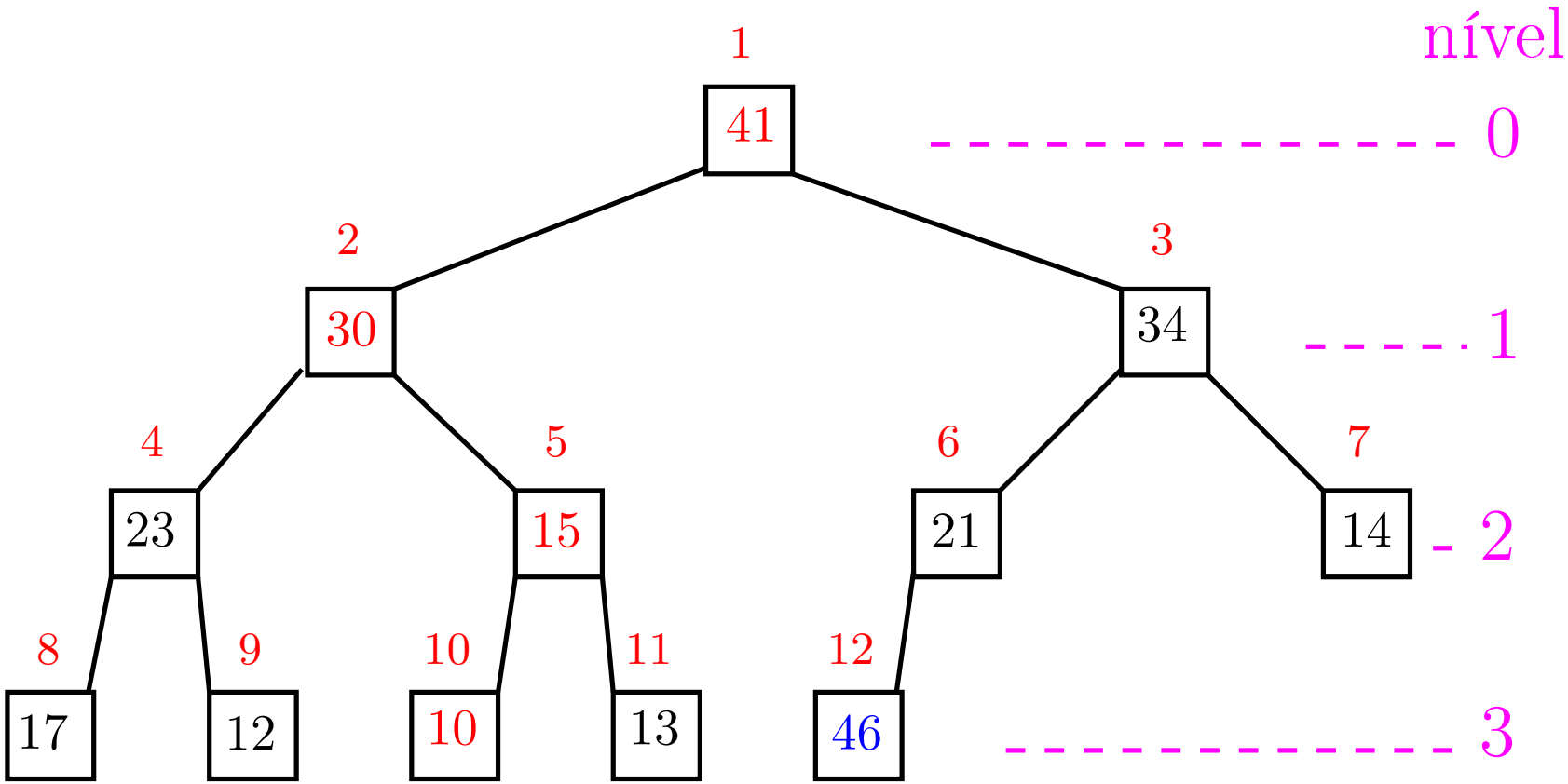
1	2	3	4	5	6	7	8	9	10	11	12
41	10	34	23	30	21	14	17	12	15	13	46

Heapsort



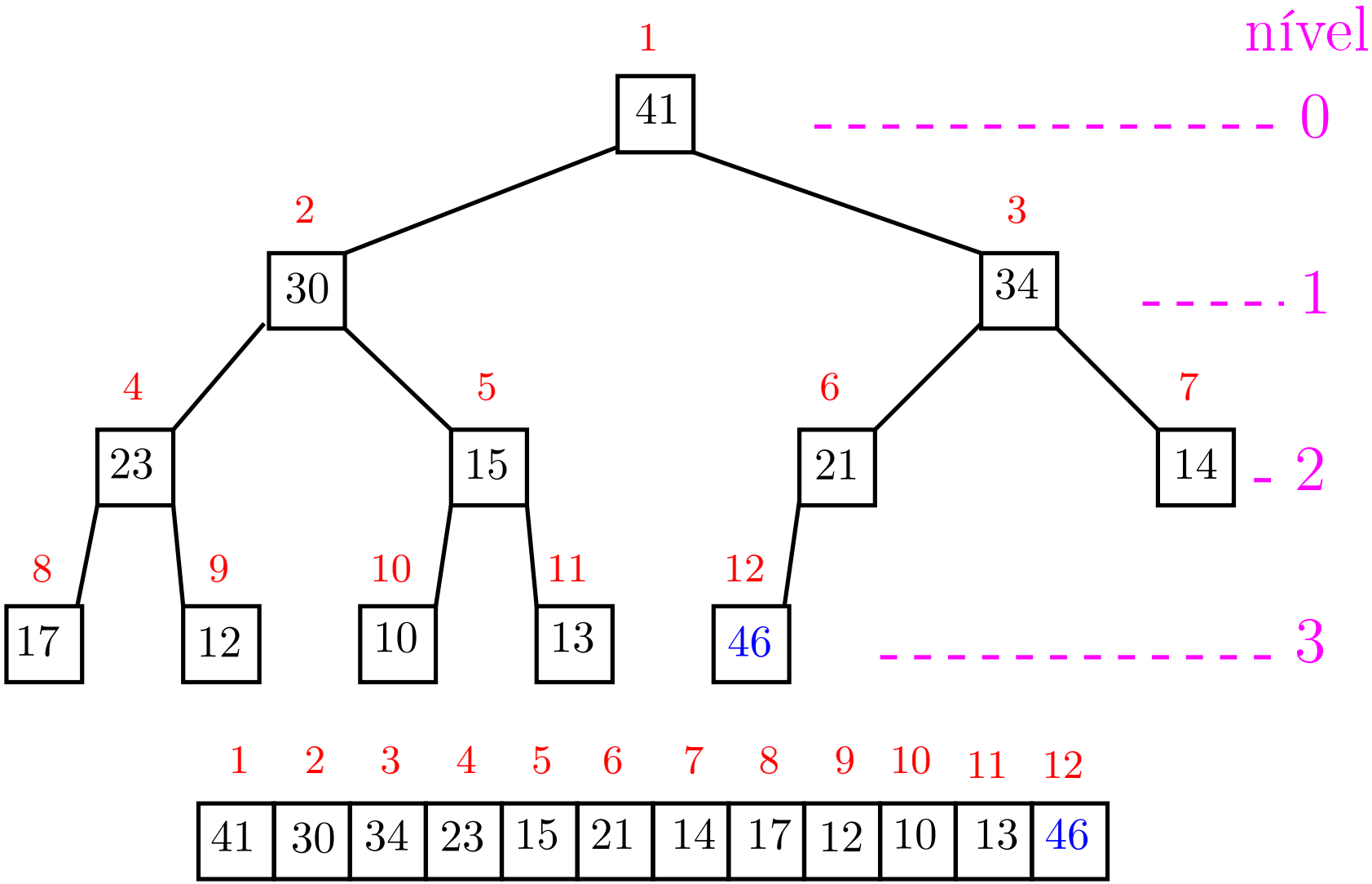
1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	10	21	14	17	12	15	13	46

Heapsort

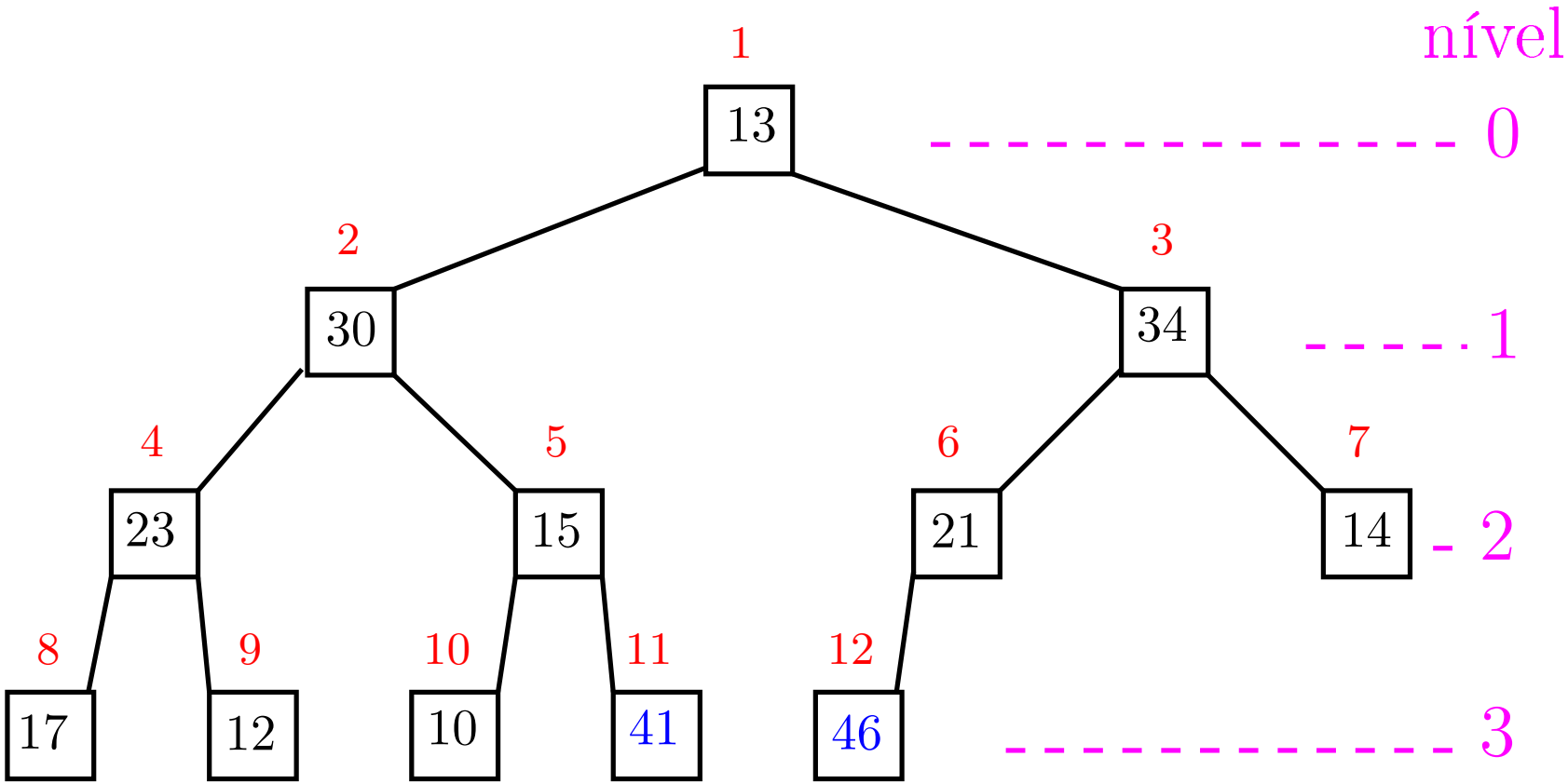


1	2	3	4	5	6	7	8	9	10	11	12
41	30	34	23	15	21	14	17	12	10	13	46

Heapsort

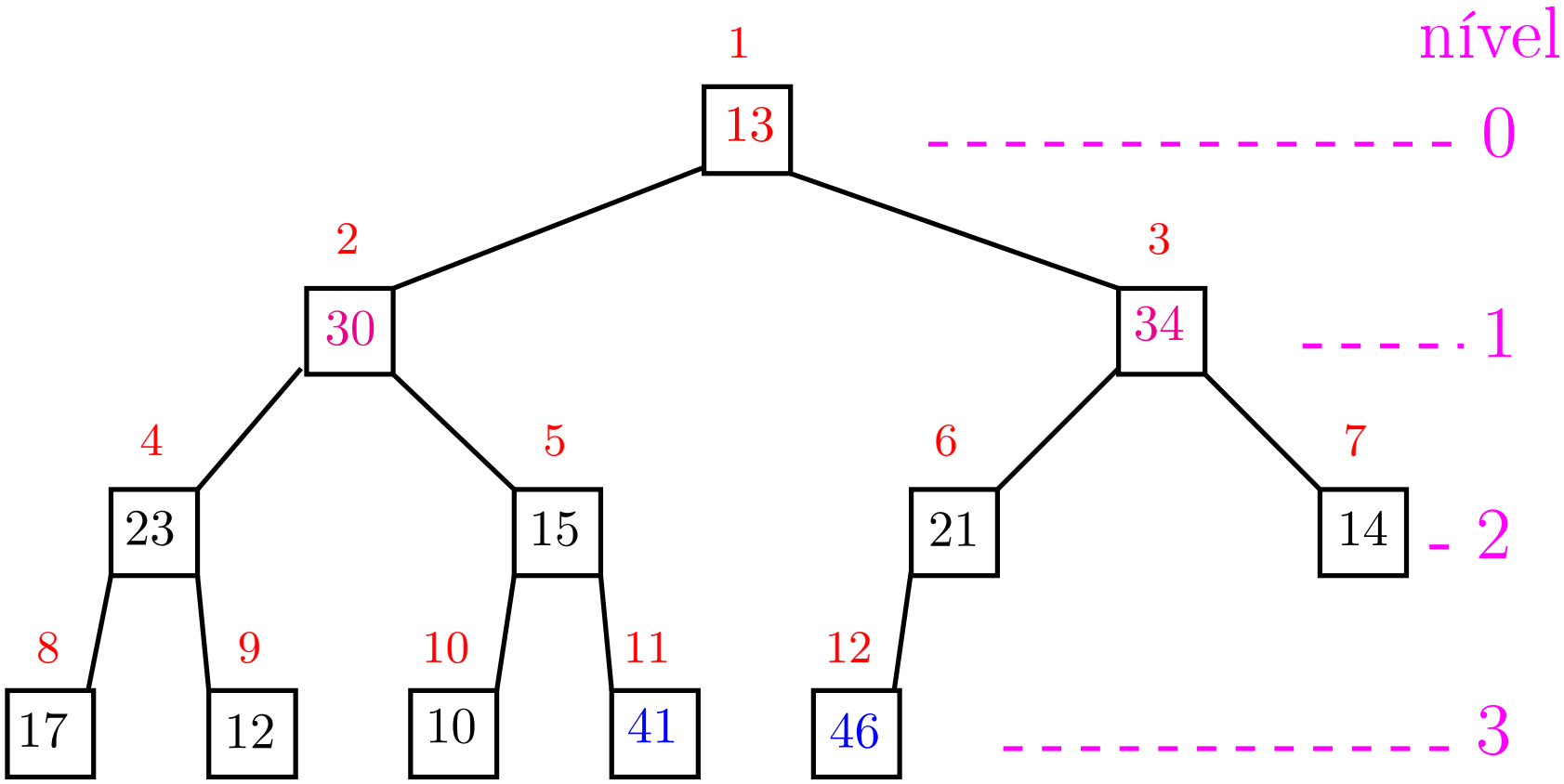

nível
1
41
0
2
30
3
34
1
4
23
5
15
6
21
7
14
2
8
17
9
12
10
10
11
13
12
46
3
1 2 3 4 5 6 7 8 9 10 11 12
41 30 34 23 15 21 14 17 12 10 13 46

Heapsort



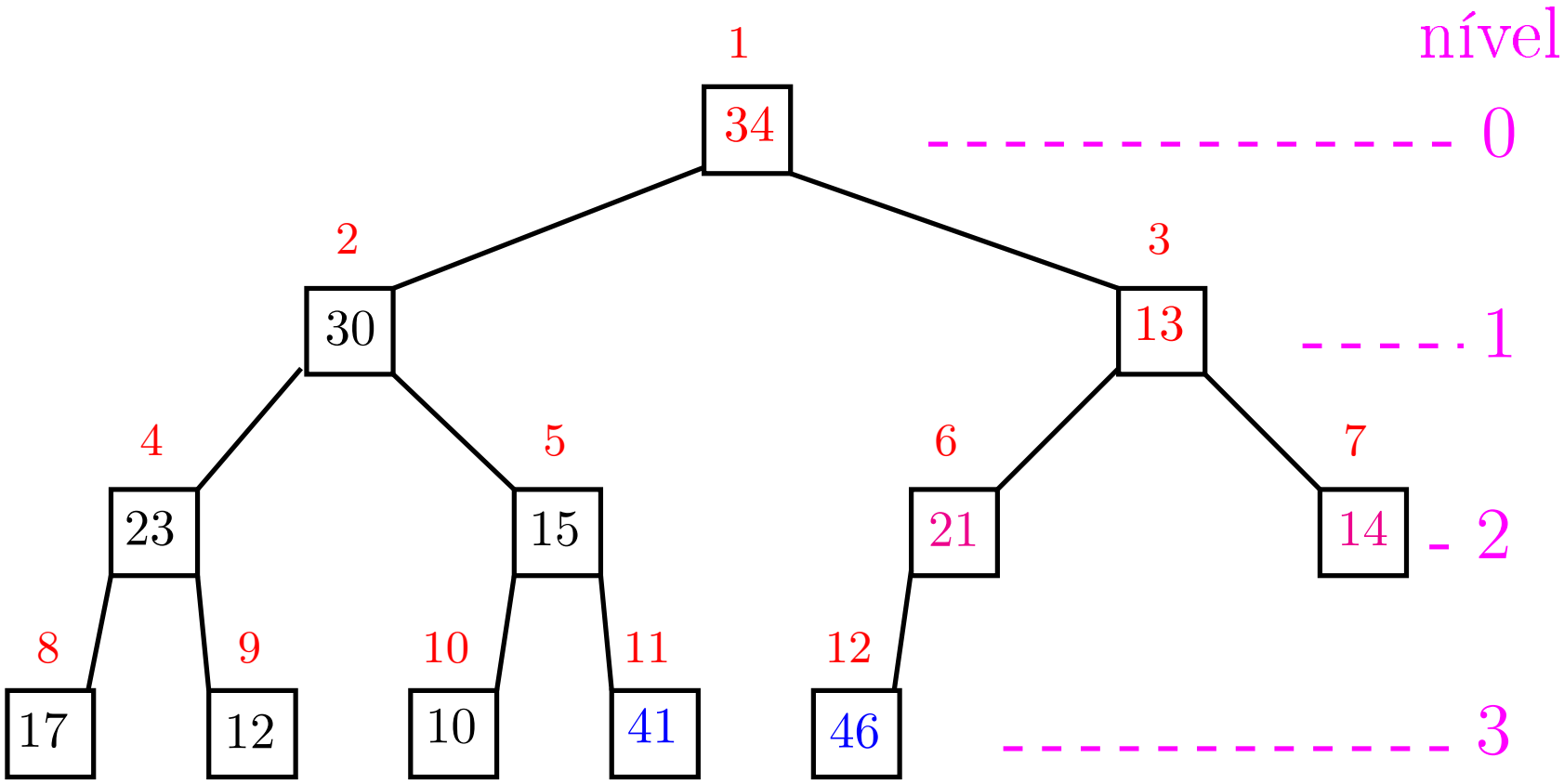
1	2	3	4	5	6	7	8	9	10	11	12
13	30	34	23	15	21	14	17	12	10	41	46

Heapsort



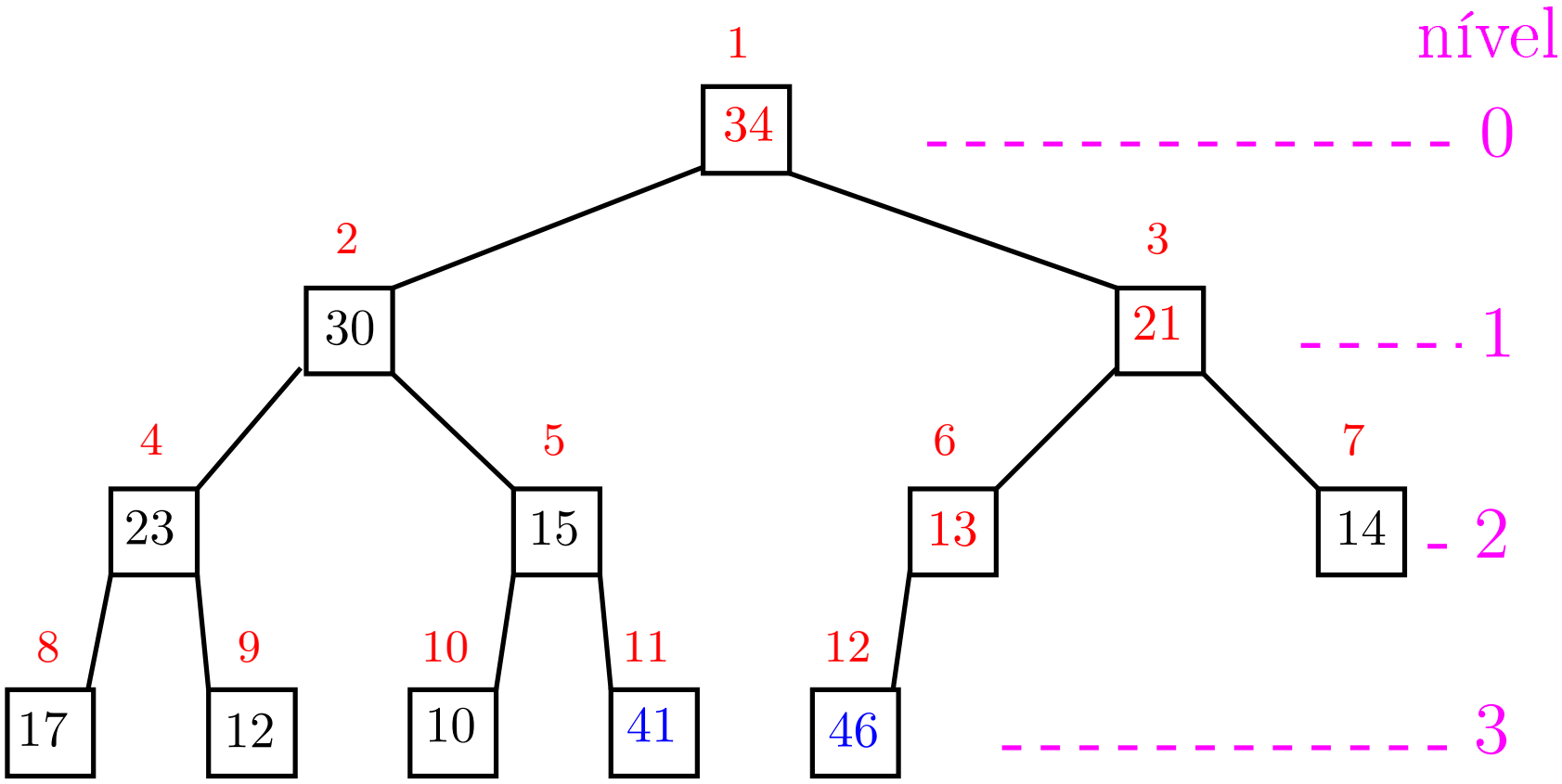
1	2	3	4	5	6	7	8	9	10	11	12
13	30	34	23	15	21	14	17	12	10	41	46

Heapsort



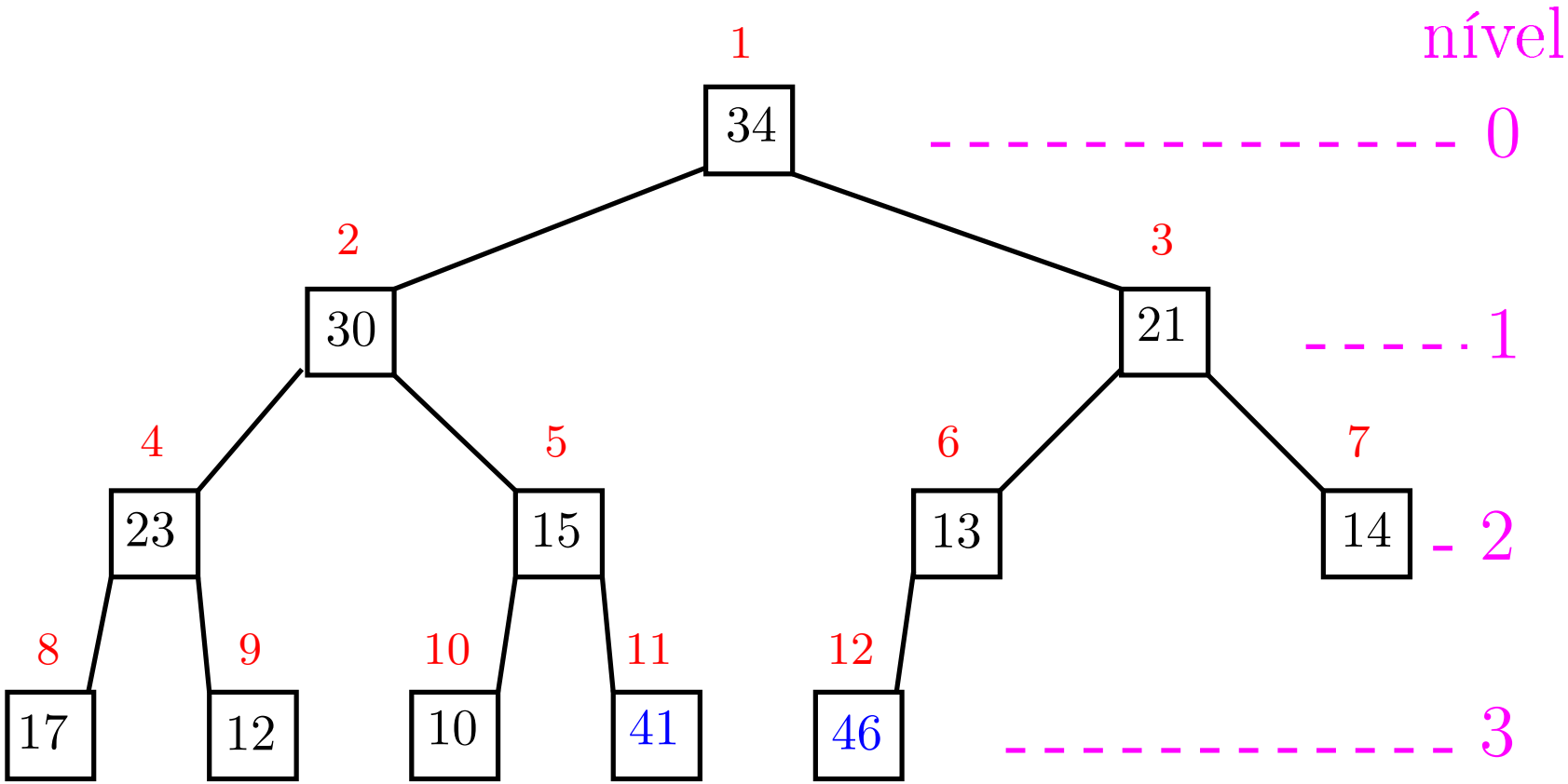
1	2	3	4	5	6	7	8	9	10	11	12
34	30	13	23	15	21	14	17	12	10	41	46

Heapsort



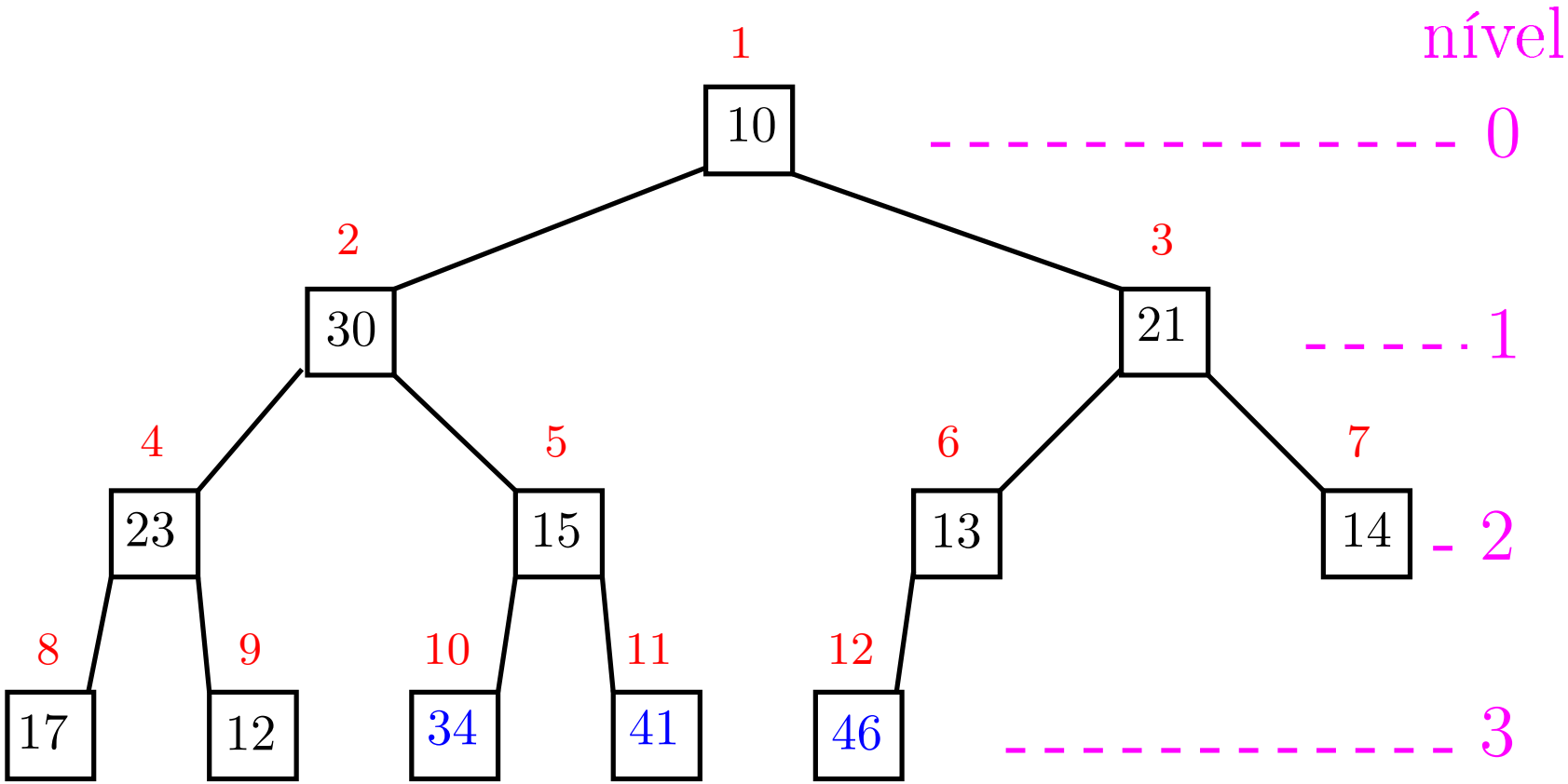
1	2	3	4	5	6	7	8	9	10	11	12
34	30	21	23	15	13	14	17	12	10	41	46

Heapsort



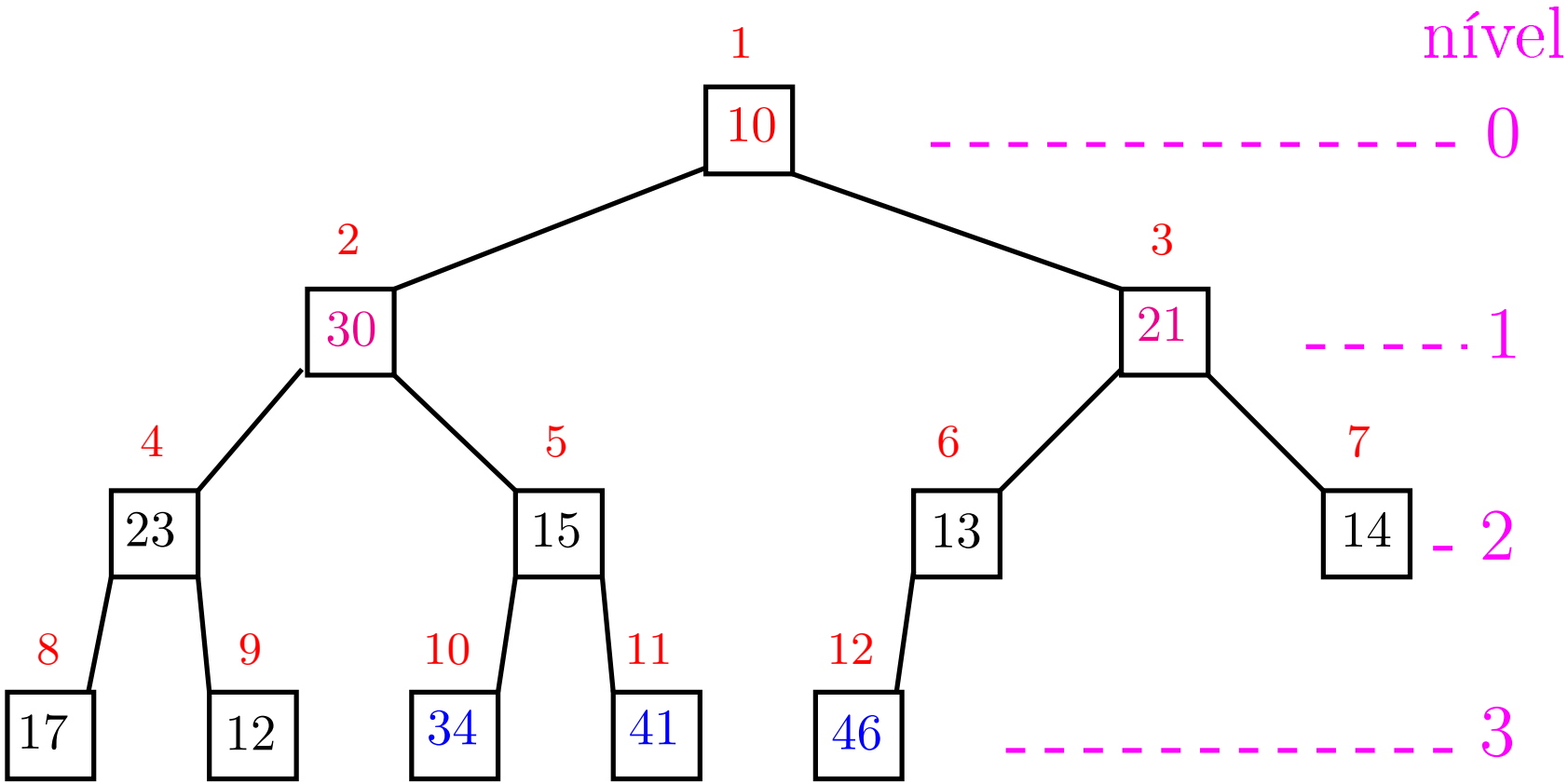
1	2	3	4	5	6	7	8	9	10	11	12
34	30	21	23	15	13	14	17	12	10	41	46

Heapsort



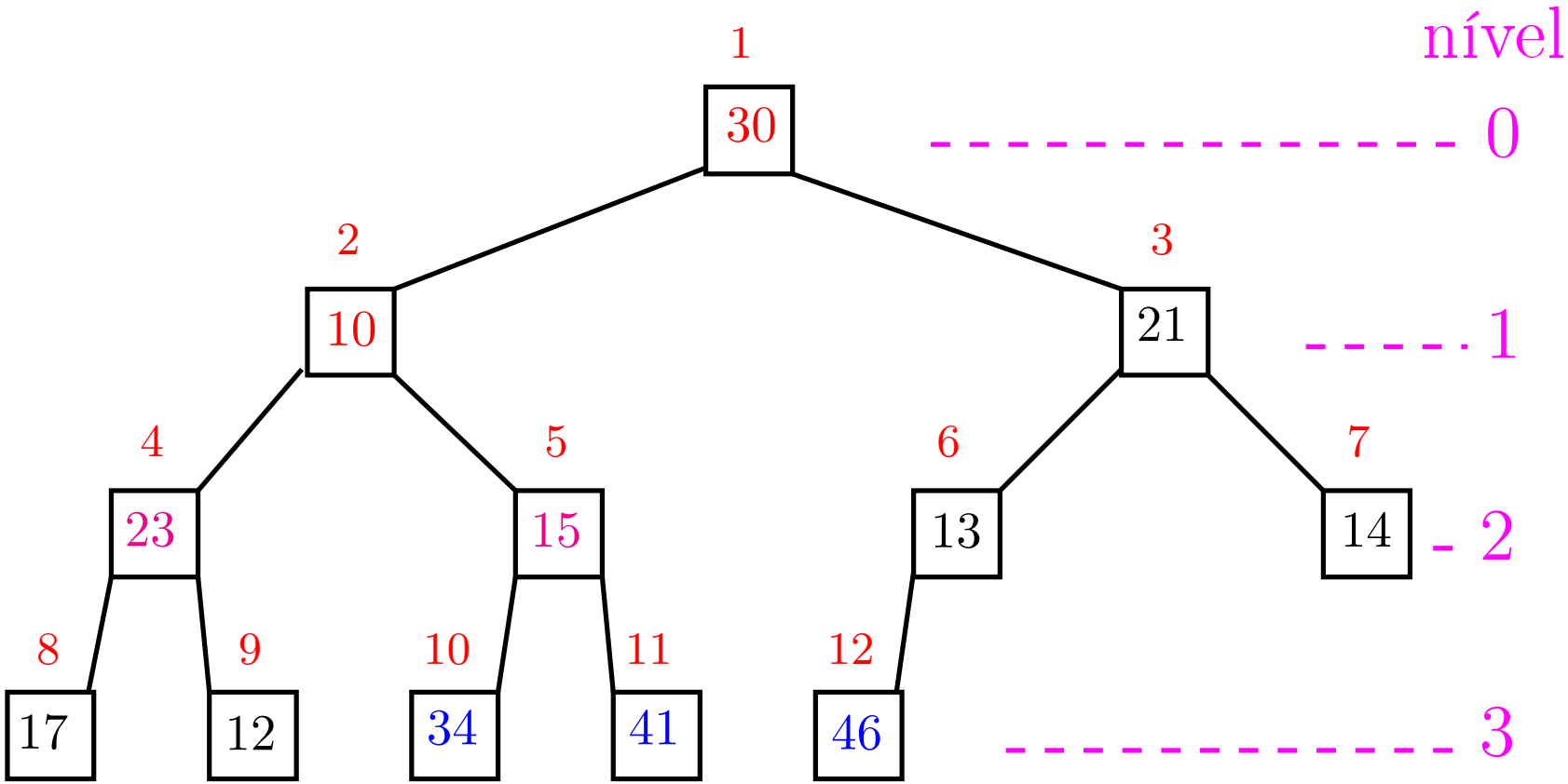
1	2	3	4	5	6	7	8	9	10	11	12
10	30	21	23	15	13	14	17	12	34	41	46

Heapsort



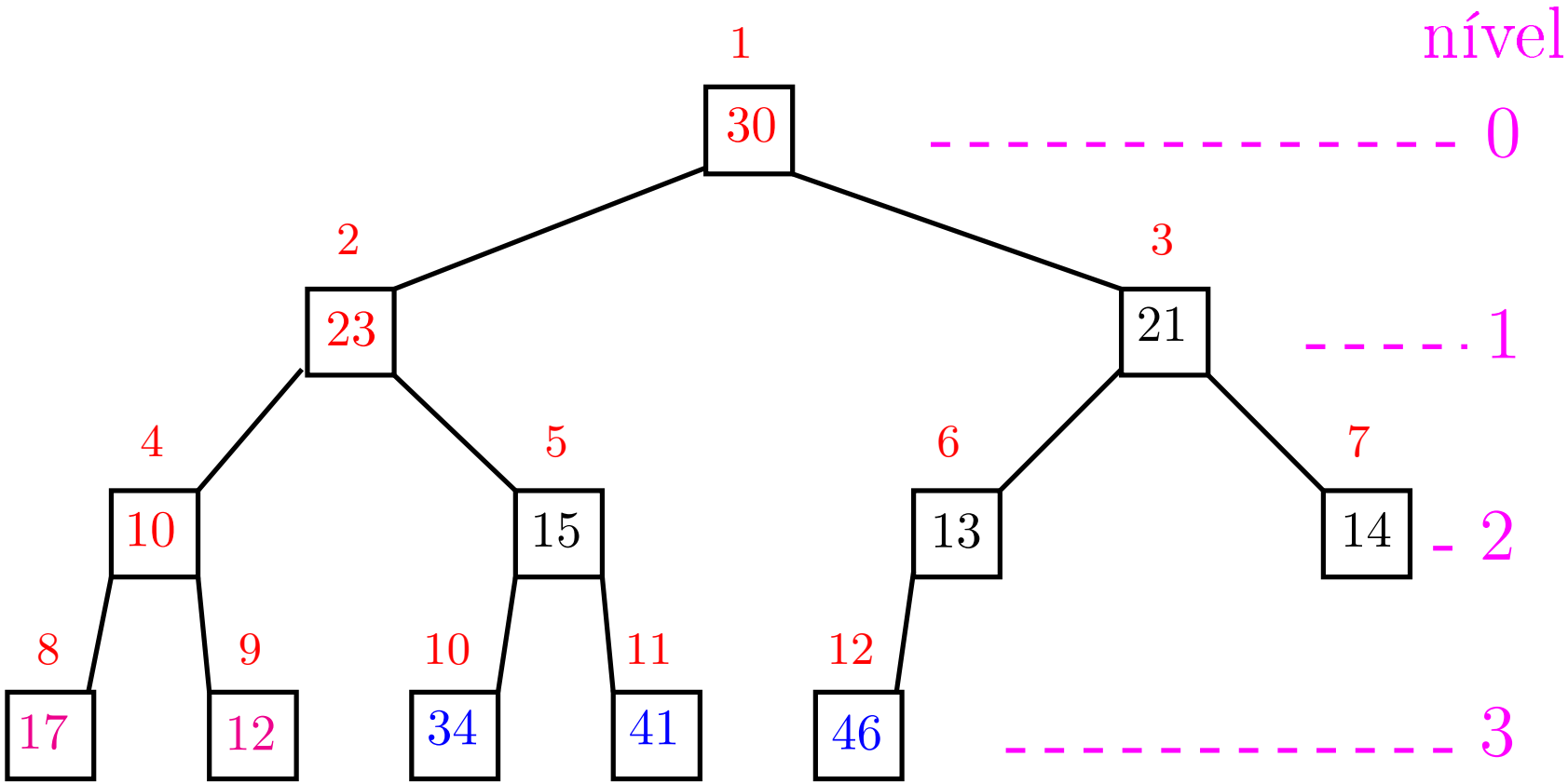
1	2	3	4	5	6	7	8	9	10	11	12
10	30	21	23	15	13	14	17	12	34	41	46

Heapsort



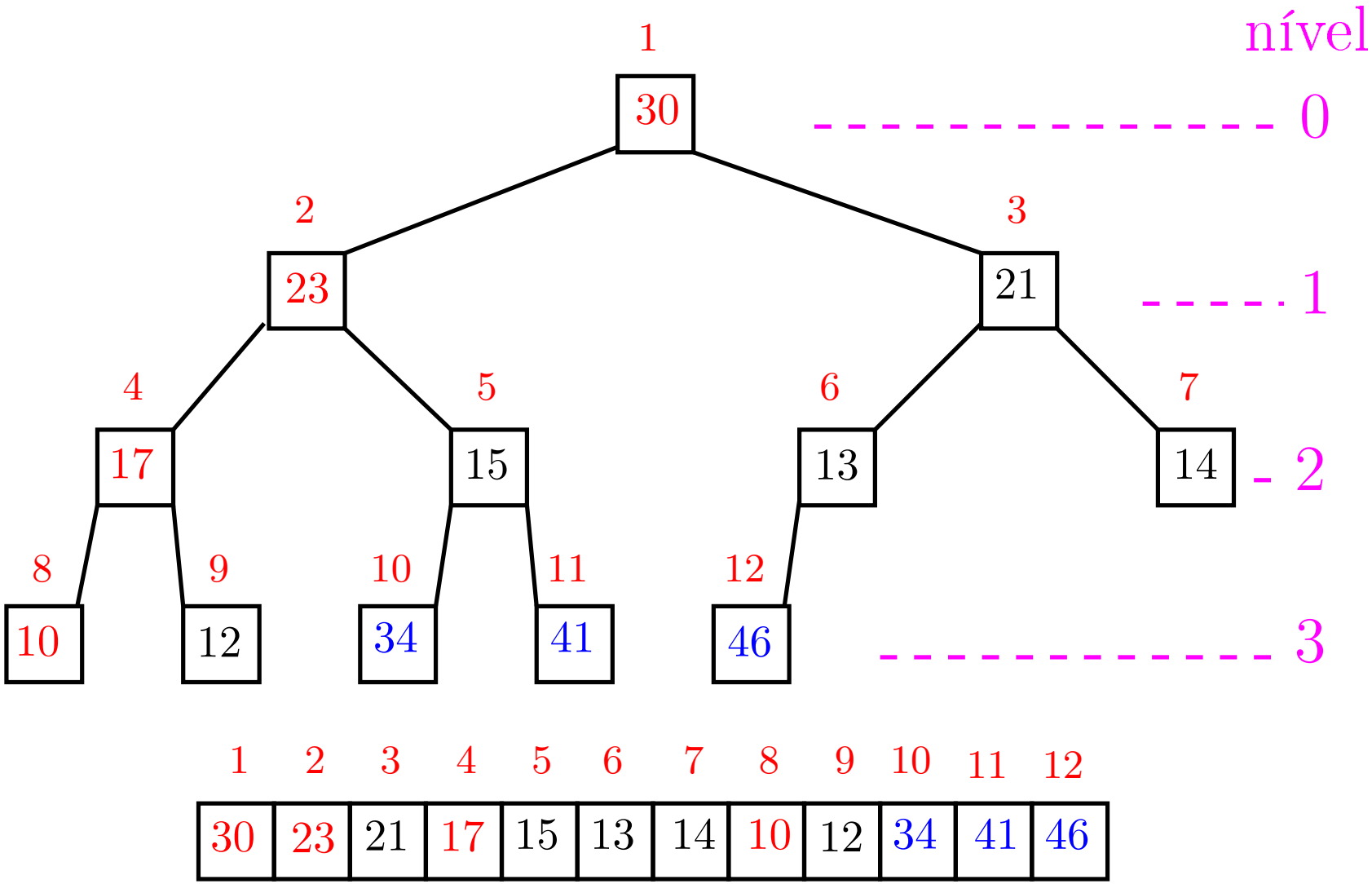
1	2	3	4	5	6	7	8	9	10	11	12
30	10	21	23	15	13	14	17	12	34	41	46

Heapsort

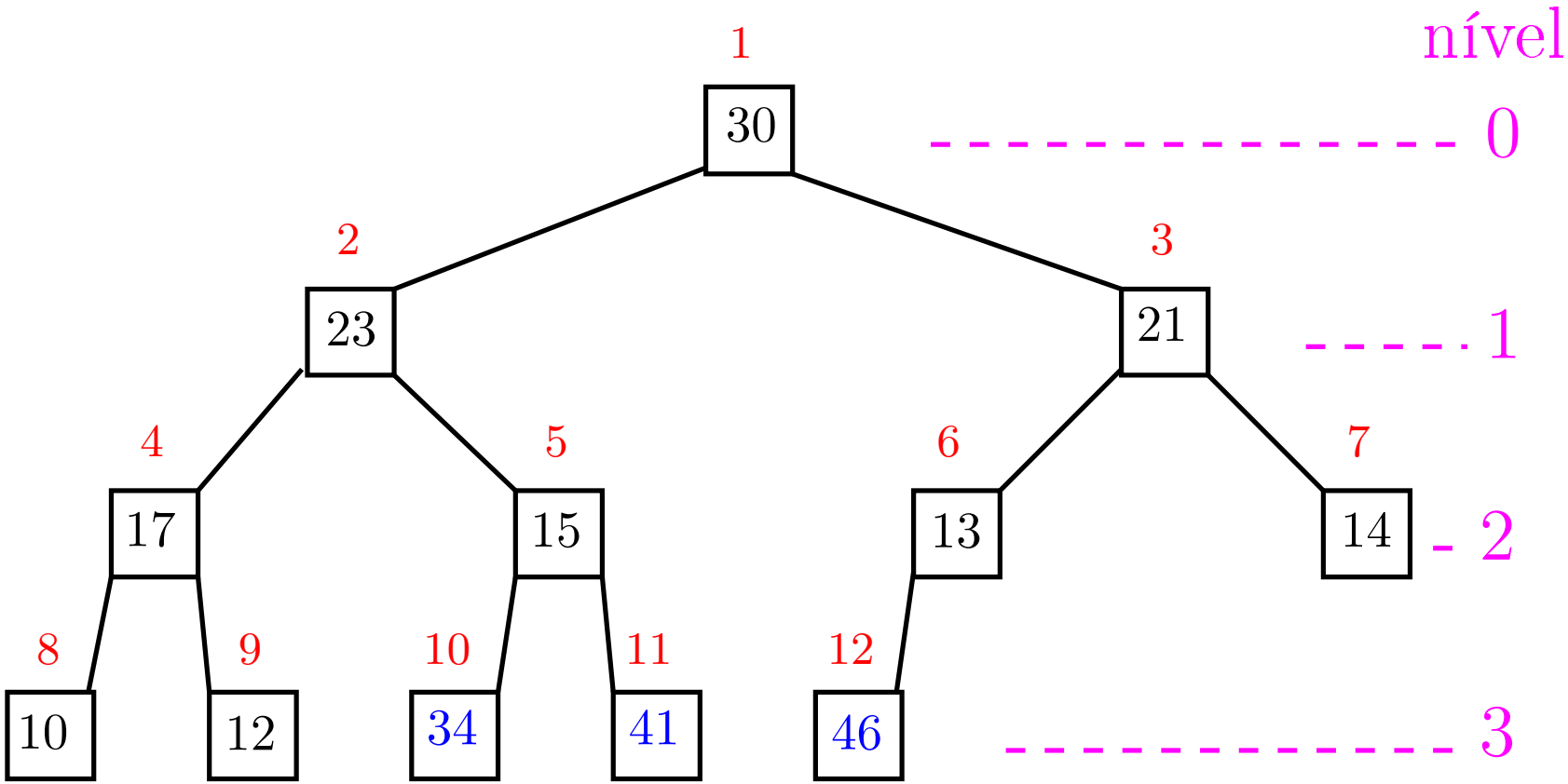


1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	10	15	13	14	17	12	34	41	46

Heapsort

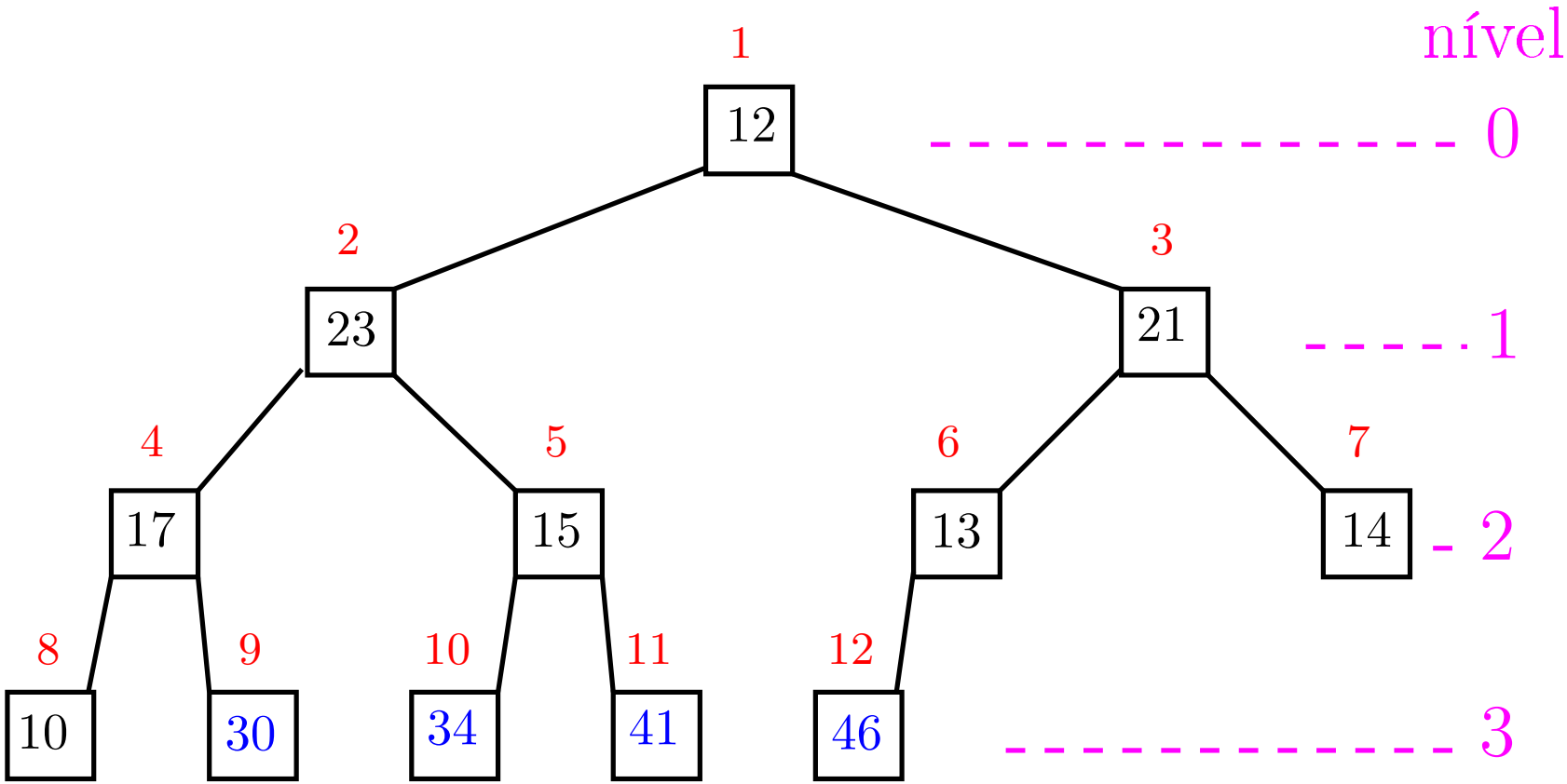

nível
1
30
0
2
23
3
21
1
4
17
5
15
6
13
7
14
2
8
10
9
12
10
34
11
41
12
46
3
1 2 3 4 5 6 7 8 9 10 11 12
30 23 21 17 15 13 14 10 12 34 41 46

Heapsort



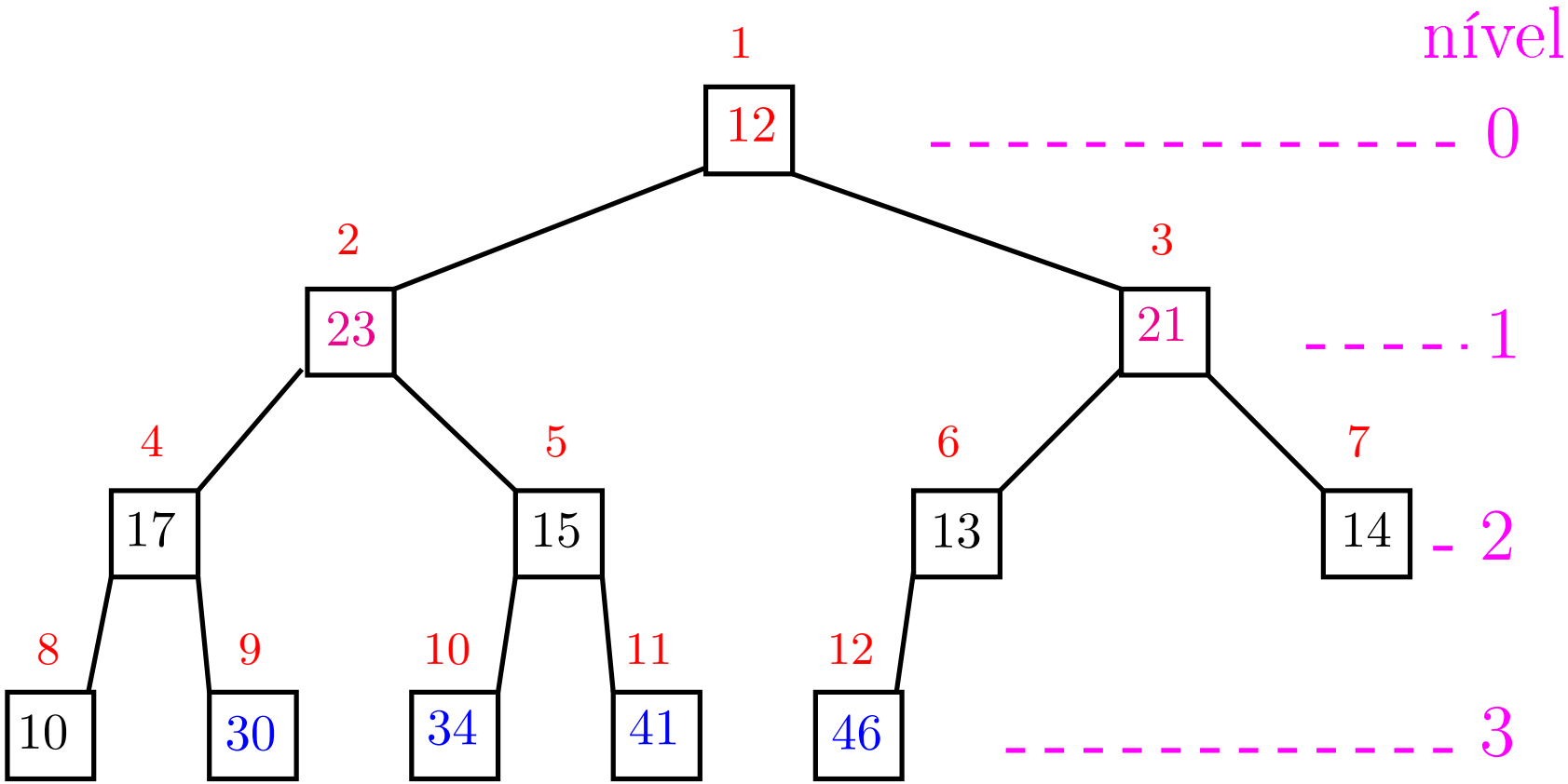
1	2	3	4	5	6	7	8	9	10	11	12
30	23	21	17	15	13	14	10	12	34	41	46

Heapsort



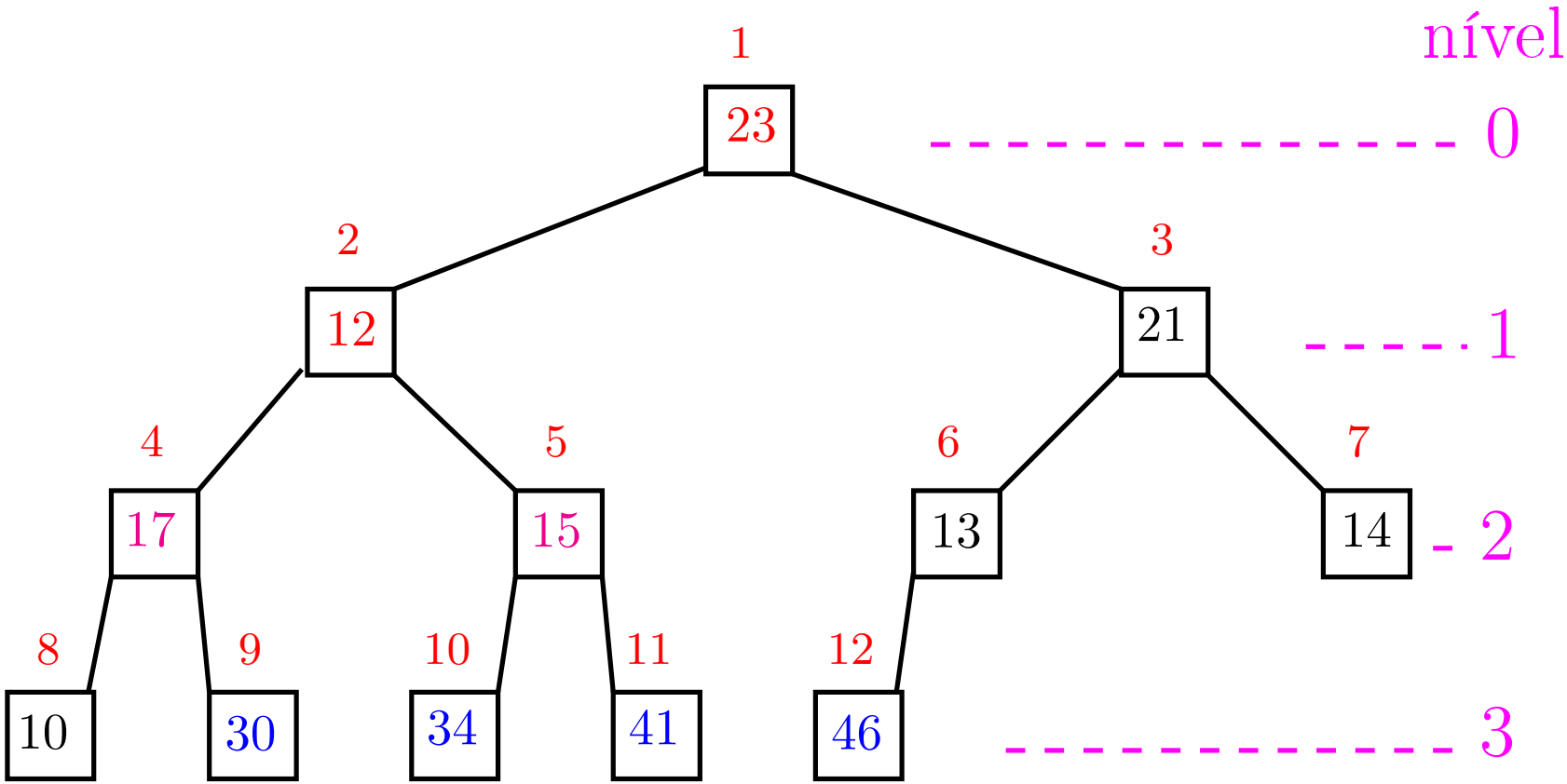
1	2	3	4	5	6	7	8	9	10	11	12
12	23	21	17	15	13	14	10	30	34	41	46

Heapsort



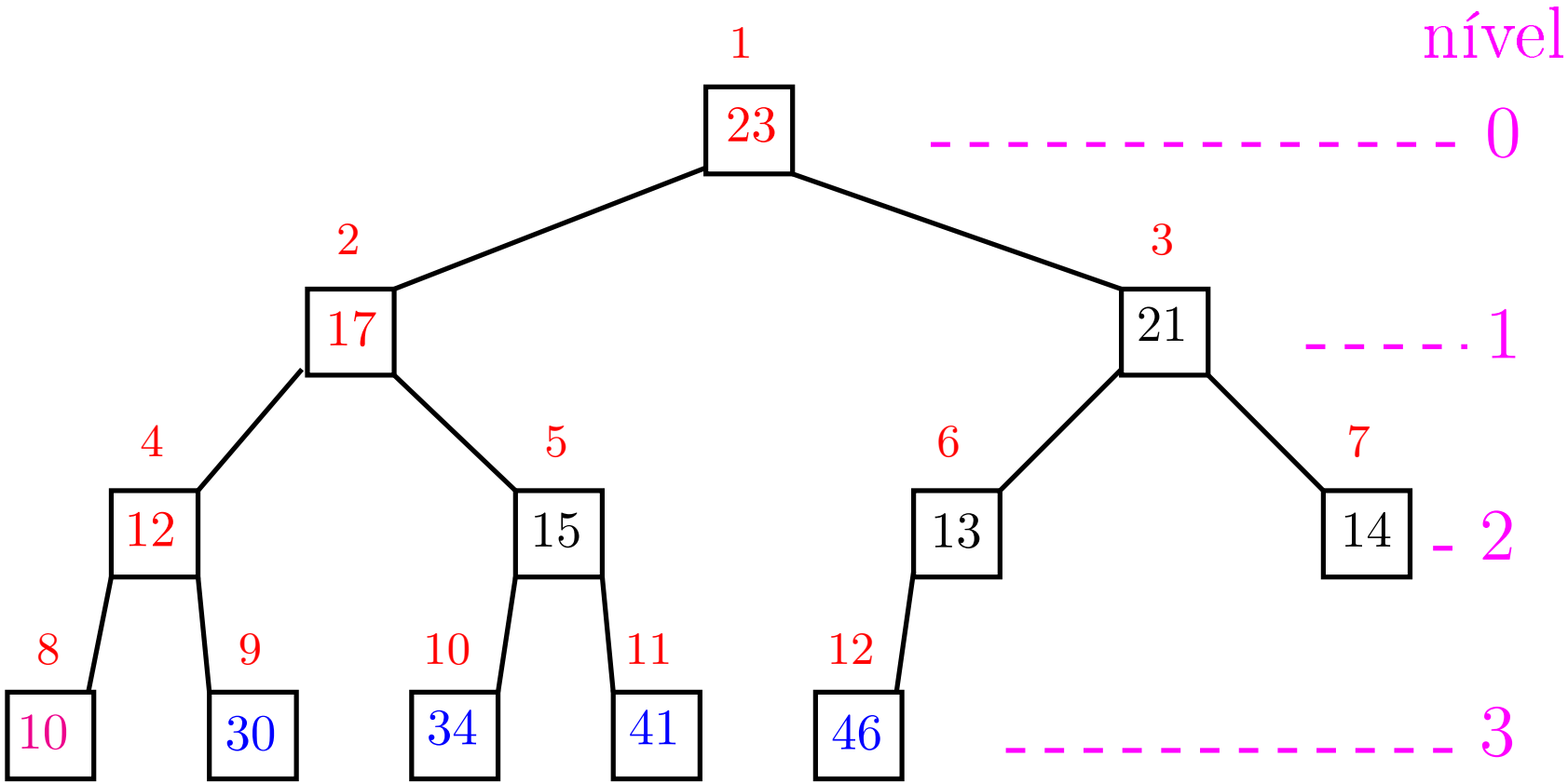
1	2	3	4	5	6	7	8	9	10	11	12
12	23	21	17	15	13	14	10	30	34	41	46

Heapsort



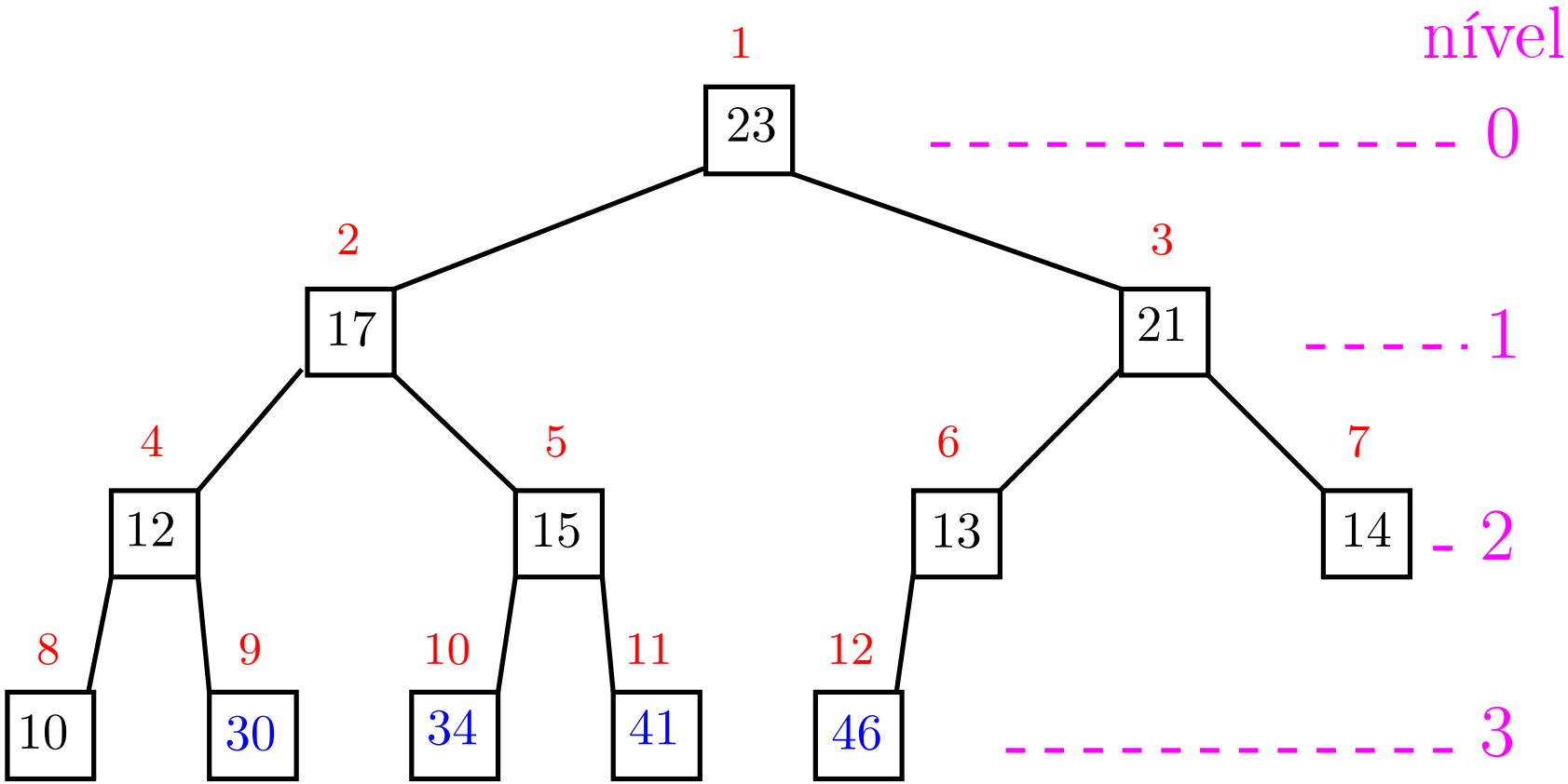
1	2	3	4	5	6	7	8	9	10	11	12
23	12	21	17	15	13	14	10	30	34	41	46

Heapsort



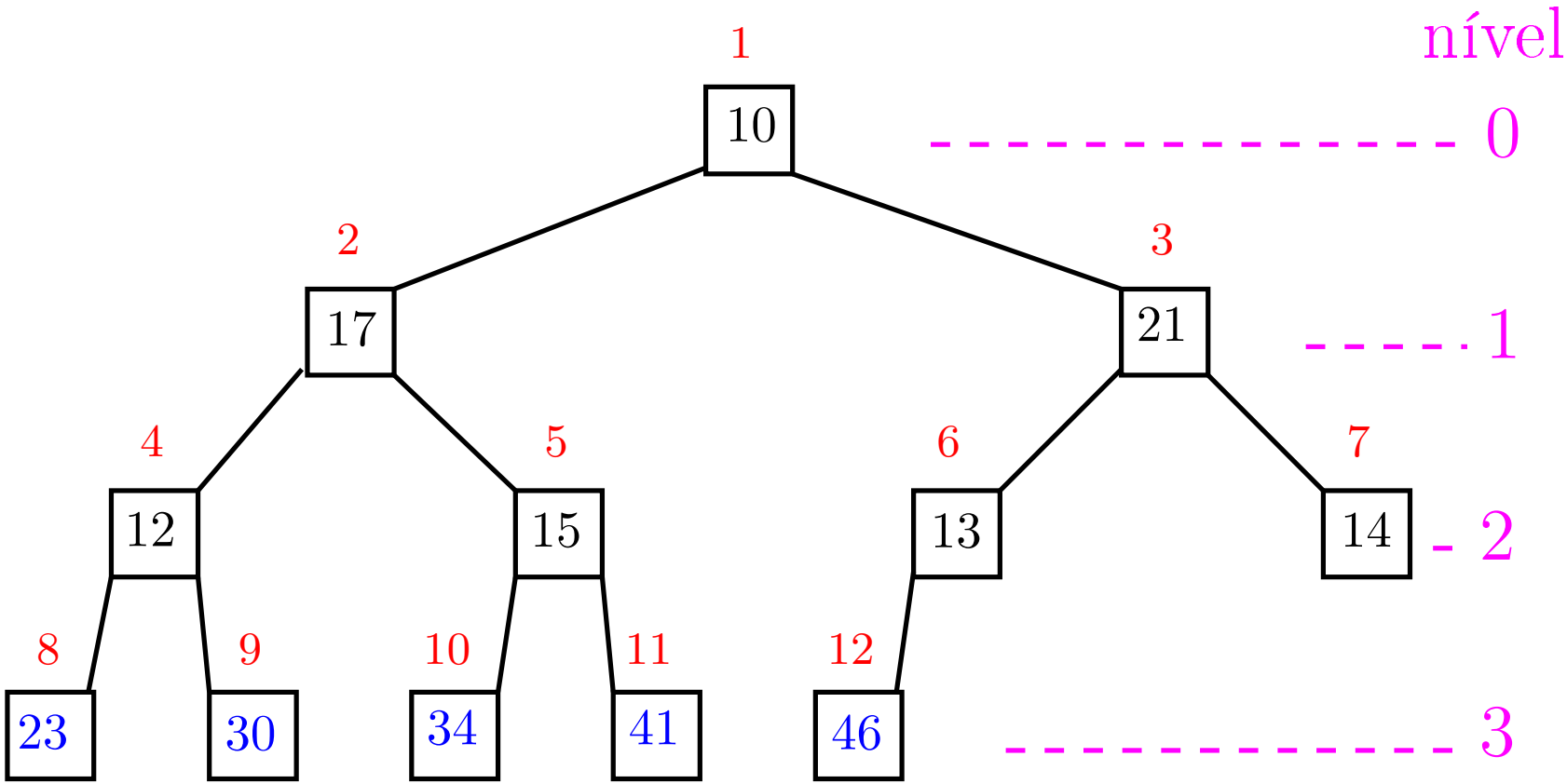
1	2	3	4	5	6	7	8	9	10	11	12
23	17	21	12	15	13	14	10	30	34	41	46

Heapsort



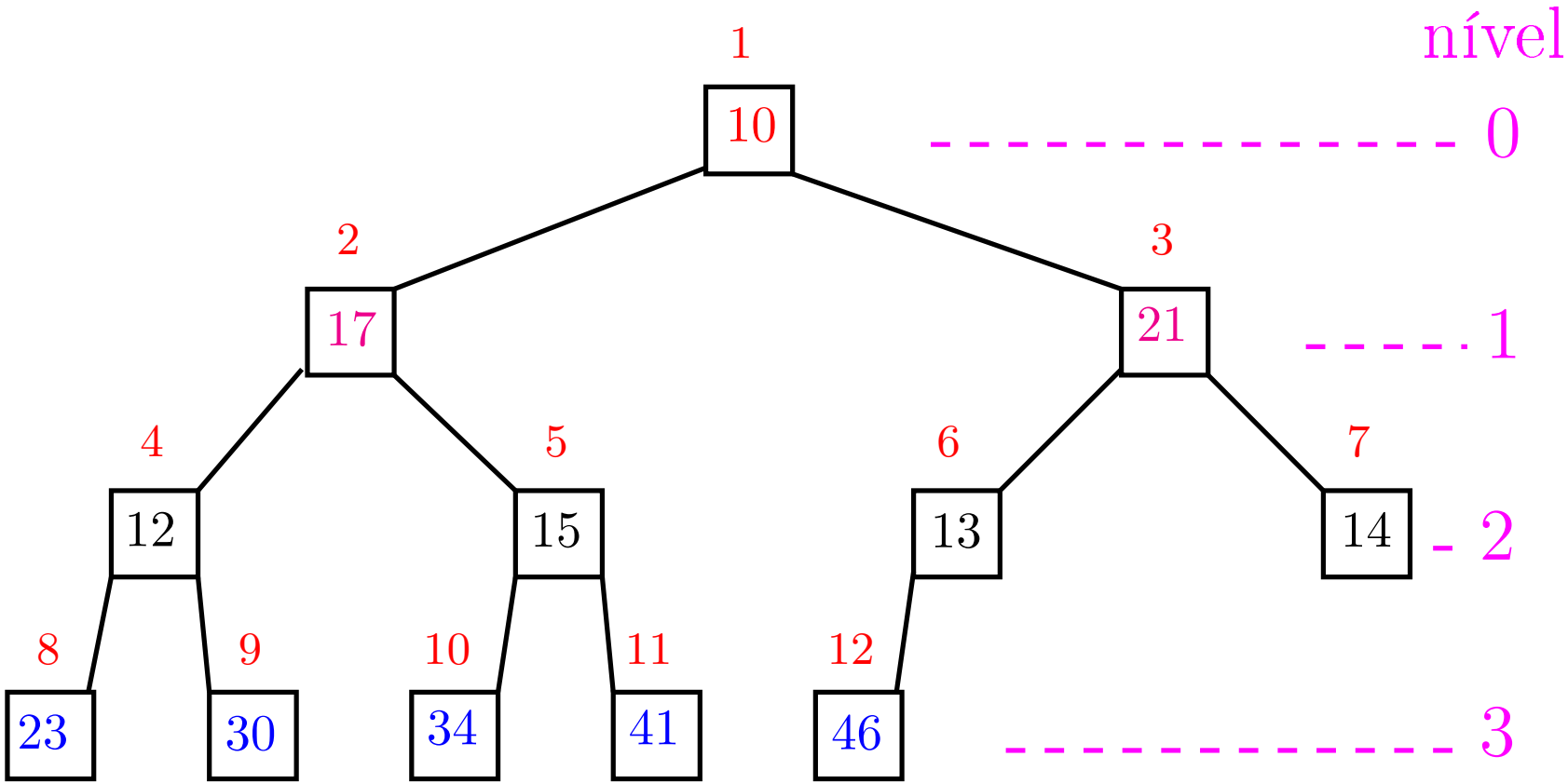
1	2	3	4	5	6	7	8	9	10	11	12
23	17	21	12	15	13	14	10	30	34	41	46

Heapsort



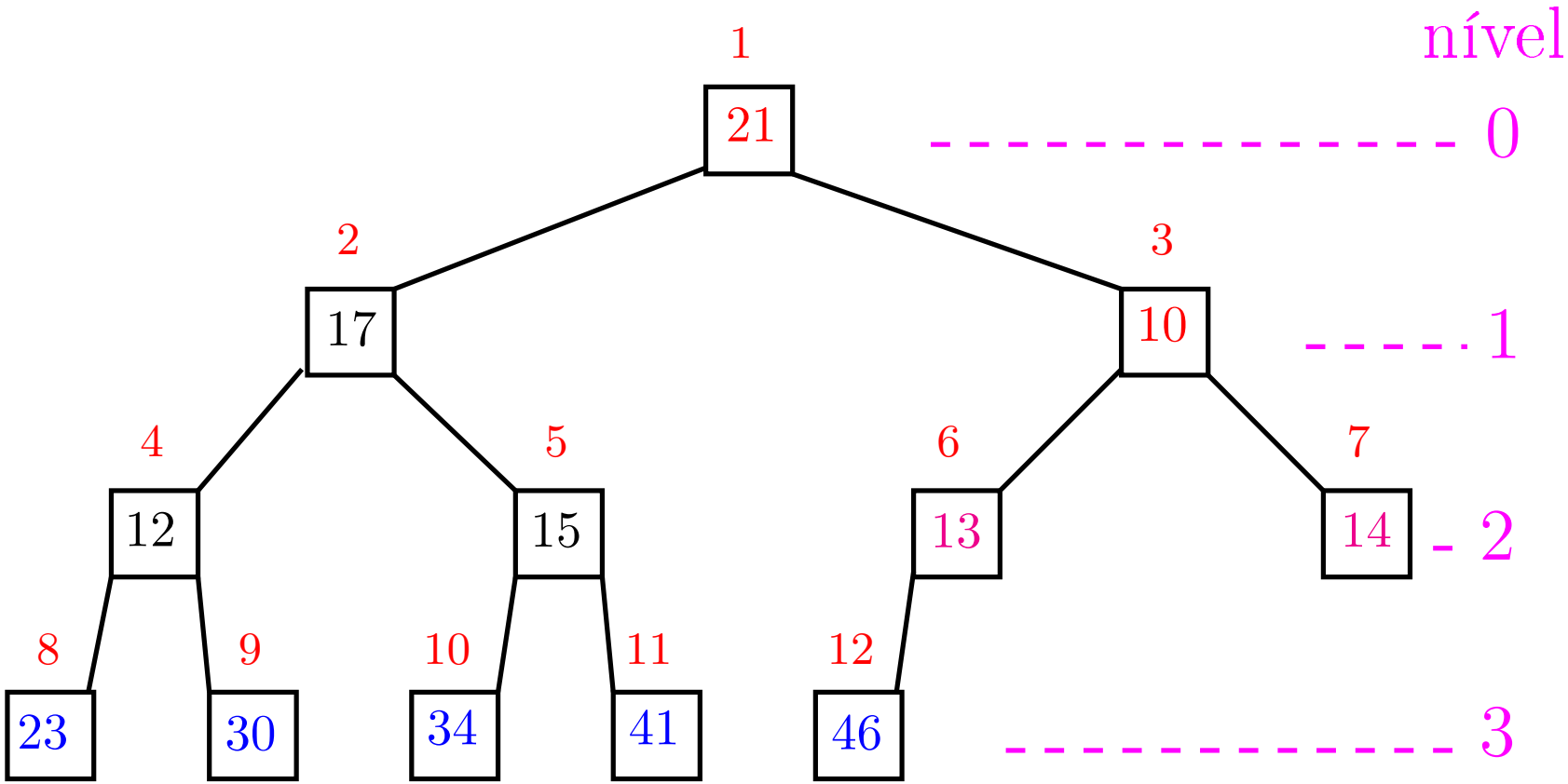
1	2	3	4	5	6	7	8	9	10	11	12
10	17	21	12	15	13	14	23	30	34	41	46

Heapsort



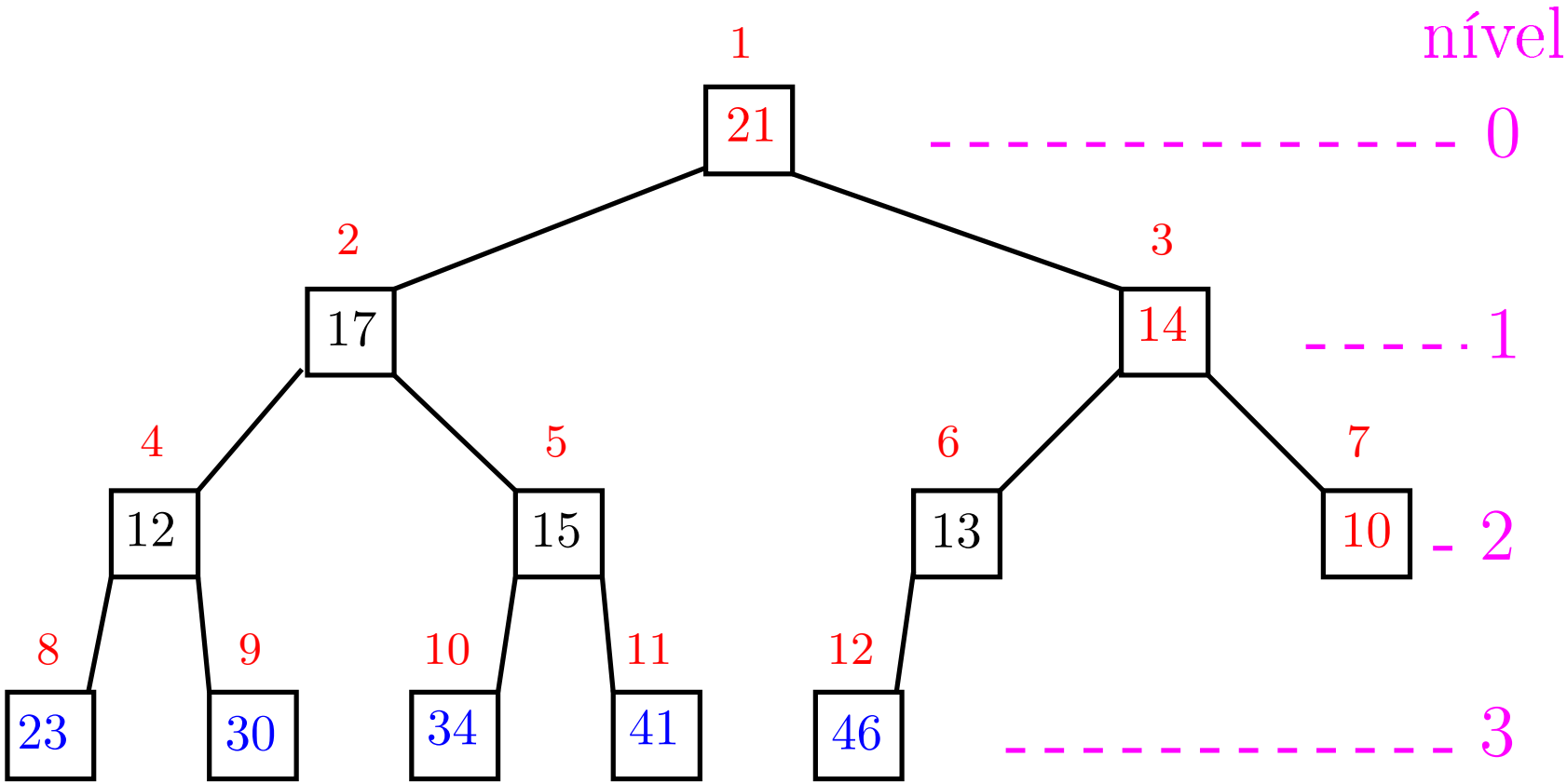
1	2	3	4	5	6	7	8	9	10	11	12
10	17	21	12	15	13	14	23	30	34	41	46

Heapsort



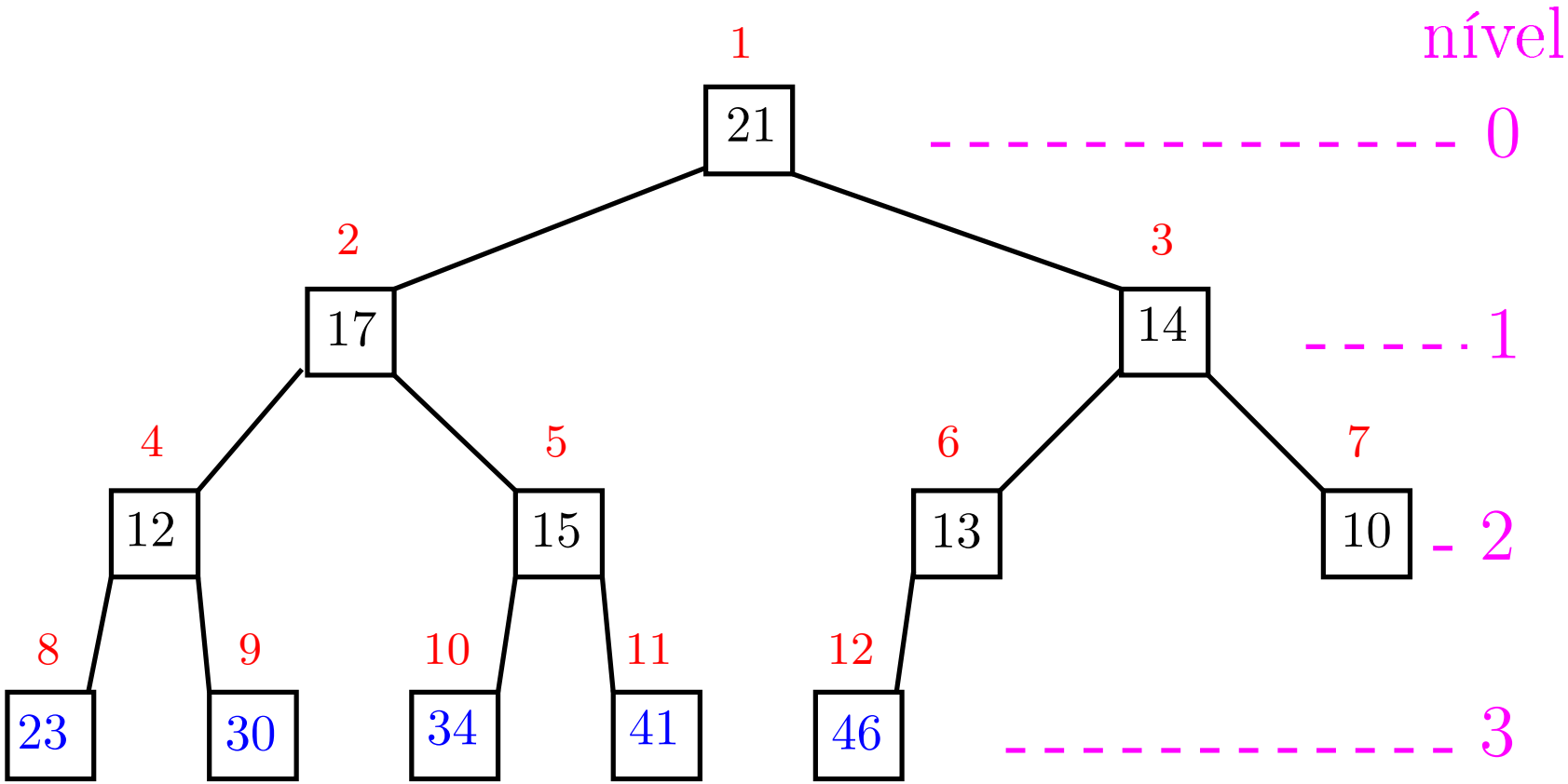
1	2	3	4	5	6	7	8	9	10	11	12
21	17	10	12	15	13	14	23	30	34	41	46

Heapsort



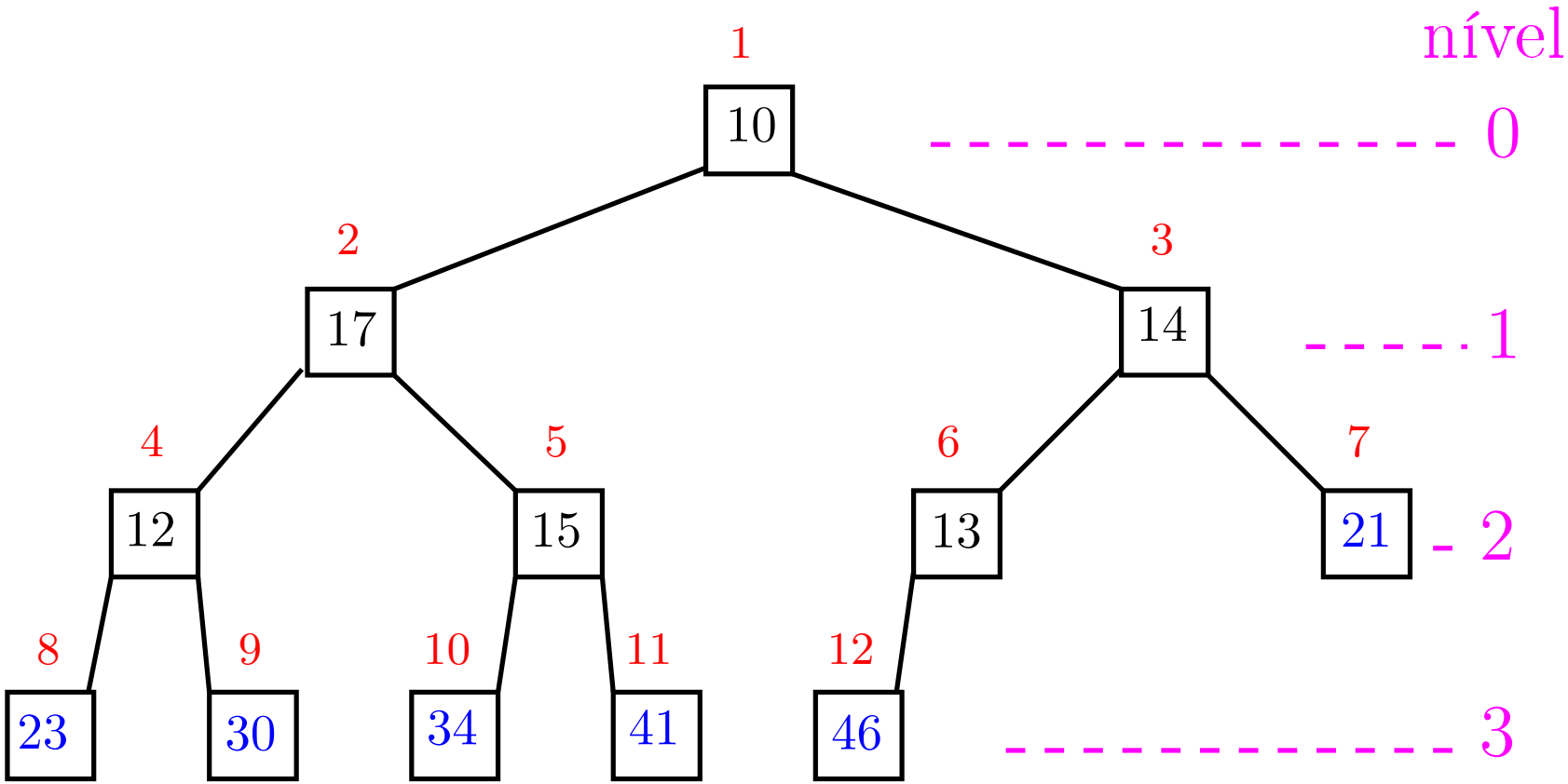
1	2	3	4	5	6	7	8	9	10	11	12
21	17	14	12	15	13	10	23	30	34	41	46

Heapsort



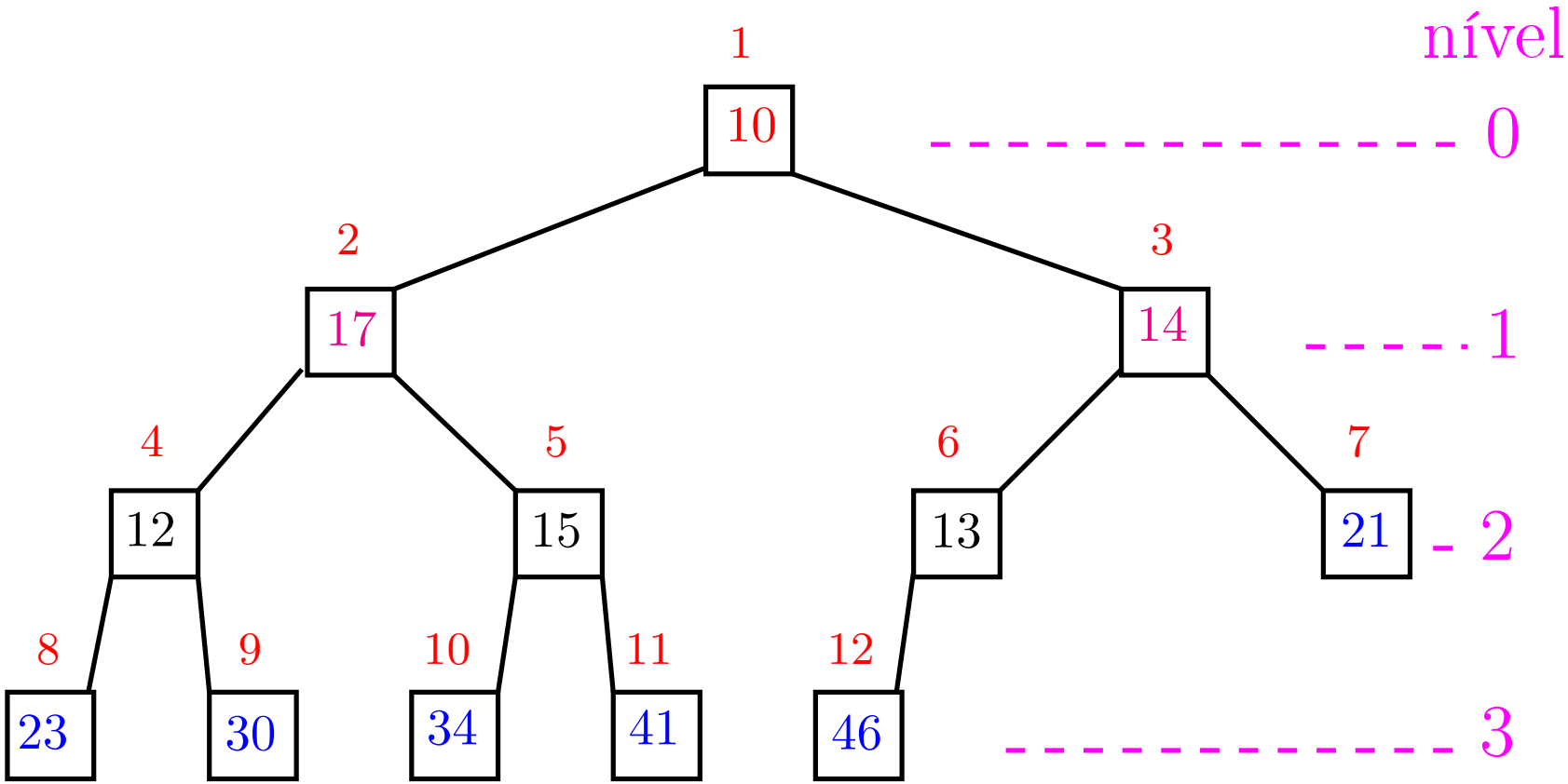
1	2	3	4	5	6	7	8	9	10	11	12
21	17	14	12	15	13	10	23	30	34	41	46

Heapsort



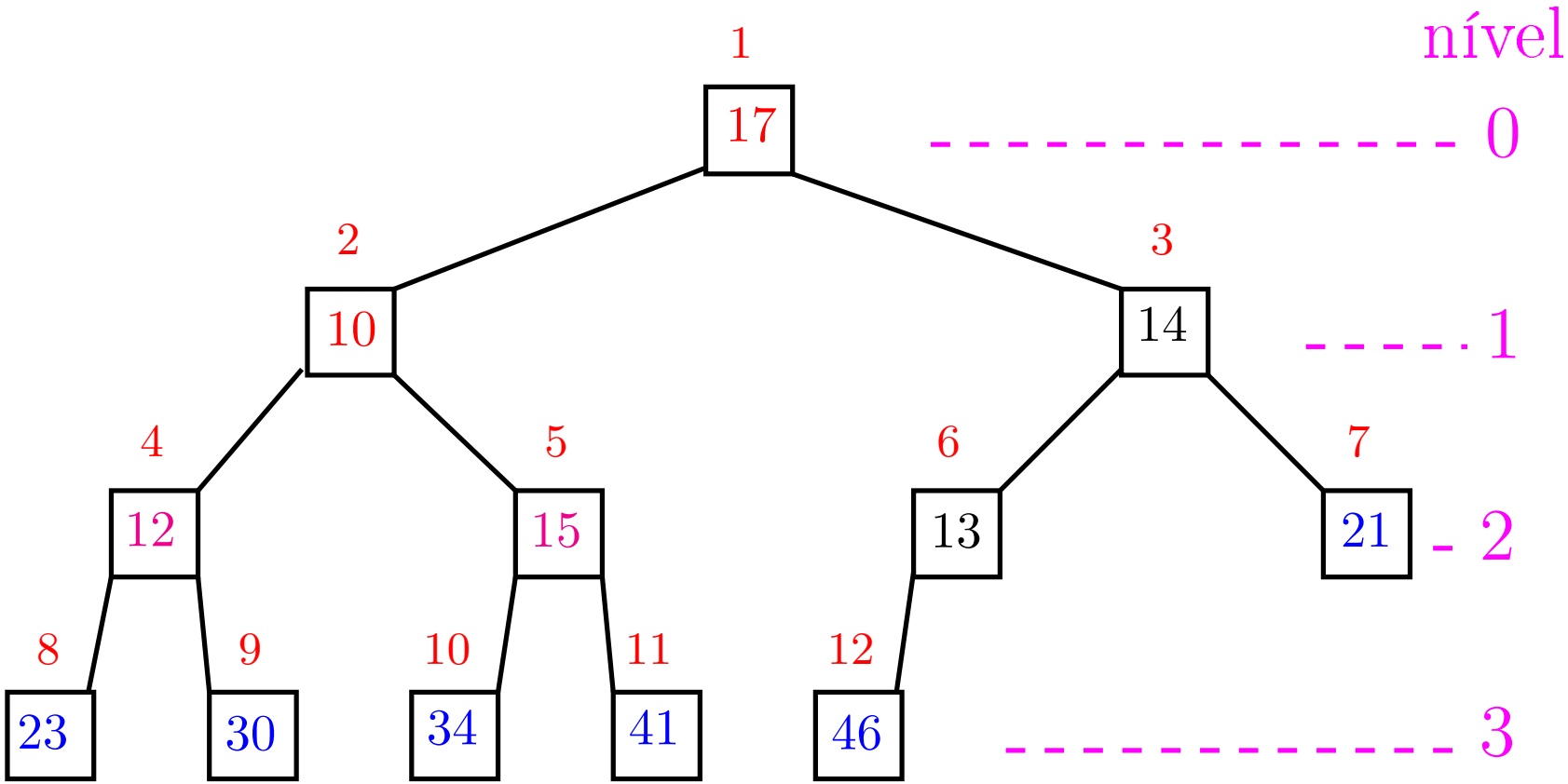
1	2	3	4	5	6	7	8	9	10	11	12
10	17	14	12	15	13	21	23	30	34	41	46

Heapsort



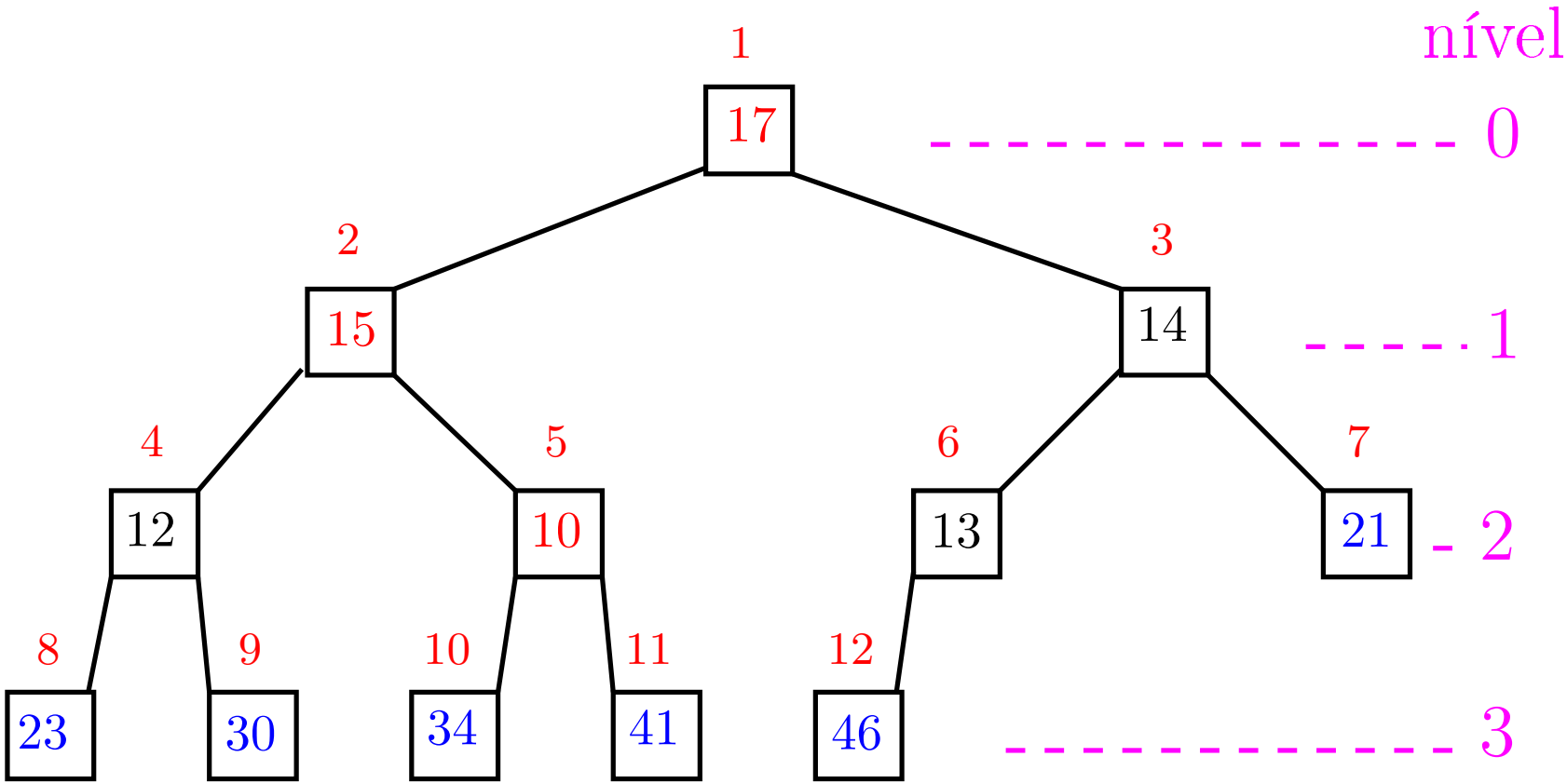
1	2	3	4	5	6	7	8	9	10	11	12
10	17	14	12	15	13	21	23	30	34	41	46

Heapsort



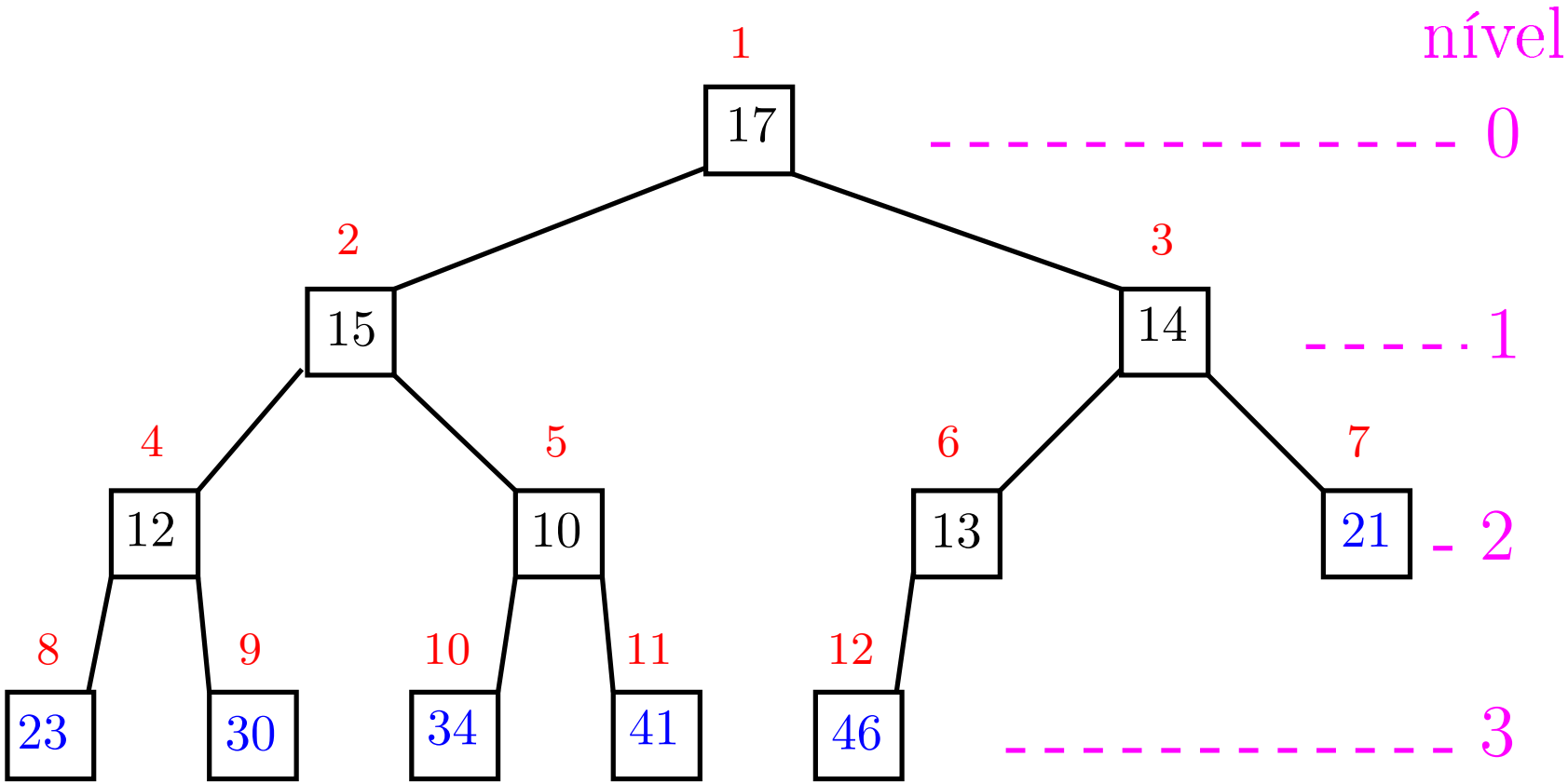
1	2	3	4	5	6	7	8	9	10	11	12
17	10	14	12	15	13	21	23	30	34	41	46

Heapsort



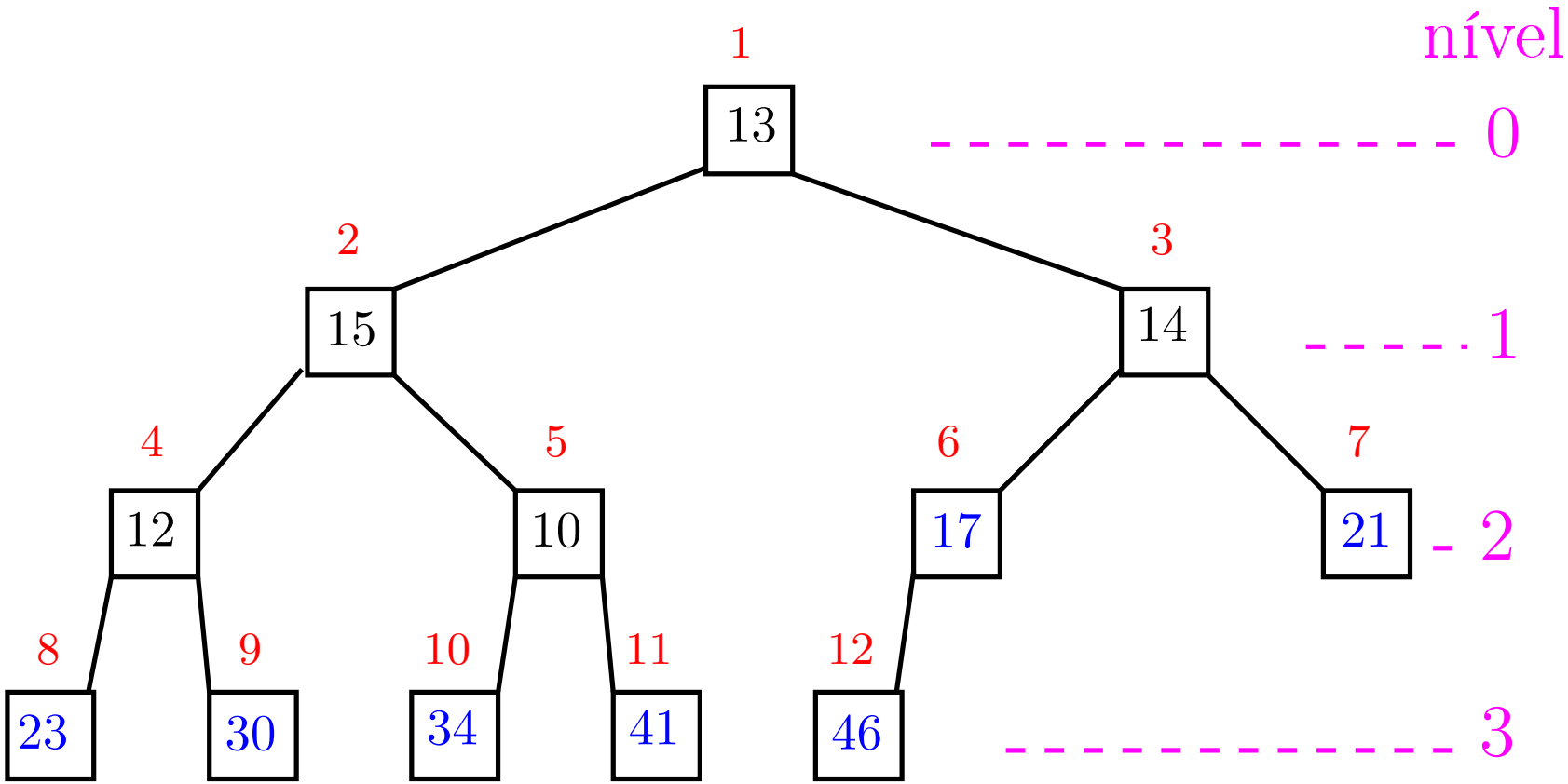
1	2	3	4	5	6	7	8	9	10	11	12
17	15	14	12	10	13	21	23	30	34	41	46

Heapsort



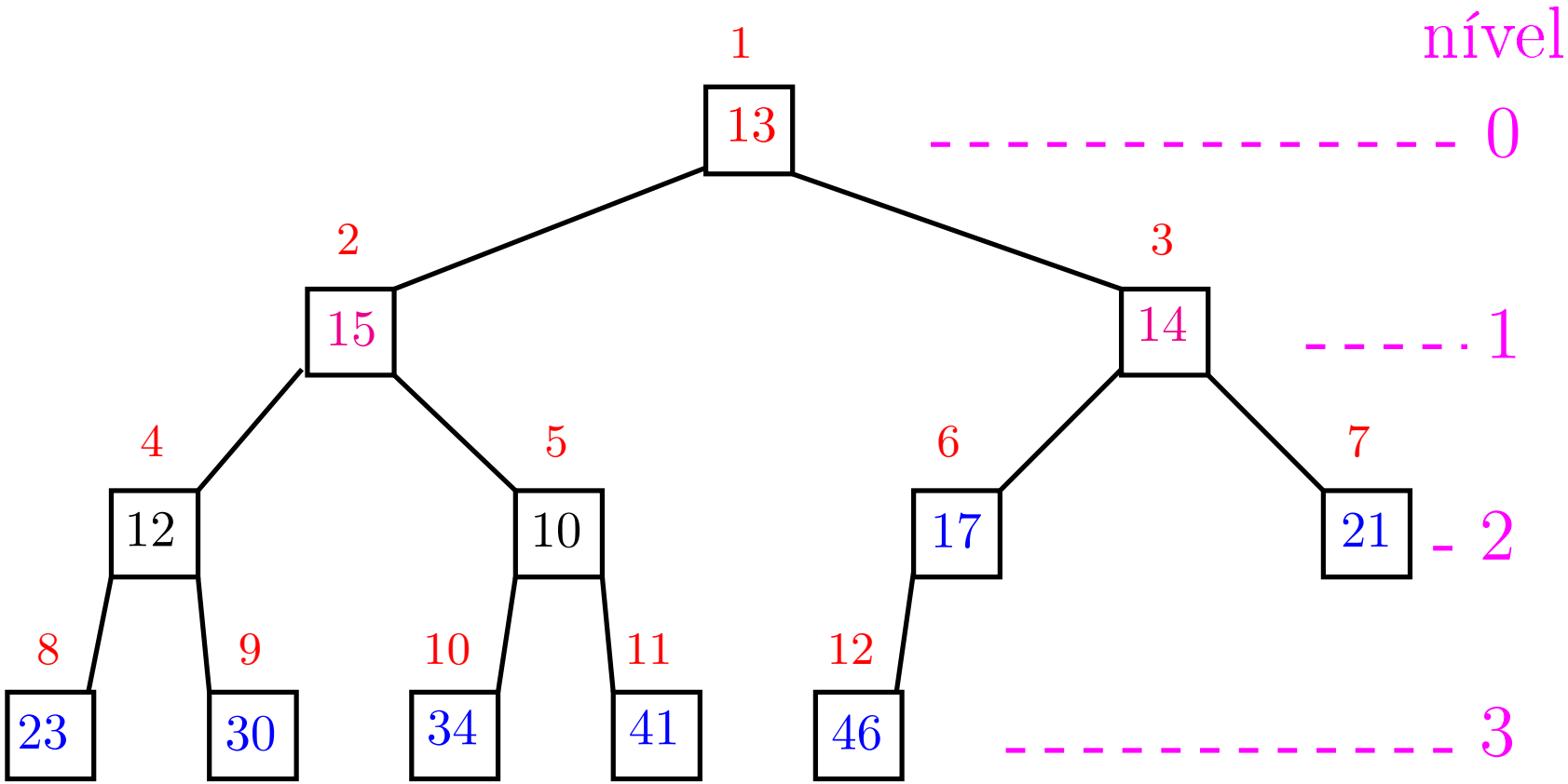
1	2	3	4	5	6	7	8	9	10	11	12
17	15	14	12	10	13	21	23	30	34	41	46

Heapsort



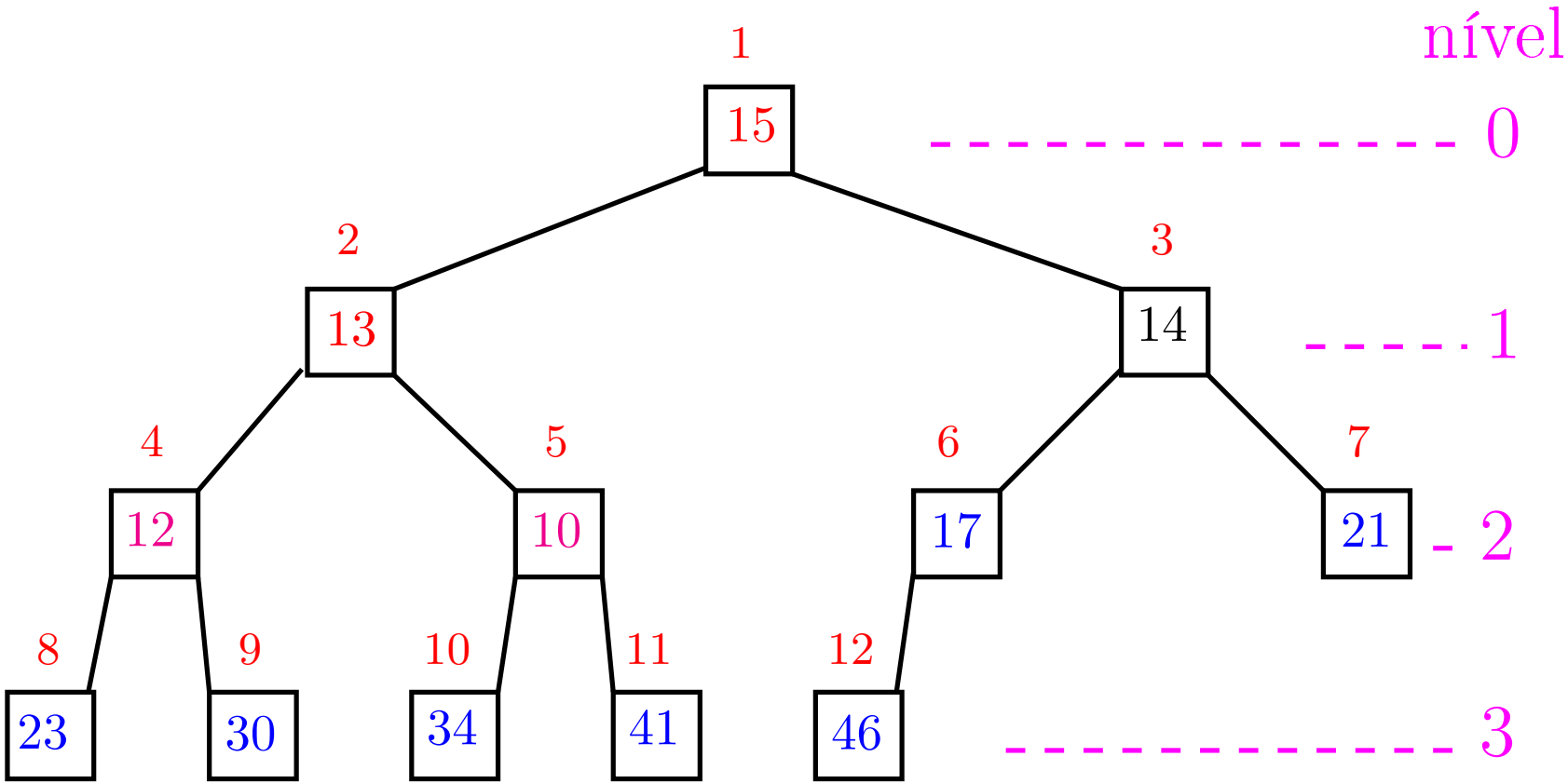
1	2	3	4	5	6	7	8	9	10	11	12
13	15	14	12	10	17	21	23	30	34	41	46

Heapsort



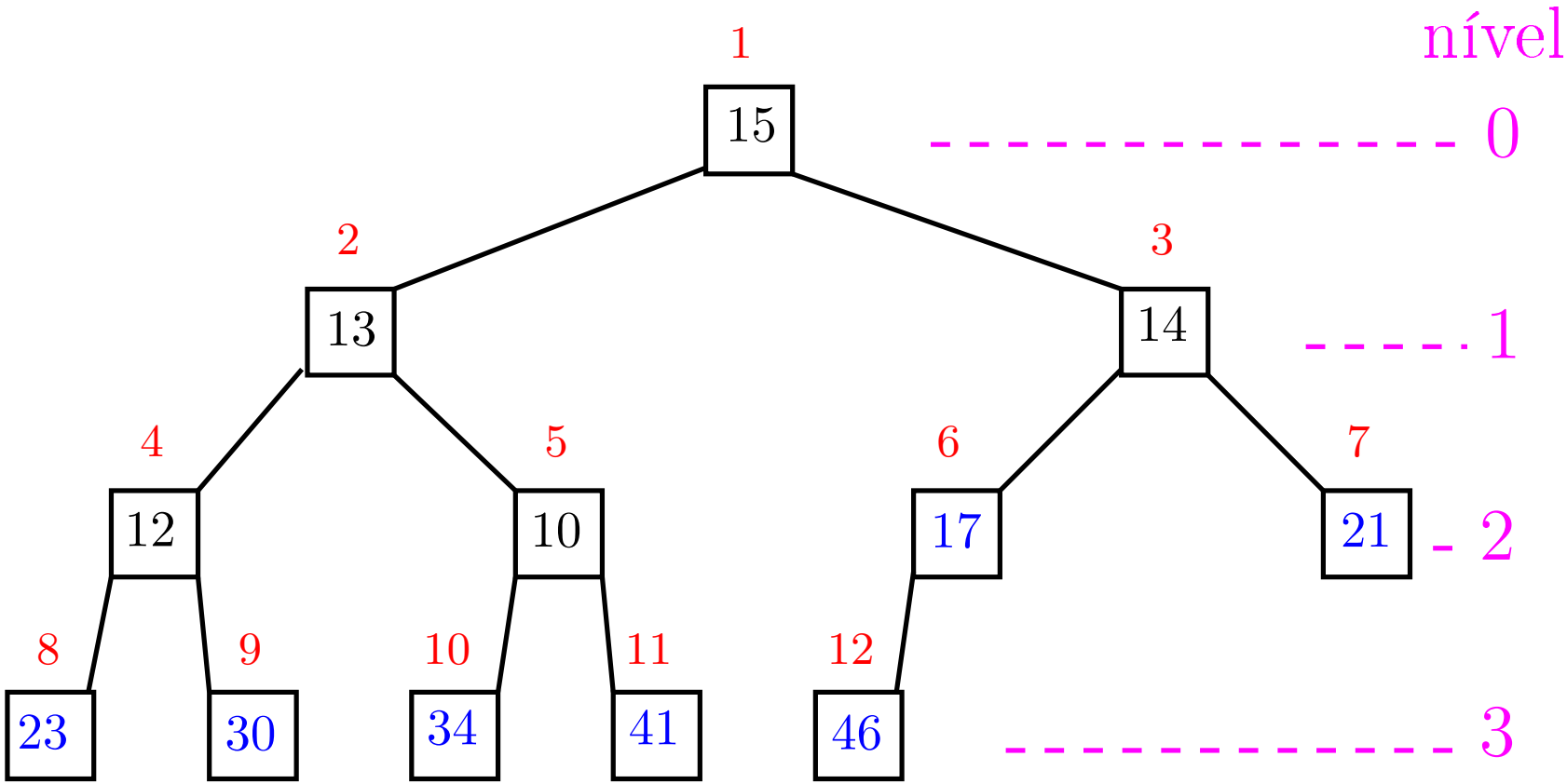
1	2	3	4	5	6	7	8	9	10	11	12
13	15	14	12	10	17	21	23	30	34	41	46

Heapsort



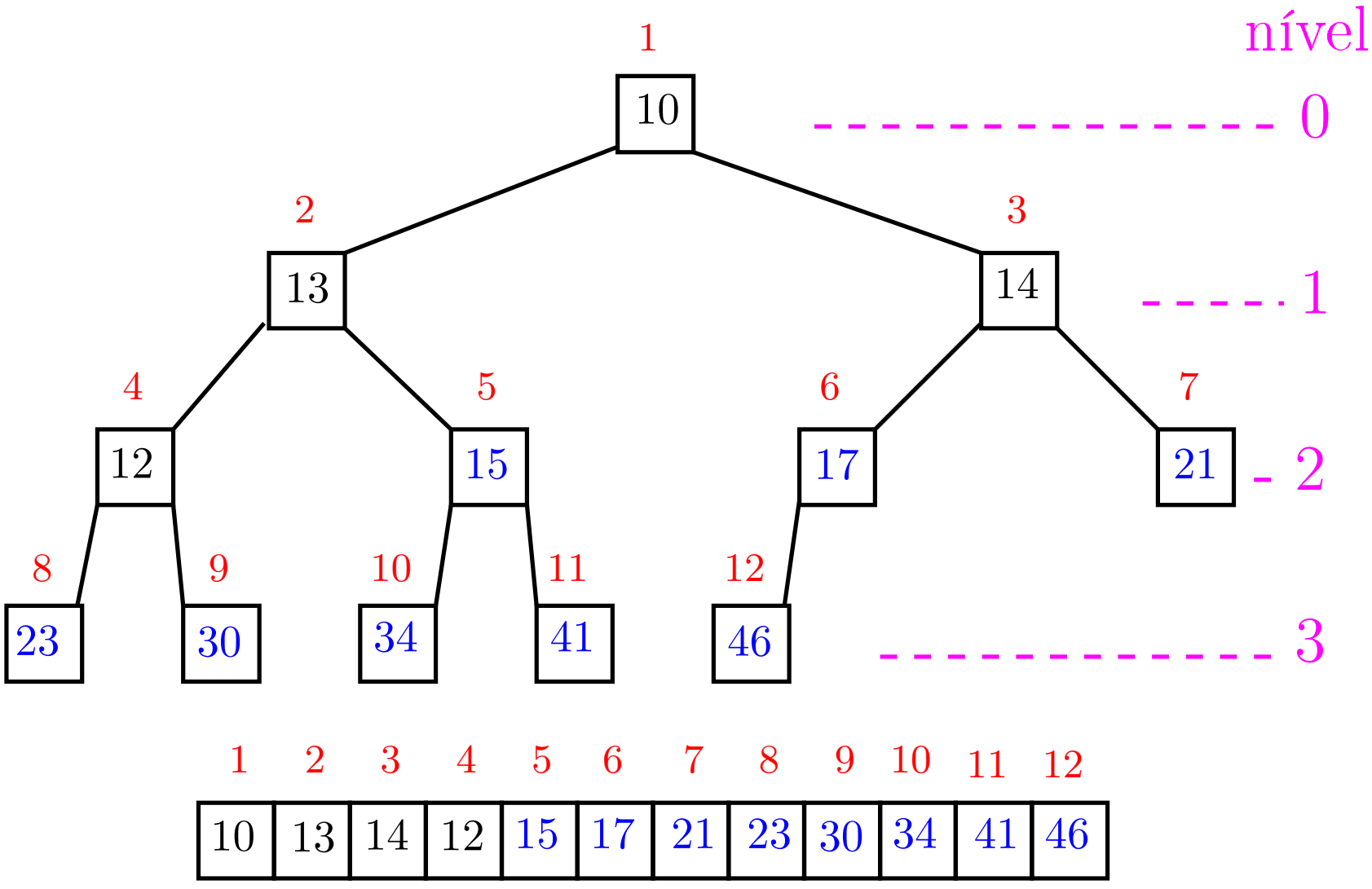
1	2	3	4	5	6	7	8	9	10	11	12
15	13	14	12	10	17	21	23	30	34	41	46

Heapsort

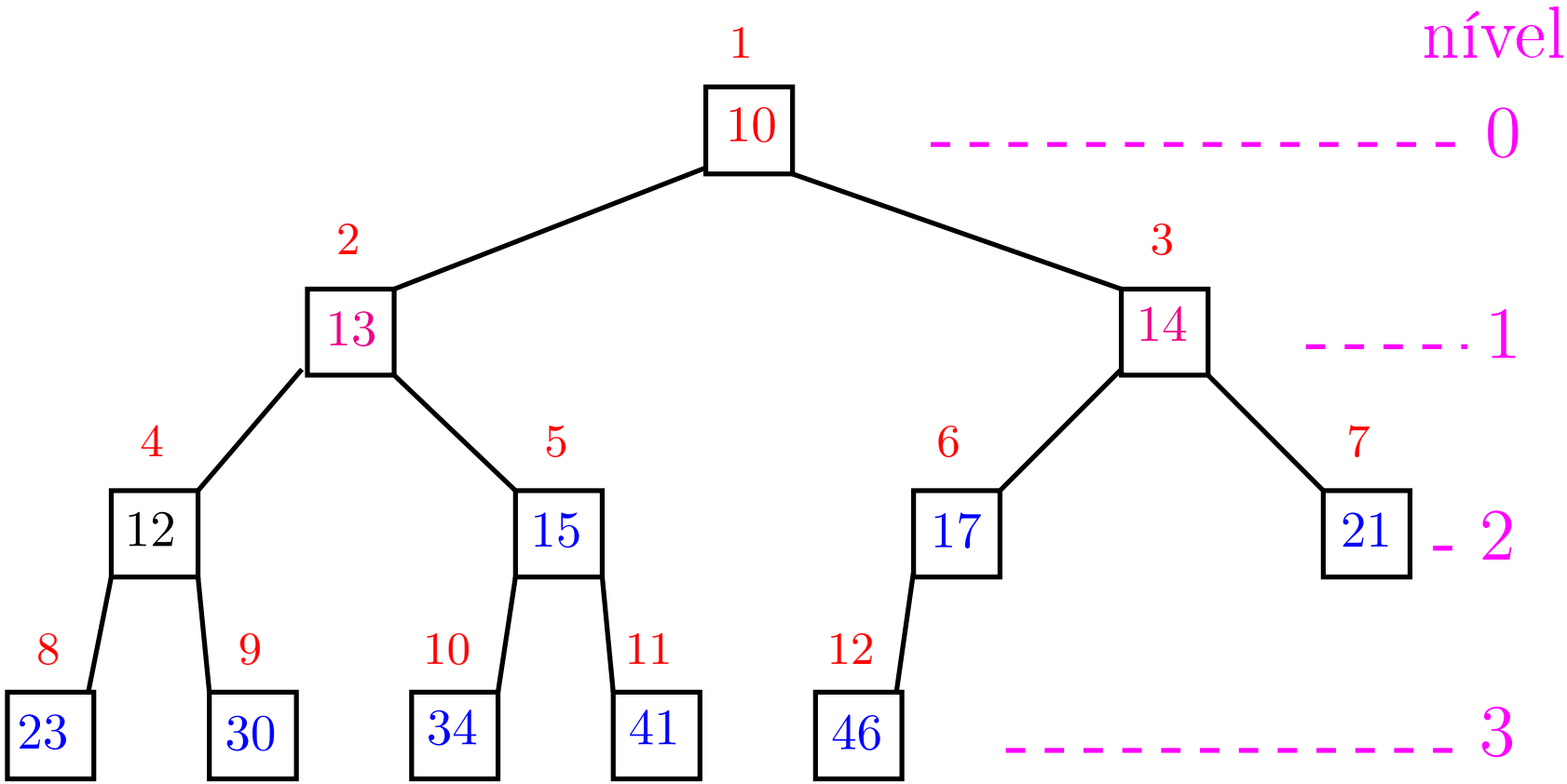


1	2	3	4	5	6	7	8	9	10	11	12
15	13	14	12	10	17	21	23	30	34	41	46

Heapsort

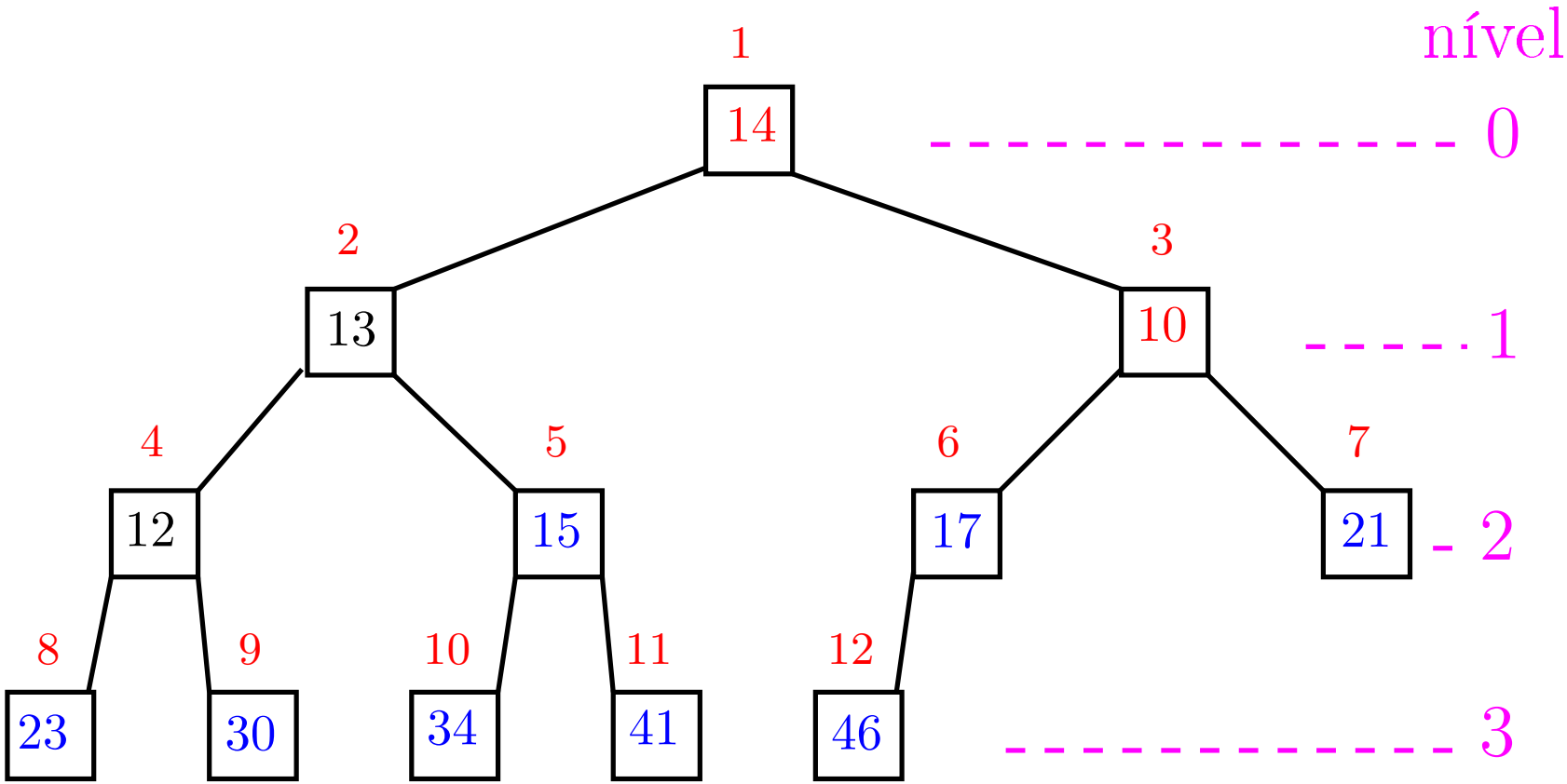

nível
1
10
0
2
13
3
14
1
4
12
5
15
6
17
7
21
2
8
23
9
30
10
34
11
41
12
46
3
1 2 3 4 5 6 7 8 9 10 11 12
10 13 14 12 15 17 21 23 30 34 41 46

Heapsort



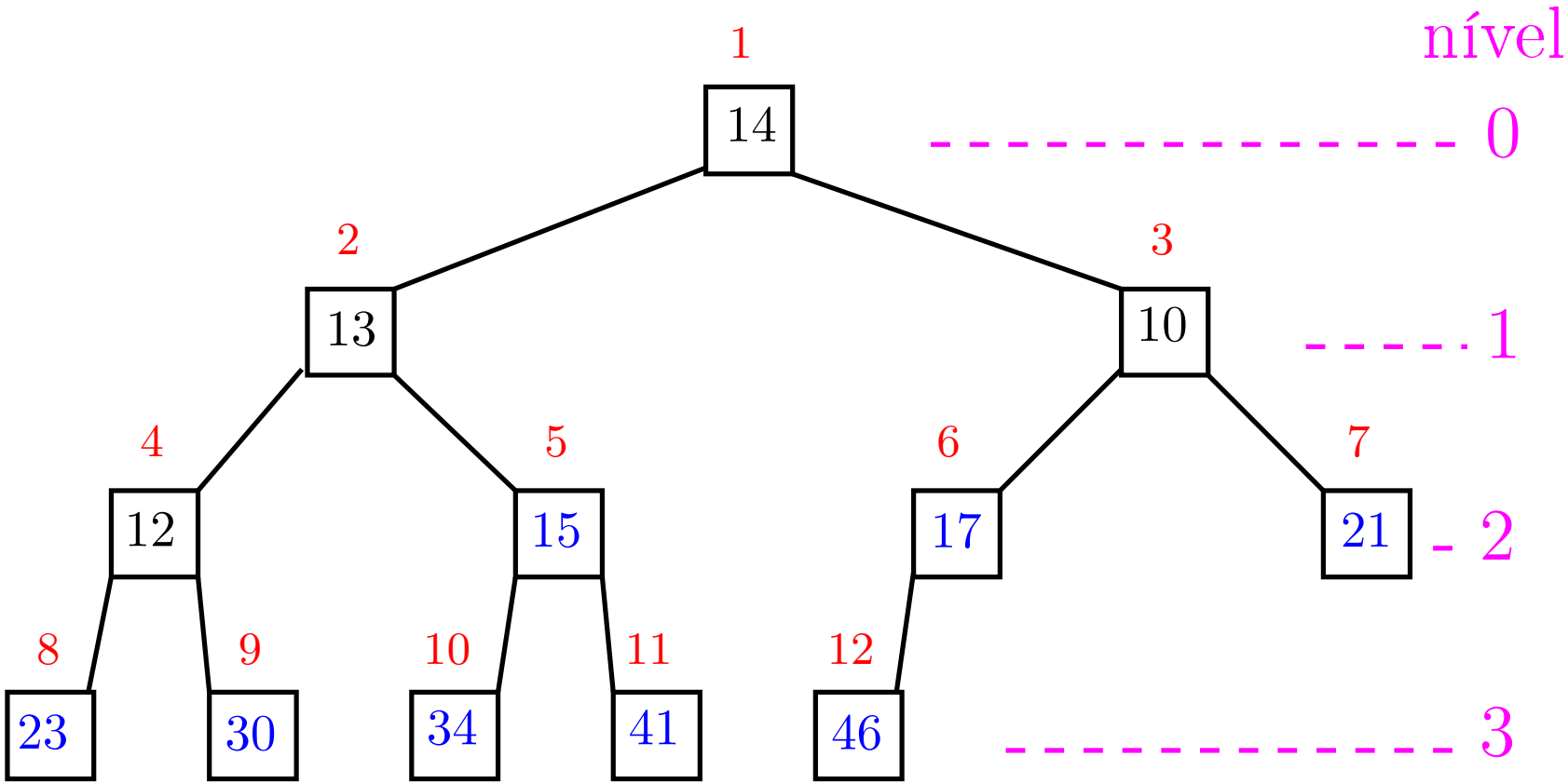
1	2	3	4	5	6	7	8	9	10	11	12
10	13	14	12	15	17	21	23	30	34	41	46

Heapsort



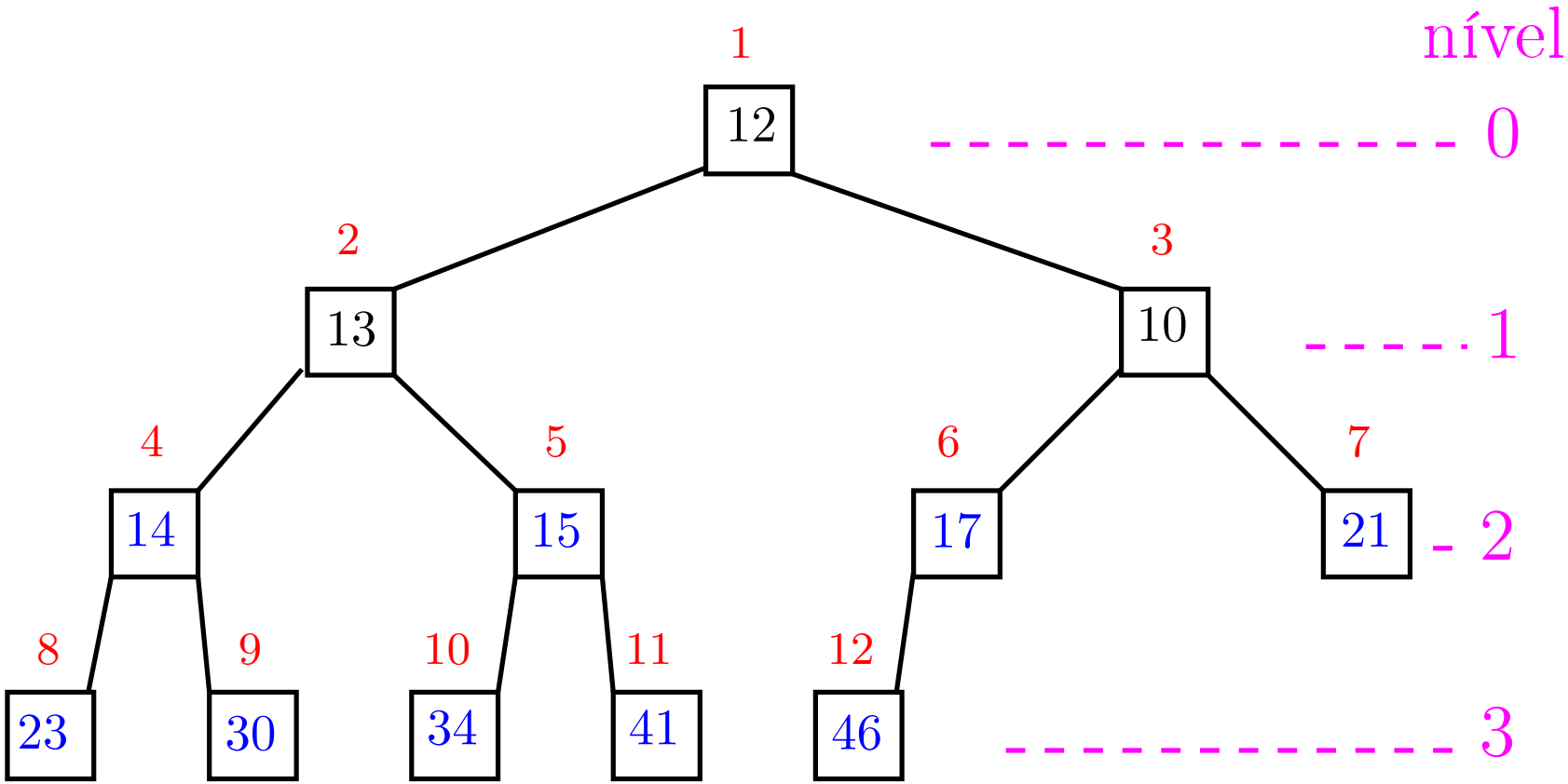
1	2	3	4	5	6	7	8	9	10	11	12
14	13	10	12	15	17	21	23	30	34	41	46

Heapsort



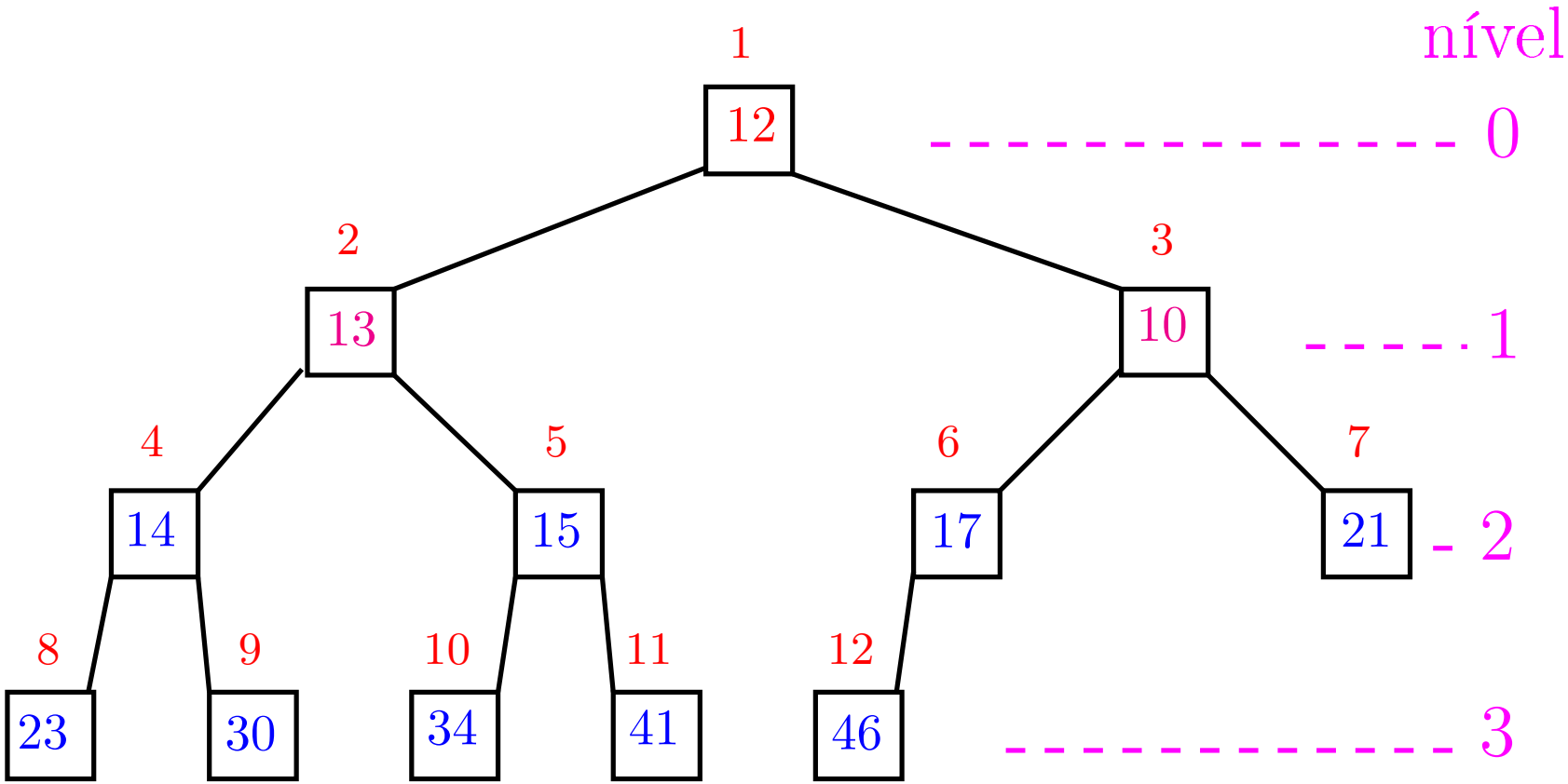
1	2	3	4	5	6	7	8	9	10	11	12
14	13	10	12	15	17	21	23	30	34	41	46

Heapsort



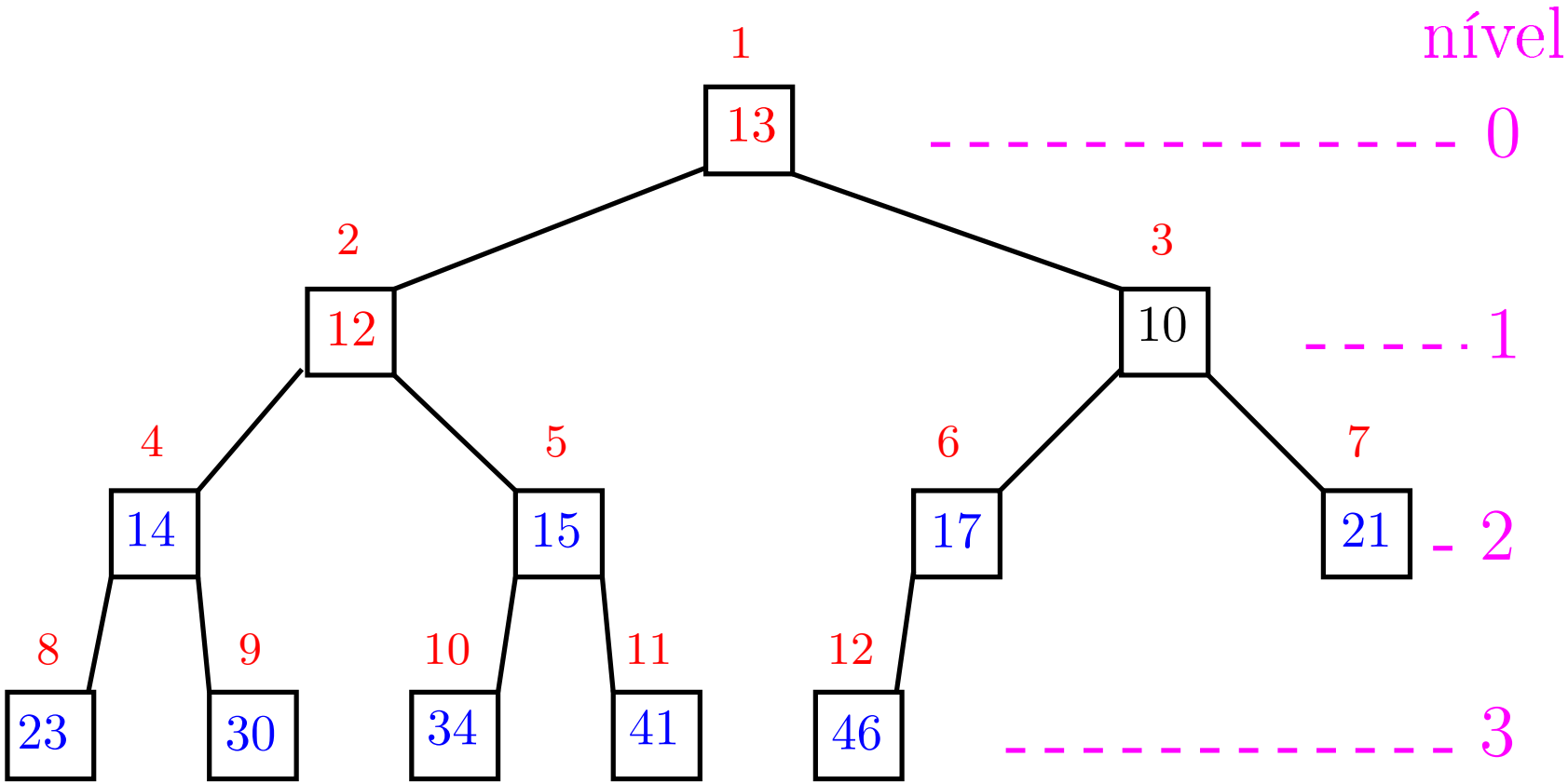
1	2	3	4	5	6	7	8	9	10	11	12
12	13	10	14	15	17	21	23	30	34	41	46

Heapsort



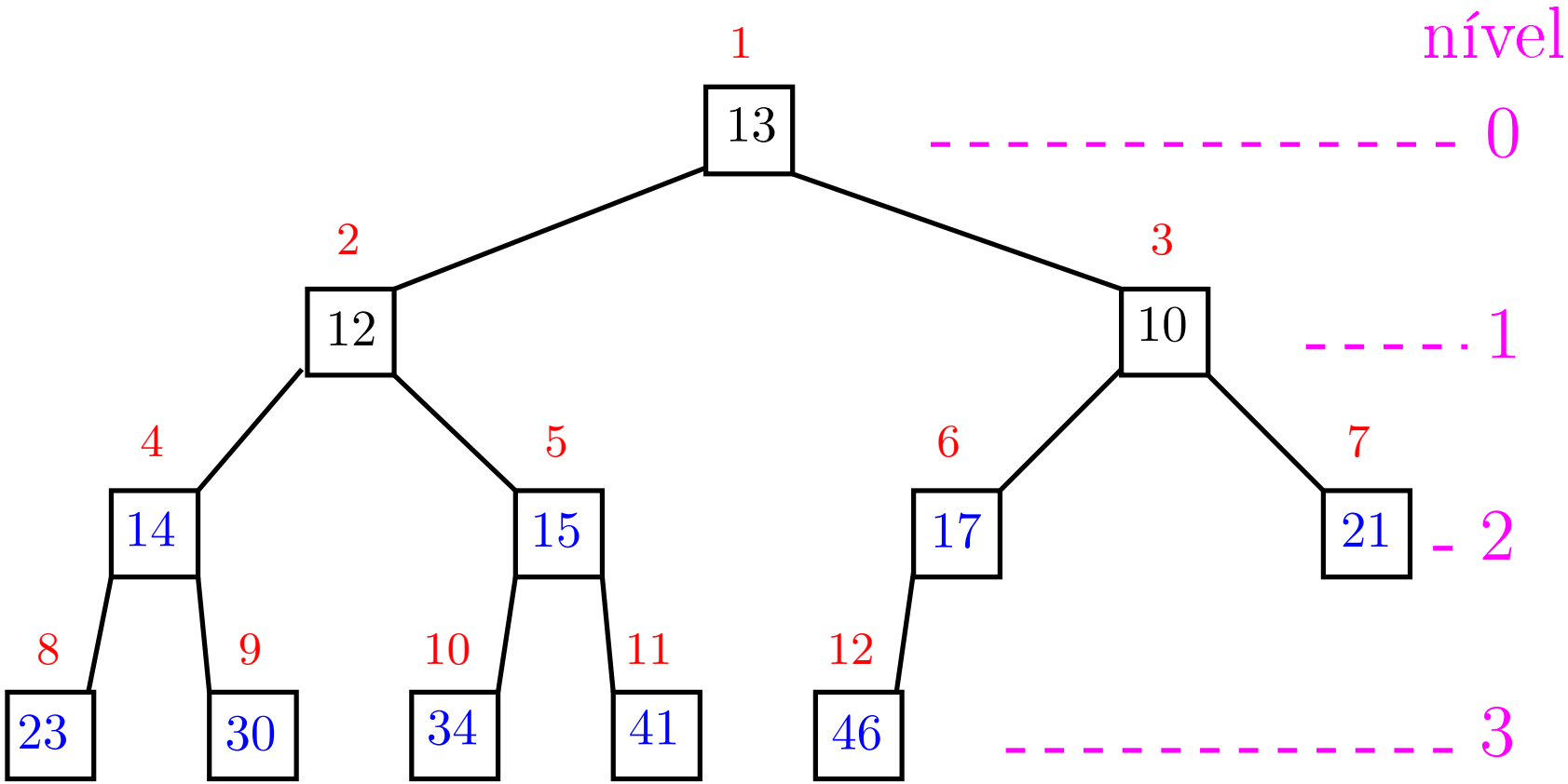
1	2	3	4	5	6	7	8	9	10	11	12
12	13	10	14	15	17	21	23	30	34	41	46

Heapsort



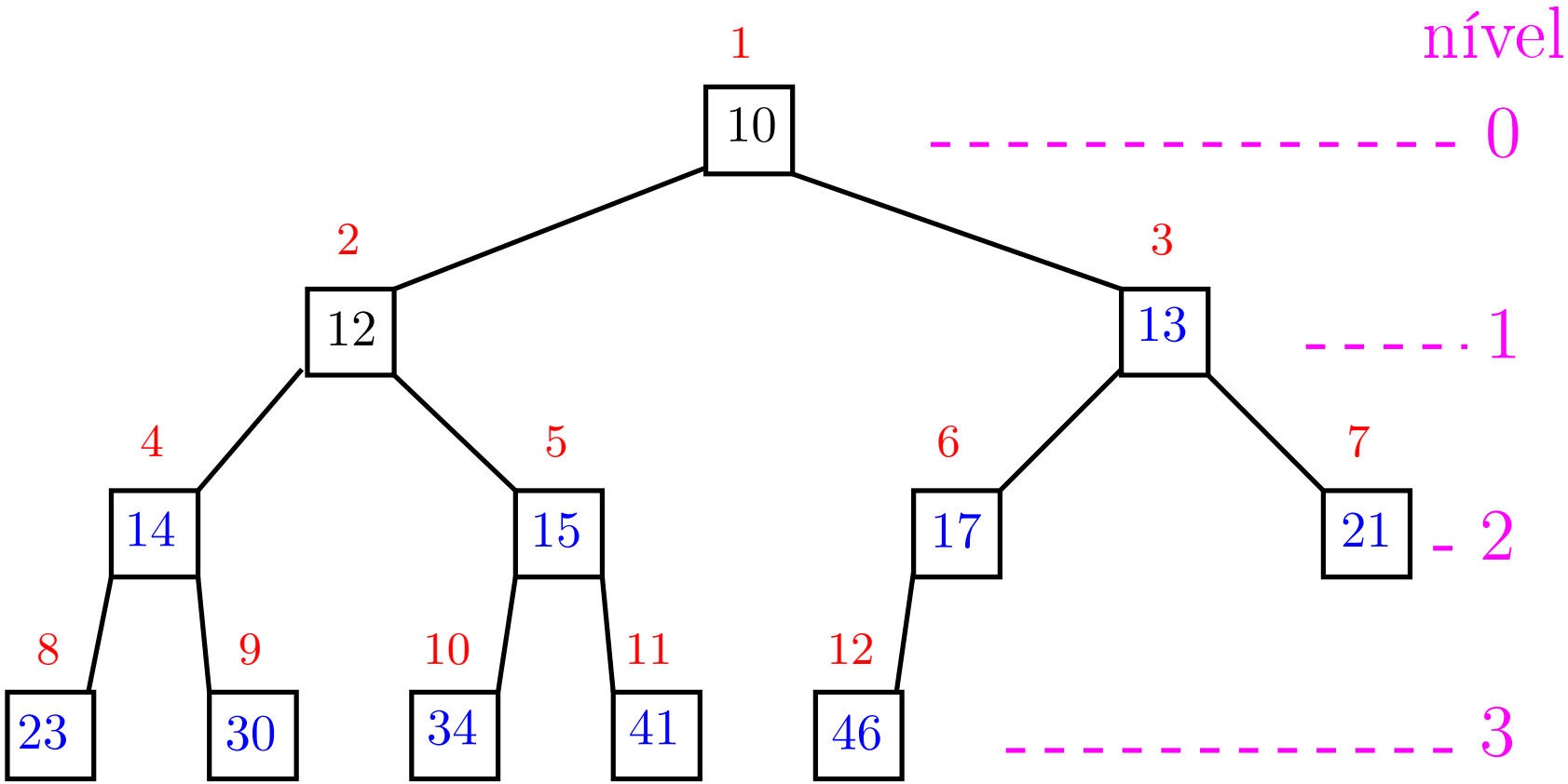
1	2	3	4	5	6	7	8	9	10	11	12
13	12	10	14	15	17	21	23	30	34	41	46

Heapsort



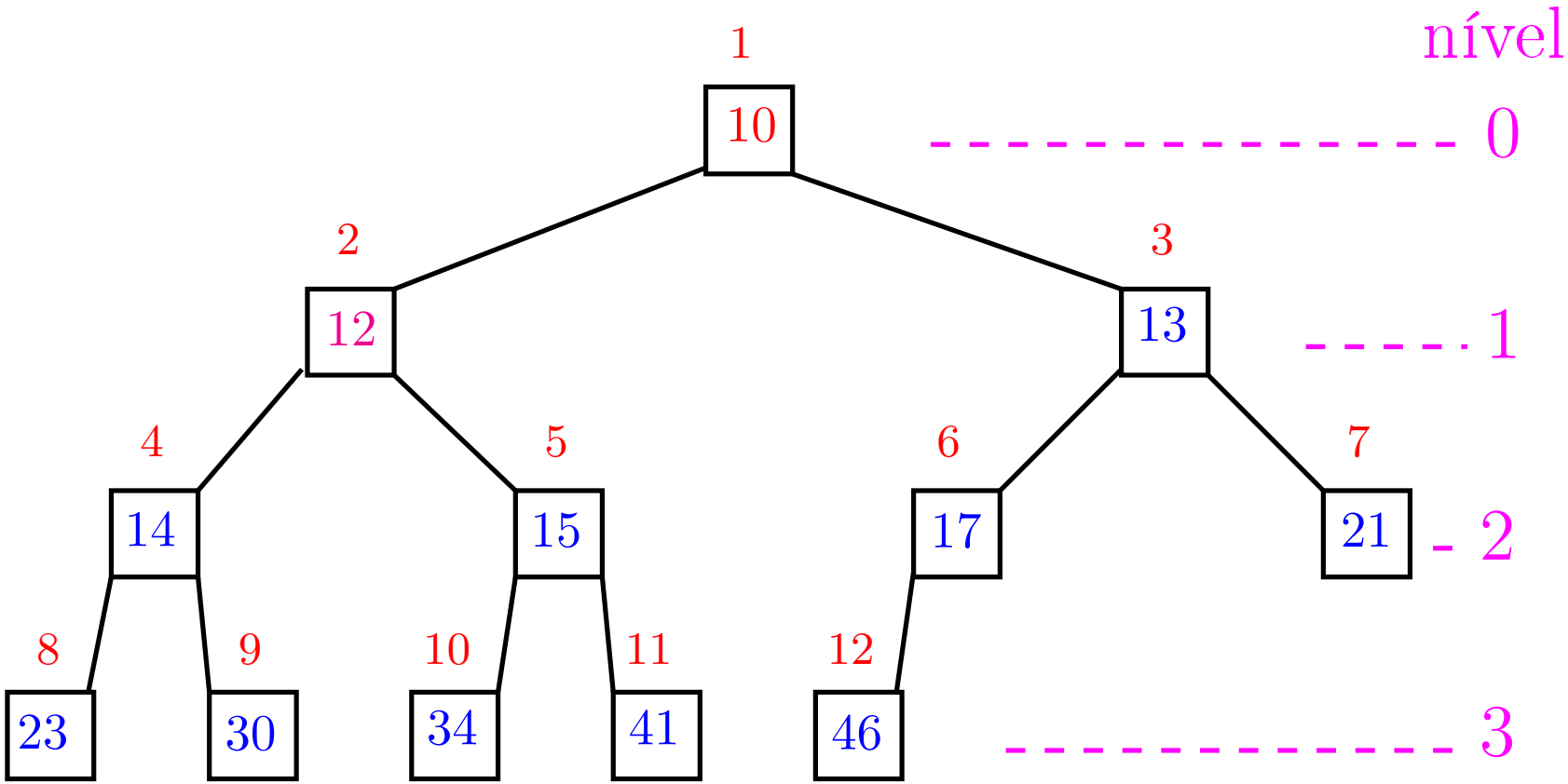
1	2	3	4	5	6	7	8	9	10	11	12
13	12	10	14	15	17	21	23	30	34	41	46

Heapsort



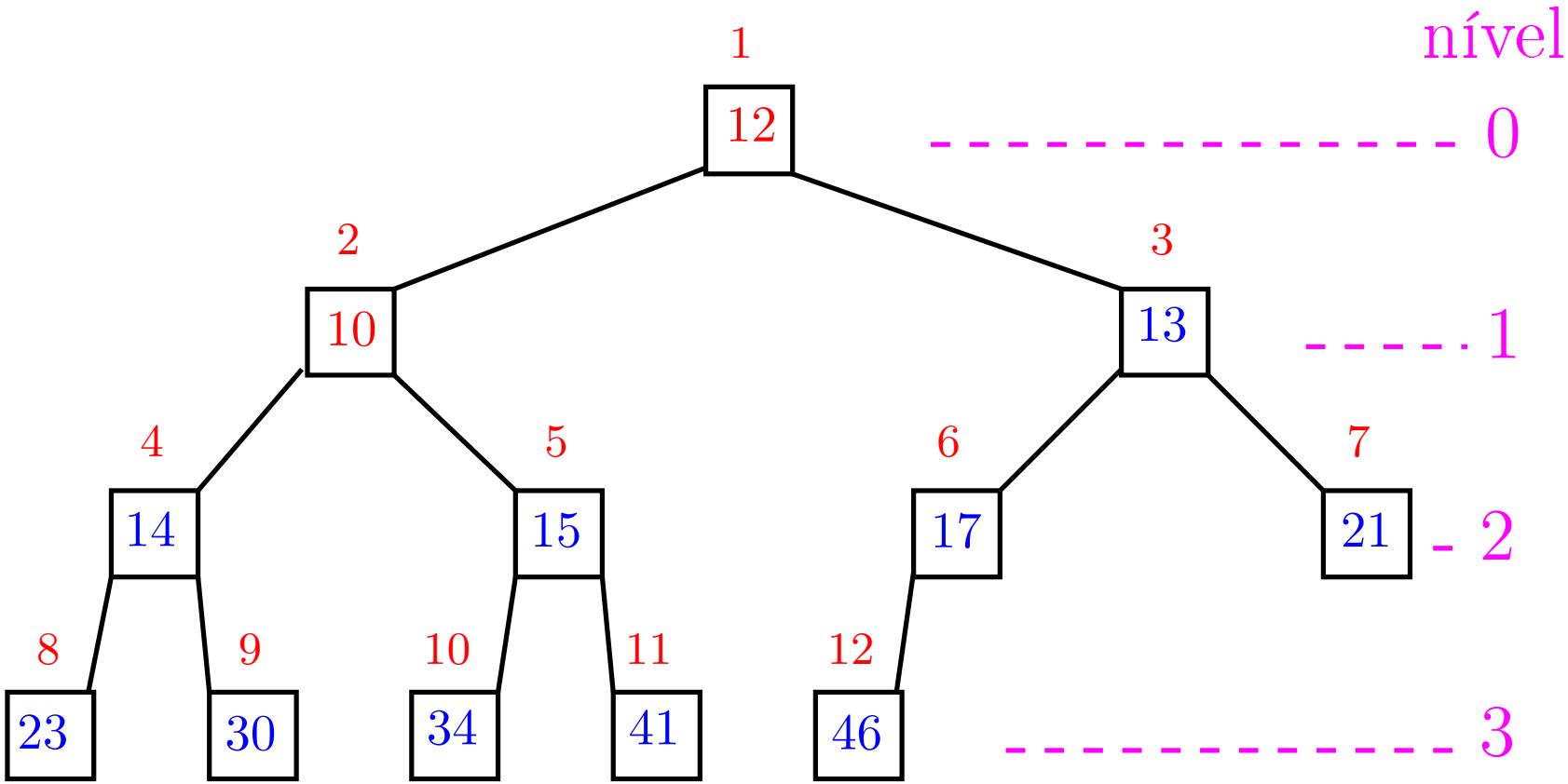
1	2	3	4	5	6	7	8	9	10	11	12
10	12	13	14	15	17	21	23	30	34	41	46

Heapsort



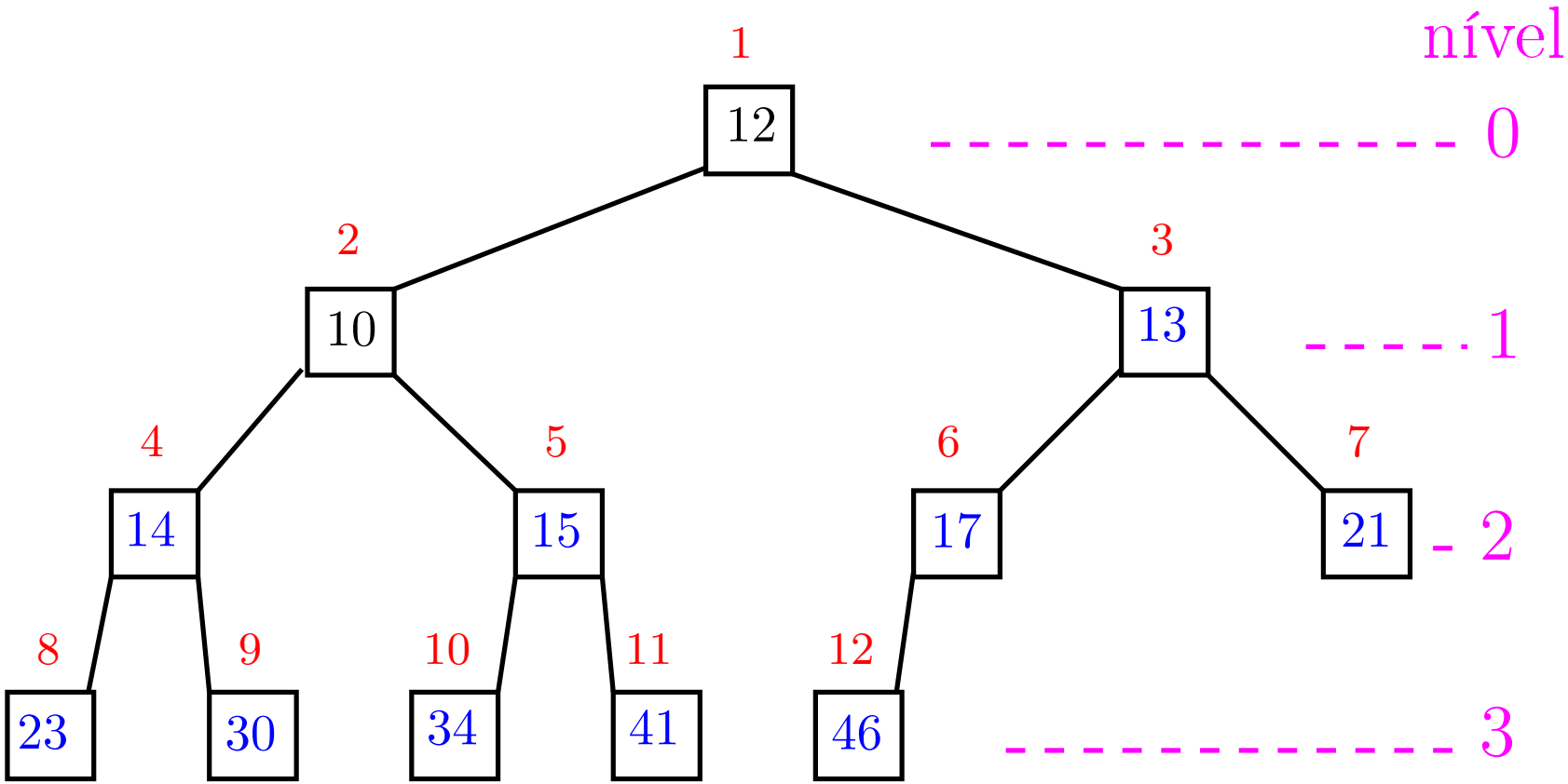
1	2	3	4	5	6	7	8	9	10	11	12
10	12	13	14	15	17	21	23	30	34	41	46

Heapsort



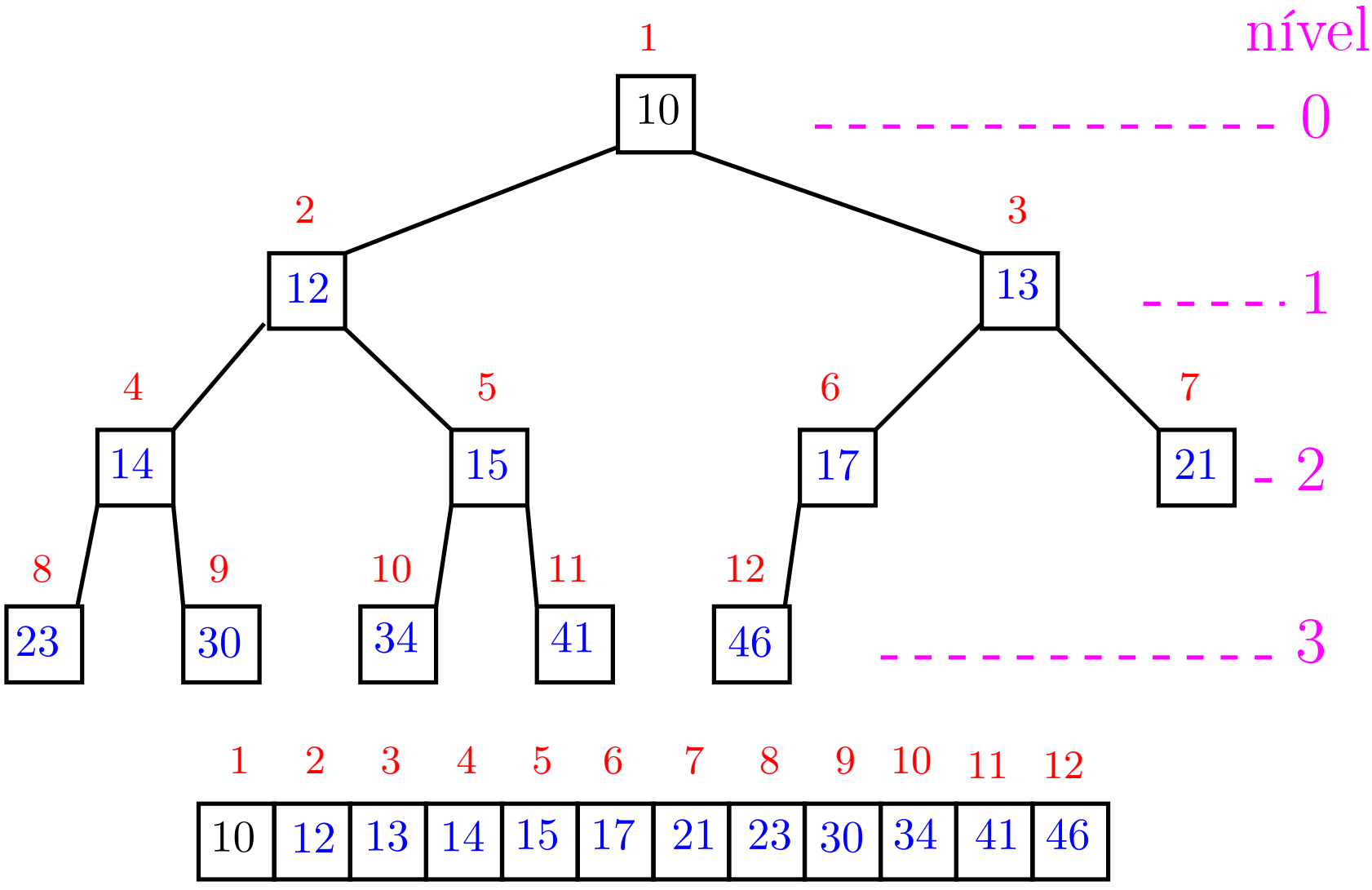
1	2	3	4	5	6	7	8	9	10	11	12
12	10	13	14	15	17	21	23	30	34	41	46

Heapsort

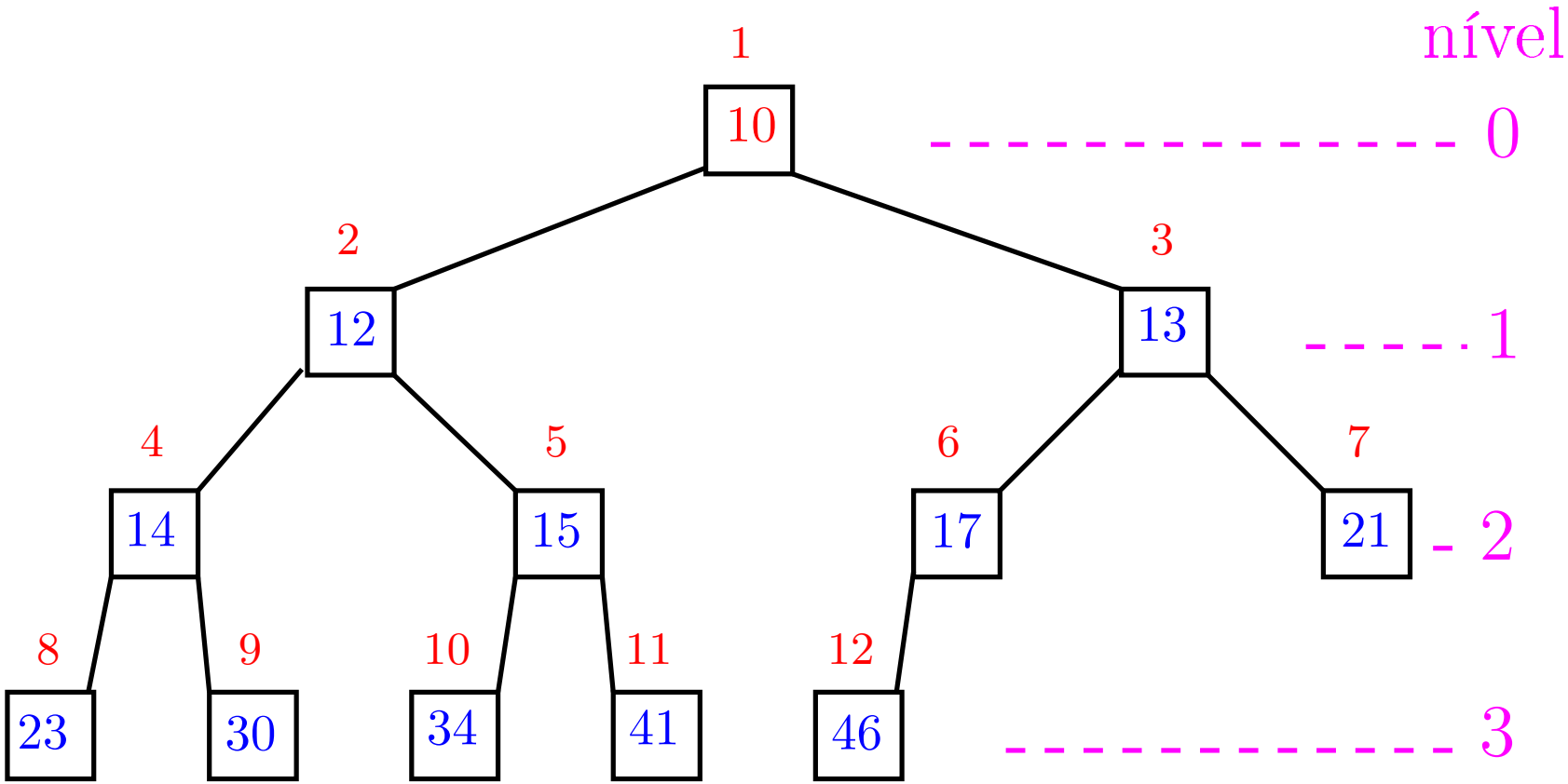


1	2	3	4	5	6	7	8	9	10	11	12
12	10	13	14	15	17	21	23	30	34	41	46

Heapsort

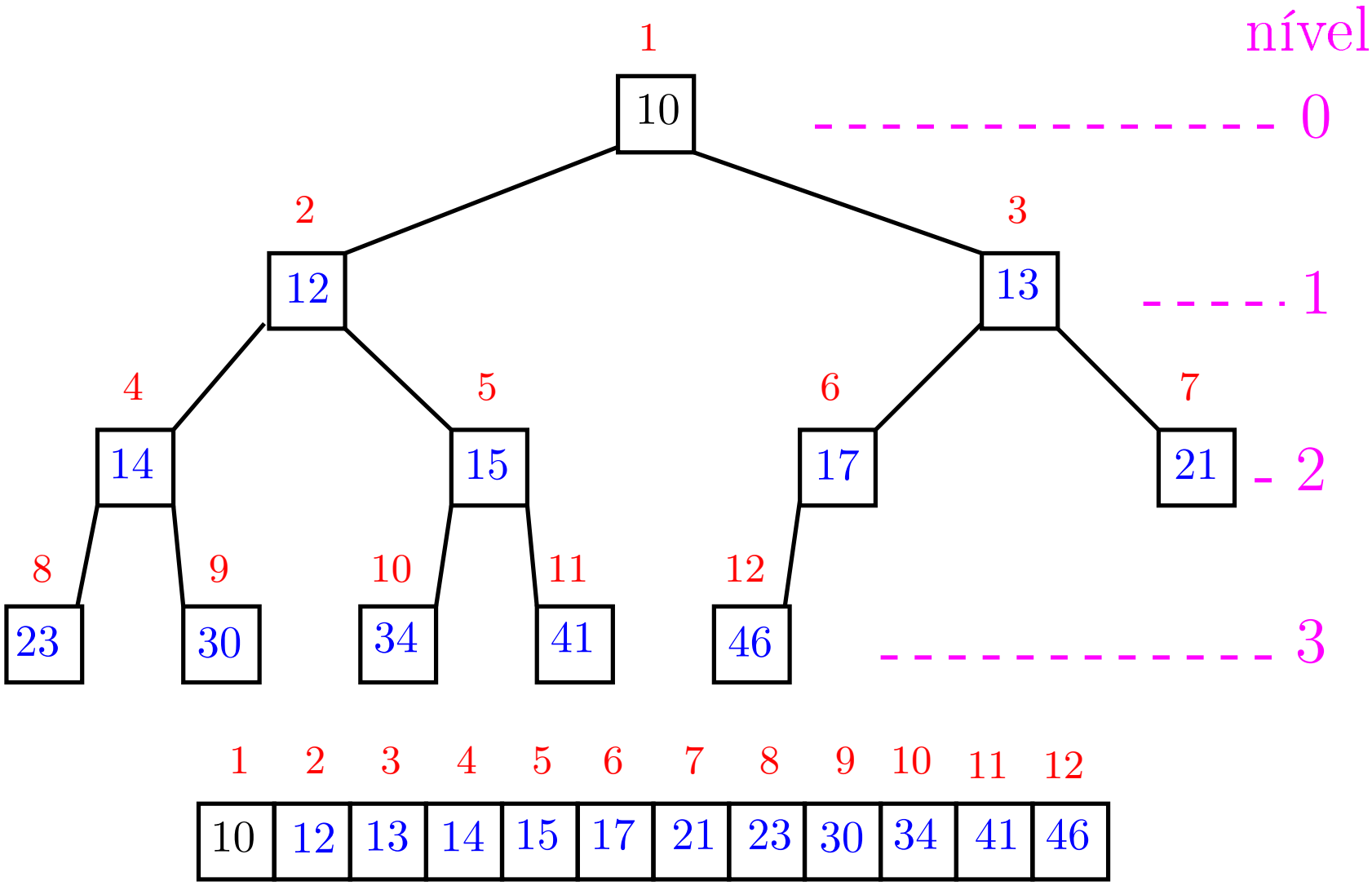

nível
1
10
0
2
12
3
13
1
4
14
5
15
6
17
7
21
2
8
23
9
30
10
34
11
41
12
46
3
1 2 3 4 5 6 7 8 9 10 11 12
10 12 13 14 15 17 21 23 30 34 41 46

Heapsort

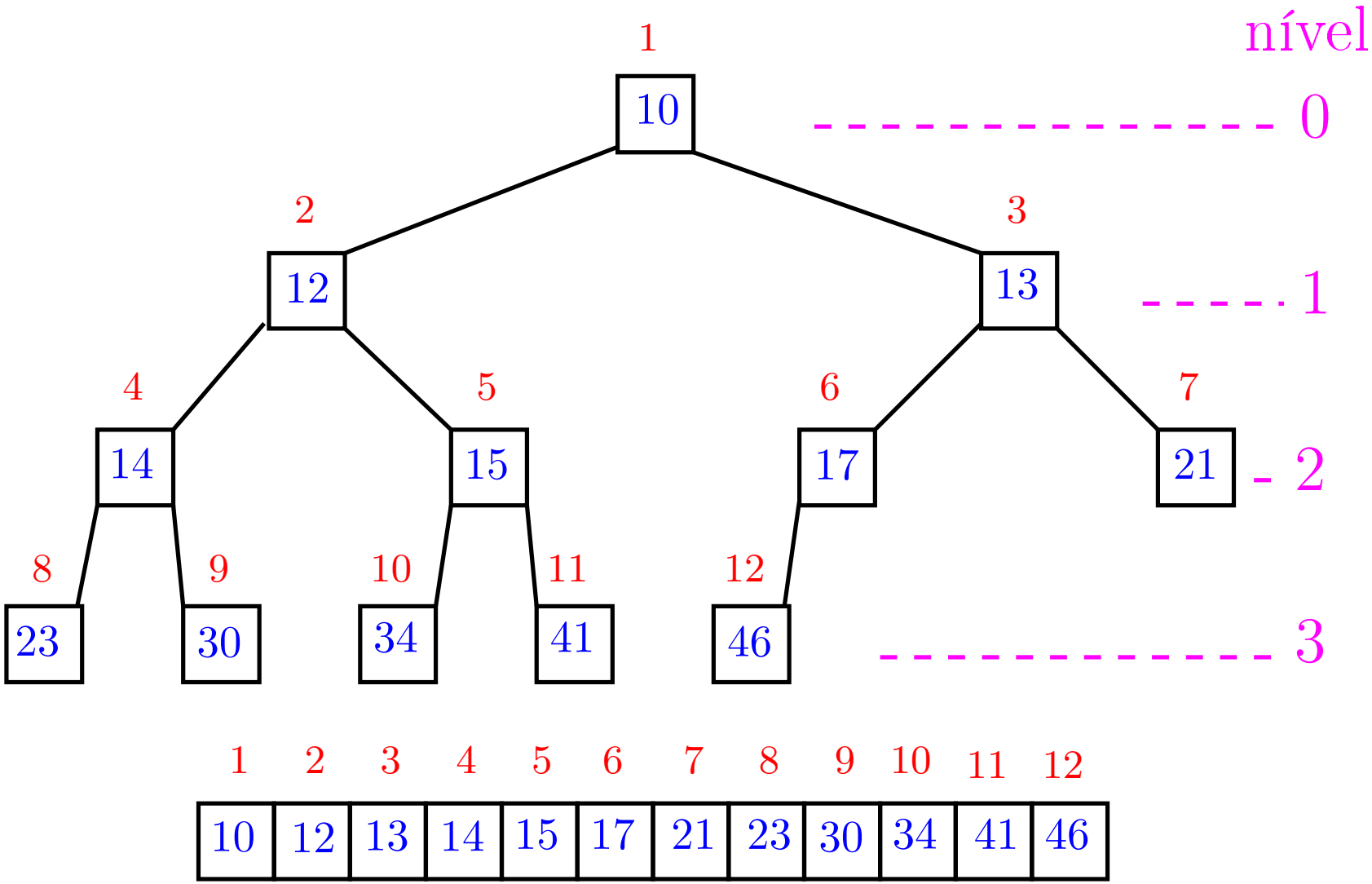


1	2	3	4	5	6	7	8	9	10	11	12
10	12	13	14	15	17	21	23	30	34	41	46

Heapsort


nível
1
10
0
2
12
3
13
1
4
14
5
15
6
17
7
21
2
8
23
9
30
10
34
11
41
12
46
3
1 2 3 4 5 6 7 8 9 10 11 12
10 12 13 14 15 17 21 23 30 34 41 46

Heapsort


nível
1
10
0
2
12
3
13
1
4
14
5
15
6
17
7
21
2
8
23
9
30
10
34
11
41
12
46
3
1 2 3 4 5 6 7 8 9 10 11 12
10 12 13 14 15 17 21 23 30 34 41 46

Função heapSort

Algoritmo rearranja $v[1..n]$ em ordem crescente

```
void heapSort (int n, int v[])
{
    int i, j, max, x;
    /* pre-processamento */
1   for (i = n/2; i >= 1; i--)
2       peneira(i, n, v);

3   for (i = n; /*C*/ i >= 2; i--) {
4       x=v[i]; v[i]=v[1]; v[1]=x;
5       peneira(1,i-1,v);
    }
}
```

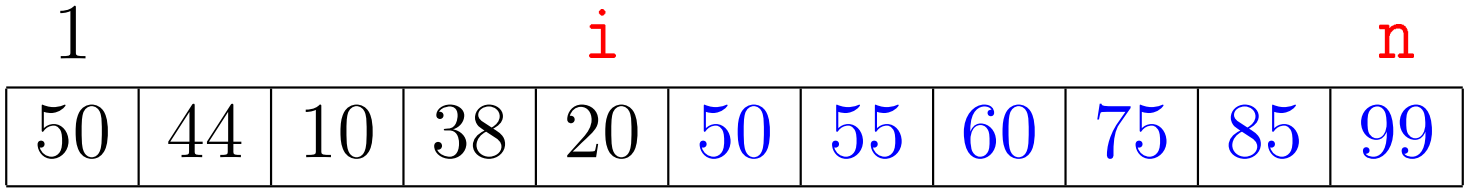
Função heapSort

Relações invariantes: Em /*C*/ vale que:

(i0) v[i+1 .. n] é crescente;

(i1) v[1 .. i] ≤ v[i+1];

(i2) v[1 .. i] é um max-heap.



Consumo de tempo

linha	consumo de tempo das execuções da linha		
1-2	$\approx$	$n \lg n$	$= O(n \lg n)$
3	$\approx$	n	$= O(n)$
4	$\approx$	n	$= O(n)$
5	$\approx$	$n \lg n$	$= O(n \lg n)$
total	$=$	$2n \lg n + 2n$	$= O(n \lg n)$

Conclusão

O consumo de tempo da função `heapSort` é proporcional a $n \lg n$.

O consumo de tempo da função `heapSort` é $O(n \lg n)$.

Mais análise experimental

Algoritmos implementados:

`mergeR`	`mergeSort` recursivo.
`mergeI`	`mergeSort` iterativo.
`quick`	`quickSort` recursivo.
`heap`	`heapSort`.

Mais análise experimental

A plataforma utilizada nos experimentos foi um computador rodando Ubuntu GNU/Linux 3.5.0-17

Compilador:

```
gcc -Wall -ansi -O2 -pedantic
-Wno-unused-result.
```

Computador:

```
 model name:  Intel(R) Core(TM)2 Quad CPU Q6600 @
2.40GHz
cpu MHz :  1596.000
cache size:  4096 KB
MemTotal :  3354708 kB
```

Aleatório: média de 10

n	mergeR	mergeI	quick	heap
8192	0.00	0.00	0.00	0.00
16384	0.00	0.00	0.00	0.00
32768	0.01	0.01	0.01	0.00
65536	0.01	0.01	0.01	0.01
131072	0.02	0.02	0.02	0.03
262144	0.05	0.04	0.04	0.06
524288	0.10	0.08	0.08	0.12
1048576	0.21	0.20	0.17	0.28
2097152	0.44	0.43	0.35	0.70
4194304	0.92	0.90	0.73	1.73
8388608	1.90	1.87	1.51	4.13

Tempos em segundos.

Decrescente

n	mergeR	mergeI	quick	heap
1024	0.00	0.00	0.00	0.00
2048	0.00	0.00	0.00	0.00
4096	0.01	0.00	0.01	0.00
8192	0.00	0.00	0.03	0.00
16384	0.00	0.00	0.14	0.00
32768	0.00	0.01	0.57	0.00
65536	0.01	0.01	2.27	0.01
131072	0.02	0.01	9.06	0.02
262144	0.03	0.03	36.31	0.04

Tempos em segundos.

Para n=524288 quickSort dá Segmentation fault (core dumped)

Crescente

n	mergeR	mergeI	quick	heap
1024	0.00	0.00	0.00	0.00
2048	0.00	0.00	0.00	0.00
4096	0.00	0.00	0.00	0.00
8192	0.00	0.00	0.03	0.00
16384	0.00	0.00	0.14	0.01
32768	0.01	0.00	0.57	0.01
65536	0.00	0.01	2.26	0.01
131072	0.02	0.02	9.05	0.02
262144	0.03	0.02	36.21	0.04

Tempos em segundos.

Para n=524288 quickSort dá Segmentation fault (core dumped)

Resumo

função	consumo de tempo	observação
`bubble`	$O(n^2)$	todos os casos
`insercao`	$O(n^2)$	pior caso
	$O(n)$	melhor caso
`insercaoBinaria`	$O(n^2)$	pior caso
	$O(n \lg n)$	melhor caso
`selecao`	$O(n^2)$	todos os casos
`mergeSort`	$O(n \lg n)$	todos os casos
`quickSort`	$O(n^2)$	pior caso
	$O(n \lg n)$	melhor caso
`heapSort`	$O(n \lg n)$	todos os casos