ORDENAÇÃO: *HEAPSORT*

PUC MINAS

ENGENHARIA DE SOFTWARE

ORDENAÇÃO POR SELEÇÃO

- O método de ordenação por seleção **itera *n*-1 vezes** sobre o conjunto de elementos a ser ordenado:
  - **em cada iteração, compara** o elemento atual com os demais elementos não ordenados;
  - **em cada iteração**, seleciona o menor elemento e o **troca** com o elemento na primeira posição não-ordenada do conjunto.

ORDENAÇÃO POR SELEÇÃO

- Problema do método de ordenação por **seleção:**
  - custo para se selecionar o elemento que ocupará a posição de referência.
- Como ele itera *n*-1 vezes e em cada iteração compara o elemento atual com os elementos restantes;
  - apresenta complexidade $O(n^2)$.

HEAPSORT

- E se...
  - a seleção puder ser feita a baixo custo?

HEAPSORT

- **Constrói a *Heap*.**
- **Itera *n*-2 vezes:**
  - **em cada iteração, troca** o **maior elemento** da *Heap* com o **elemento** da **última posição** não ordenada do conjunto;
  - em cada iteração, **restaura** as **propriedades da *Heap*.**

HEAPSORT

- **Constrói a *Heap*.** ← **Como se constrói uma *Heap*?**
- **Itera *n*-2 vezes:**
  - **em cada iteração, troca** o **maior elemento** da *Heap* com o **elemento** da **última posição** não ordenada do conjunto;
  - em cada iteração, **restaura** as **propriedades da *Heap*.**

HEAPSORT

- **Constrói a *Heap*.** ← **O que é uma *Heap*?**
- **Itera *n*-2 vezes:**
  - **em cada iteração, troca** o **maior elemento** da *Heap* com o **elemento** da **última posição** não ordenada do conjunto;
  - em cada iteração, **restaura** as **propriedades da *Heap*.**

HEAPSORT

- **Constrói a *Heap*.**
- **Itera *n*-2 vezes:**
  - **em cada iteração, troca** o **maior elemento** da *Heap* com o **elemento** da **última posição** não ordenada do conjunto;
  - em cada iteração, **restaura** as **propriedades da *Heap*.**

Como se restaura as propriedades de uma *Heap*?

HEAP

- Estrutura de dados que **implementa** uma **fila de prioridades:**
  - **não** garante a ordem FIFO;
    - mas sim a ordem de prioridade pré-estabelecida;
  - o primeiro elemento que sai;
    - é o elemento que apresenta a maior prioridade.

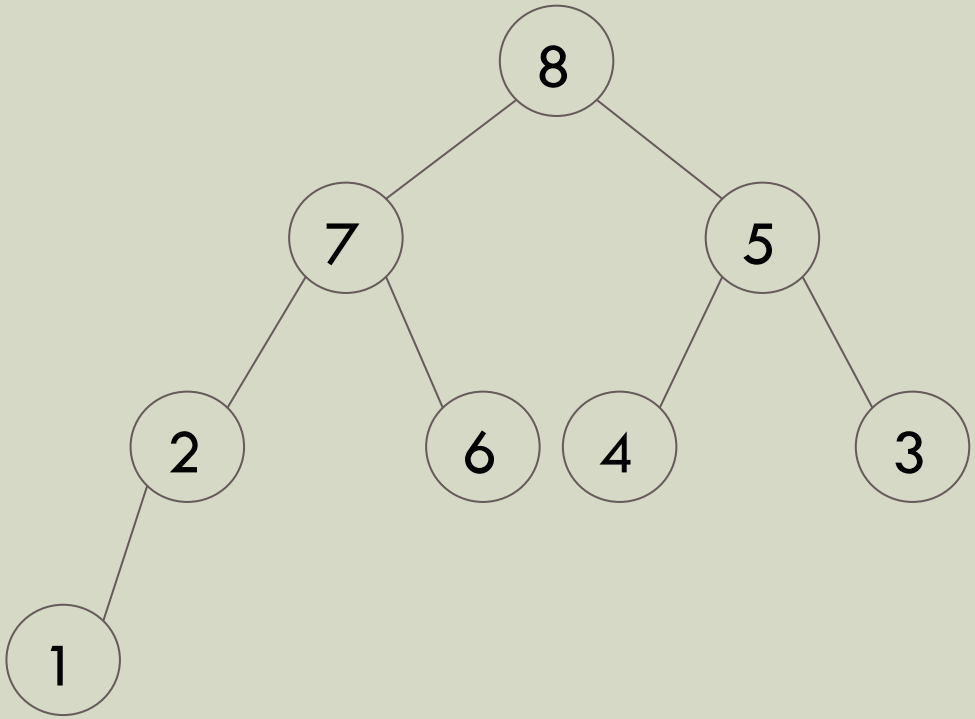
HEAP

- Propósito de uma fila de prioridades:
  - **acessar** o **elemento de maior prioridade** com custo **O(1)**.
- Para o nosso propósito (ordenação);
  - **maior elemento:**
    - **elemento de maior prioridade.**

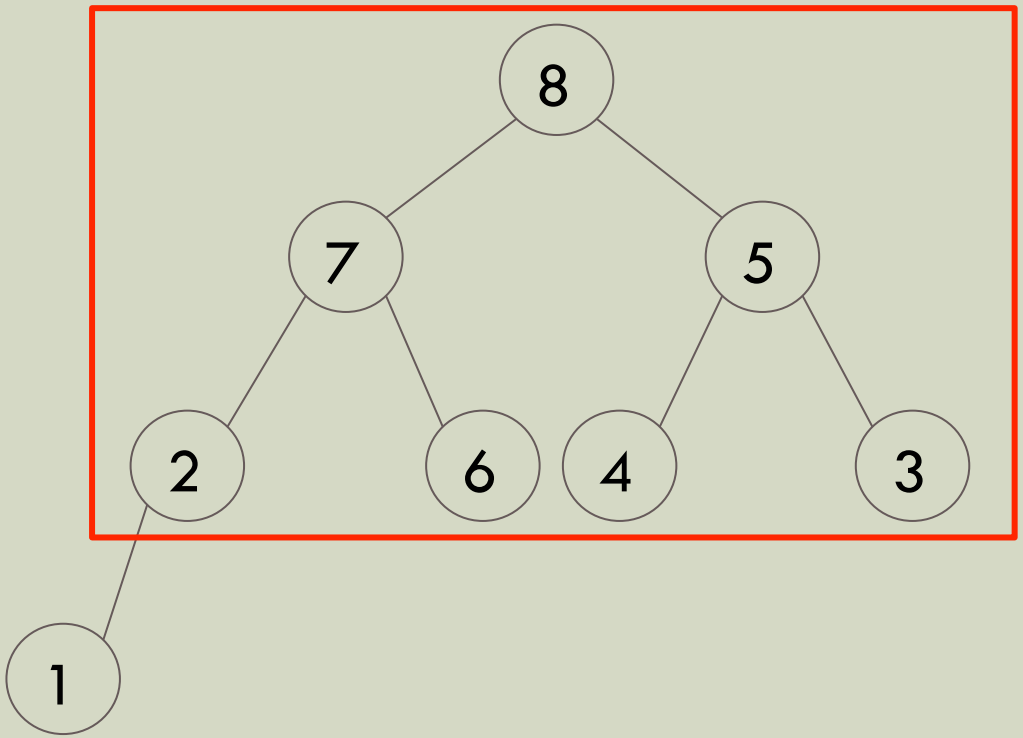
HEAP

- Em geral, implementada em um **vetor** ou em uma **árvore binária**, com as seguintes propriedades:
  - **até o penúltimo nível**, ela é **completa;**
  - no **último nível**, os **nós estão o mais à esquerda possível;**
  - **cada nó tem prioridade** em relação **a qualquer nó descendente.**

HEAP

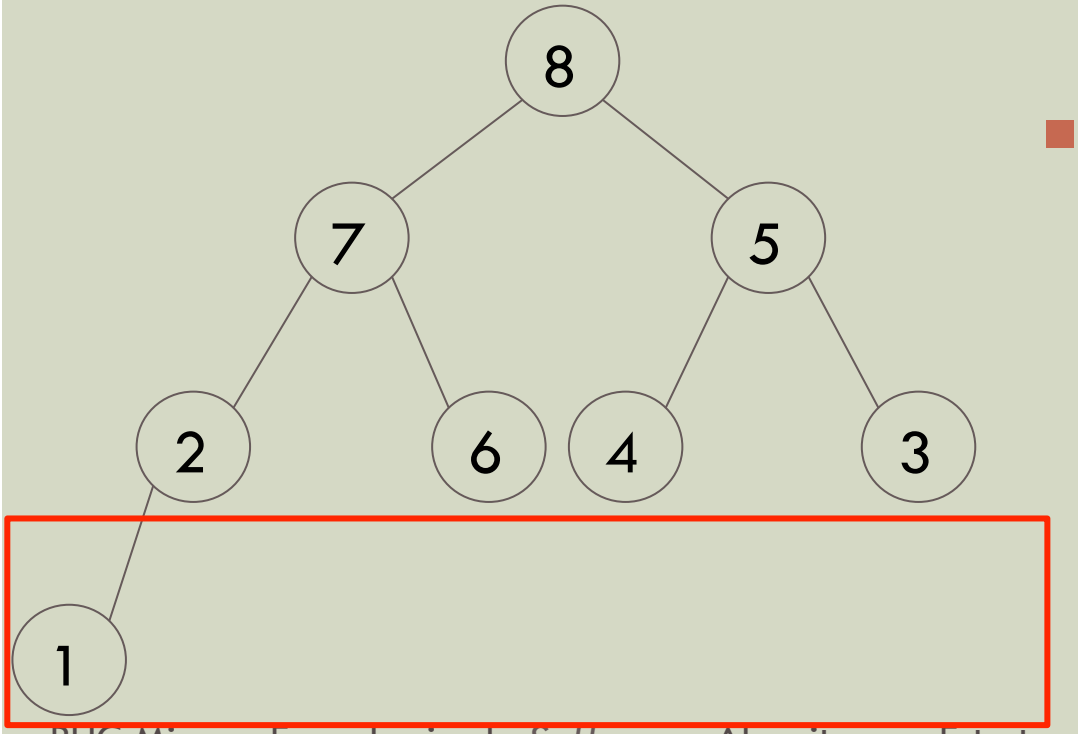


HEAP



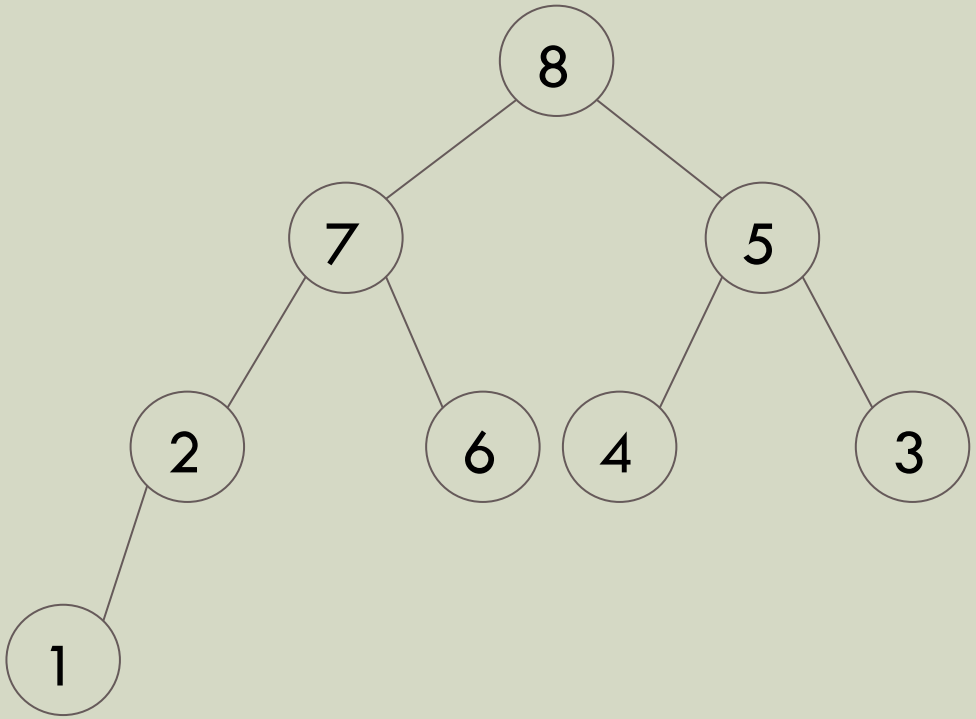
- **Até o penúltimo nível;**
  - ela é **completa.**

HEAP



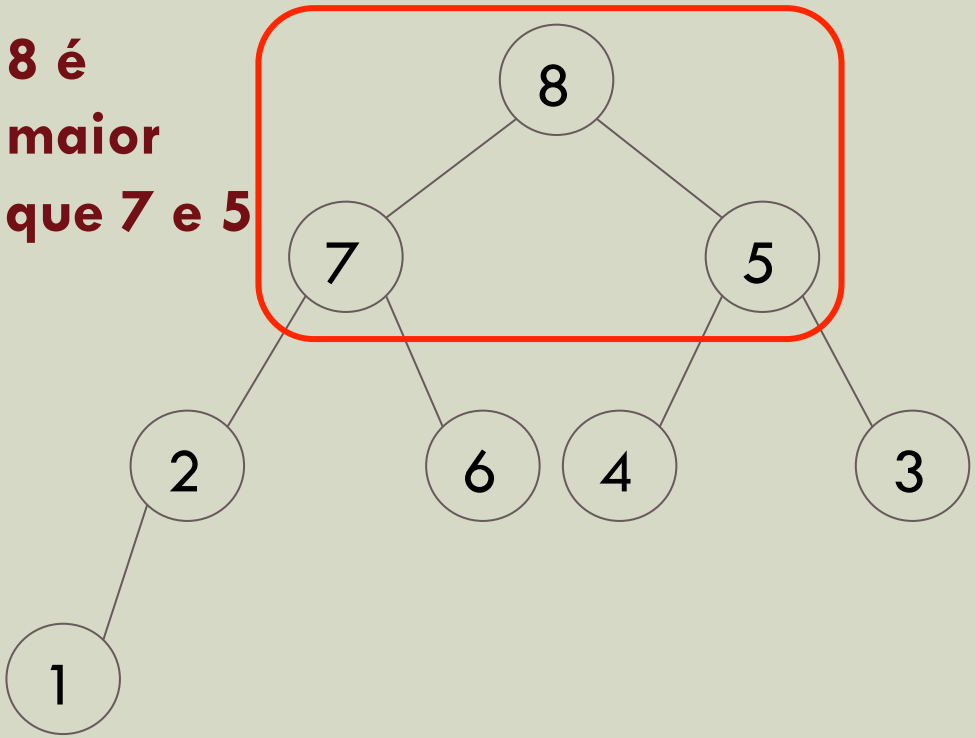
- No último nível;
  - os nós estão o mais à esquerda possível.

HEAP



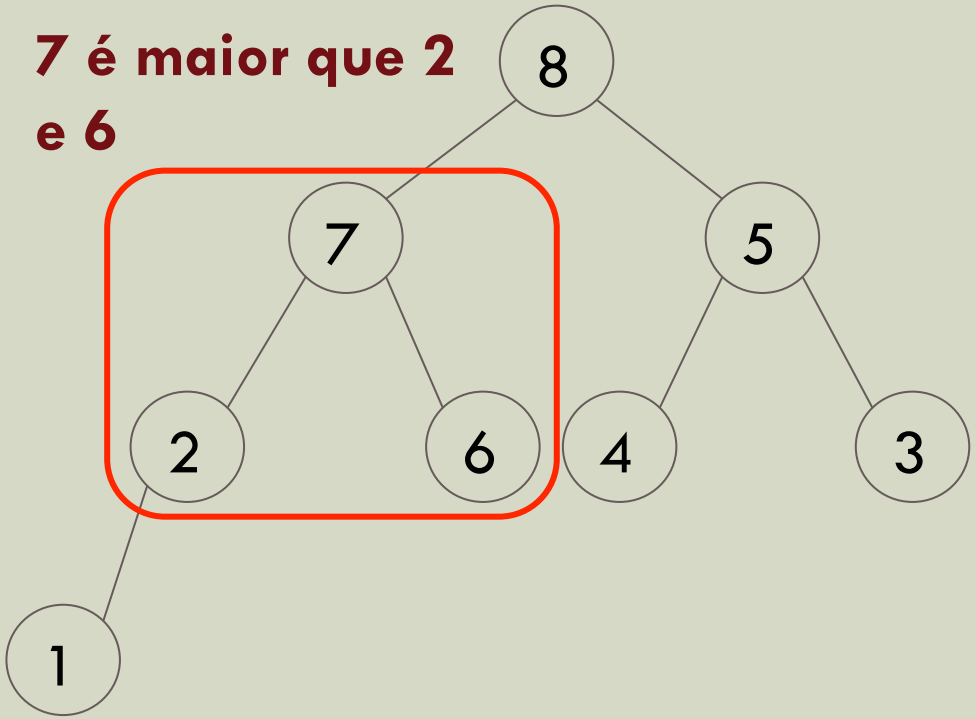
- **Cada nó tem prioridade;**
  - em relação a **qualquer nó descendente.**

HEAP



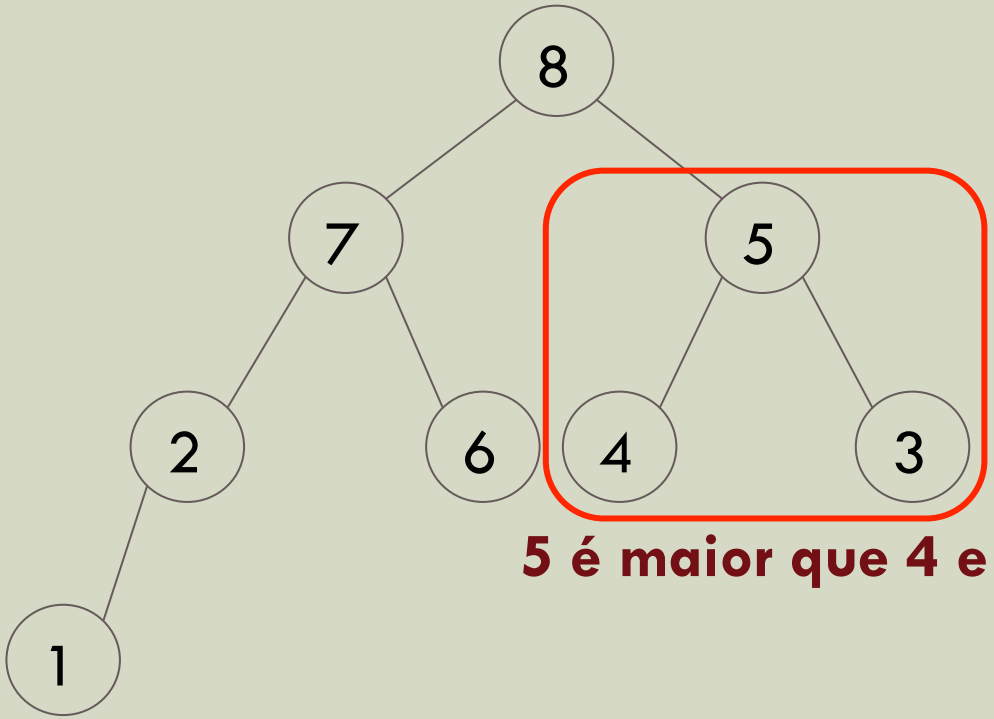
- **Cada nó tem prioridade;**
  - em relação a **qualquer nó descendente.**

HEAP



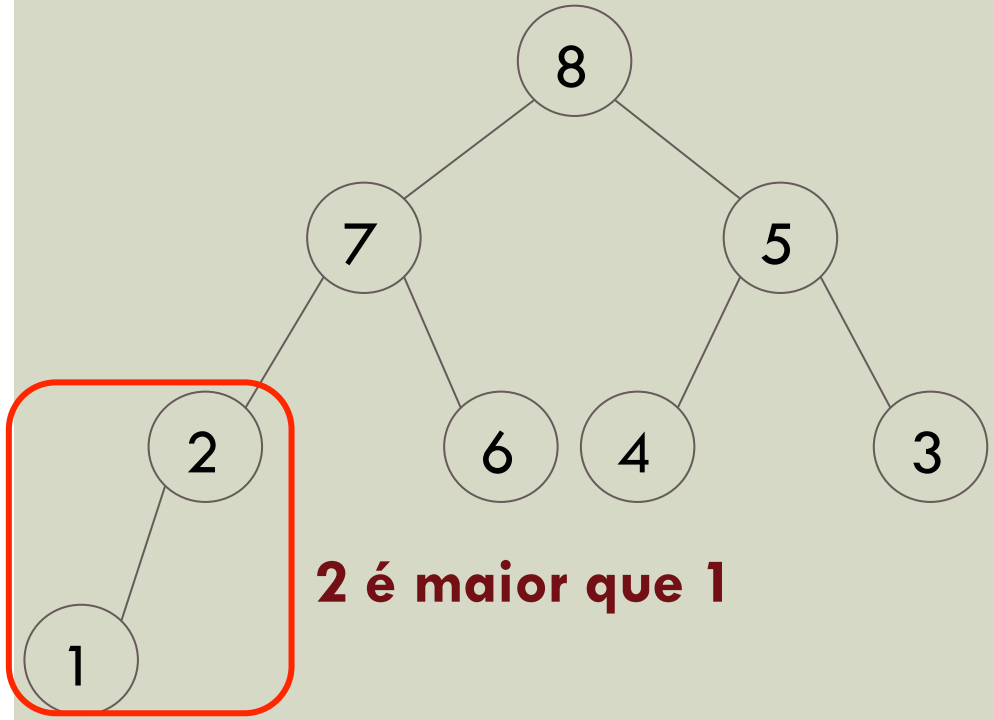
- **Cada nó tem prioridade;**
  - em relação a **qualquer nó descendente.**

HEAP



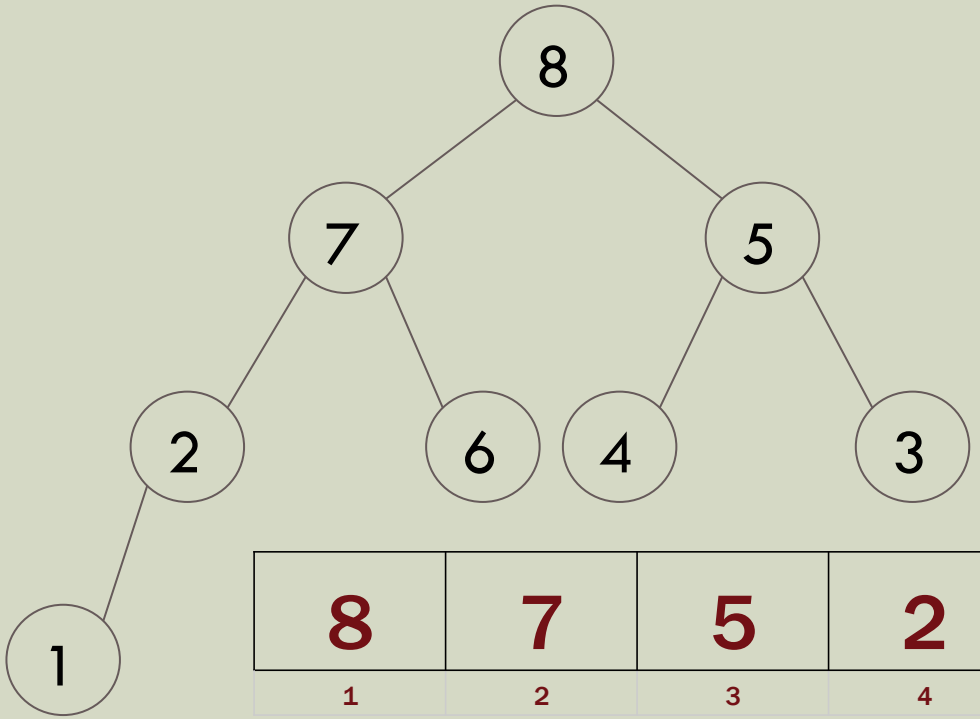
- **Cada nó tem prioridade;**
  - em relação a **qualquer nó descendente.**

HEAP



- **Cada nó tem prioridade;**
  - em relação a **qualquer nó descendente.**

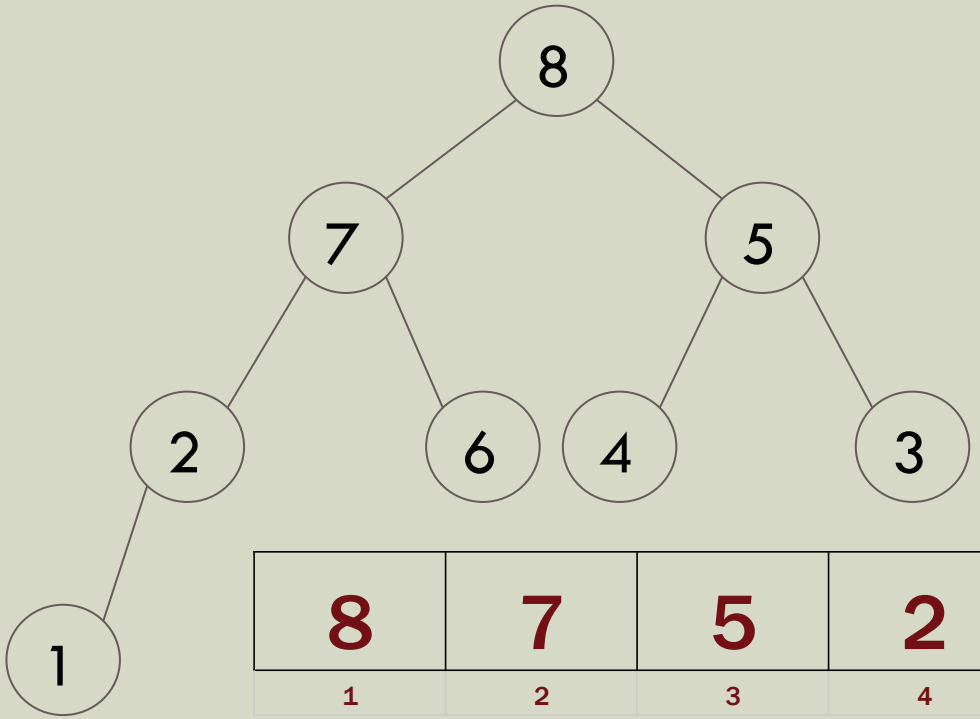
HEAP



- **Filho à esquerda:**
  - acessado com o **índice 2*i.**
- **Filho à direita:**
  - acessado com o **índice 2*i+1.**

8	7	5	2	6	4	3	1
1	2	3	4	5	6	7	8

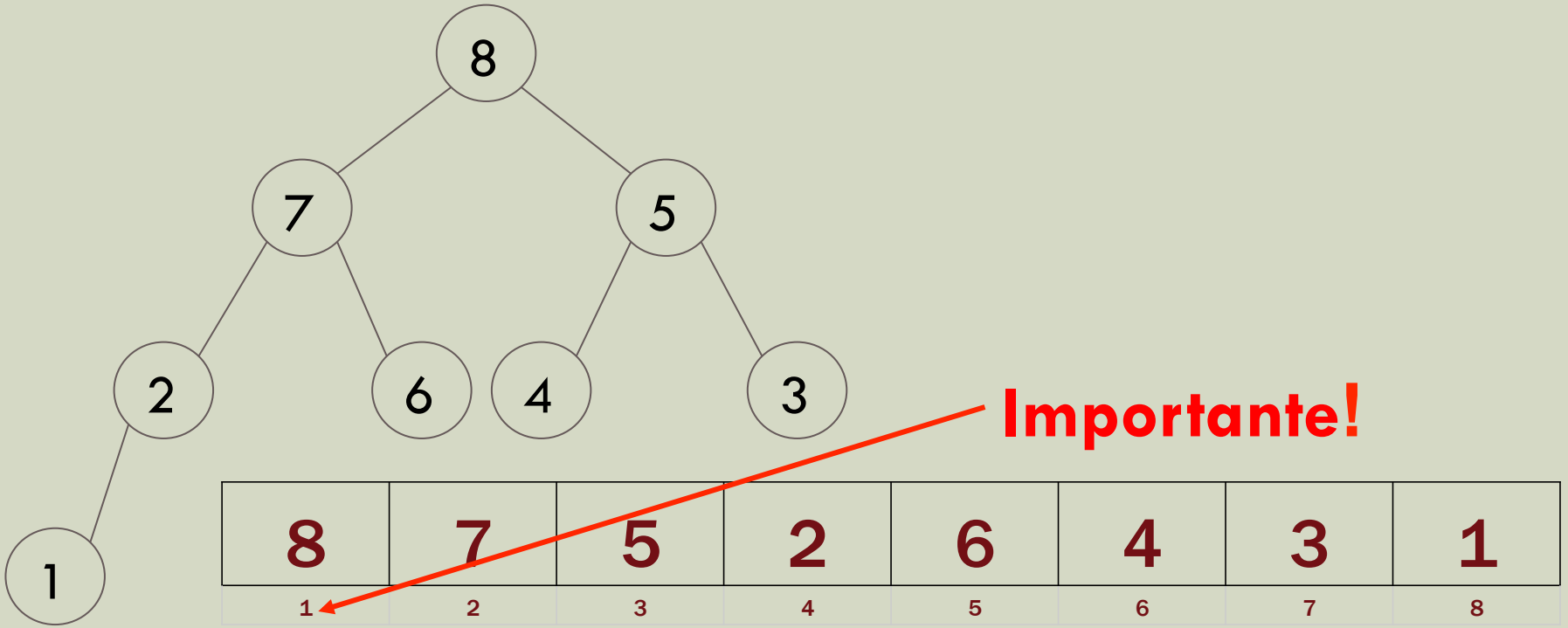
HEAP



- **Pai:**
  - acessado com **índice *i*/2.**

8	7	5	2	6	4	3	1
1	2	3	4	5	6	7	8

HEAP



HEAP

- É possível **acessar** o **maior elemento** em **tempo constante** por meio da *Heap*;
  - está sempre na **posição 1**.

HEAP

- É possível **acessar** o **maior elemento** em **tempo constante** por meio da *Heap*;
  - está sempre na **posição 1.**
- Mas como **construímos a *Heap*?**
- E como **restauramos** as **propriedades da *Heap*** após remover o maior elemento?

HEAP

- ***Heap*** é **construída estabelecendo-se suas propriedades** em **todos os nós;**
  - **de baixo para cima.**
- Isso é feito executando a rotina de **restauração das propriedades da *Heap*;**
  - **a partir do elemento *n*/2.**

RESTAURAÇÃO DA *HEAP*

```
restaura(i) {
    maior = i; esquerda = 2 * i;
    direita = 2 * i + 1;
    se array[esquerda] > array[maior] então
        maior = esquerda;
    se array[direita] > array[maior] então
        maior = direita;
    se (maior != i) então
        troca(i, maior);
        restaura(maior);
}
```

RESTAURAÇÃO DA *HEAP*

```
restaura(i) {
    maior = i; esquerda = 2 * i;
    direita = 2 * i + 1;
    se array[esquerda] > array[maior] então
        maior = esquerda;
    se array[direita] > array[maior] então
        maior = direita;
    se (maior != i) então
        troca(i, maior);
        restaura(maior);
}
```

RESTAURAÇÃO DA *HEAP*

```
restaura(i) {
    maior = i; esquerda = 2 * i;
    direita = 2 * i + 1;
    se array[esquerda] > array[maior] então
        maior = esquerda;
    se array[direita] > array[maior] então
        maior = direita;
    se (maior != i) então
        troca(i, maior);
        restaura(maior);
}
```

RESTAURAÇÃO DA *HEAP*

```
restaura(i) {
    maior = i; esquerda = 2 * i;
    direita = 2 * i + 1;
    se array[esquerda] > array[maior] então
        maior = esquerda;
    se array[direita] > array[maior] então
        maior = direita;
    se (maior != i) então
        troca(i, maior);
        restaura(maior);
}
```

RESTAURAÇÃO DA *HEAP*

```
restaura(i) {
    maior = i; esquerda = 2 * i;
    direita = 2 * i + 1;
    se array[esquerda] > array[maior] então
        maior = esquerda;
    se array[direita] > array[maior] então
        maior = direita;
    se (maior != i) então
        troca(i, maior);
        restaura(maior);
}
```

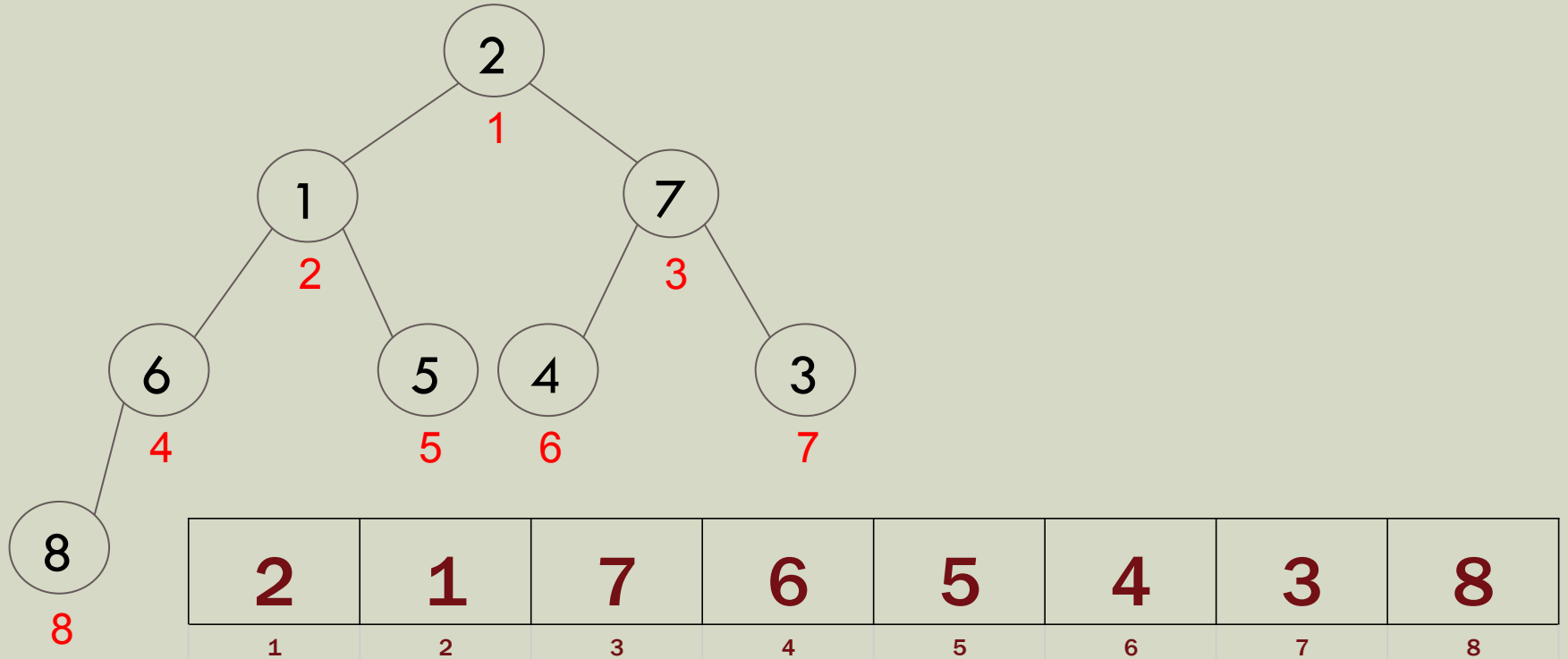
RESTAURAÇÃO DA *HEAP*

```
restaura(i) {
    maior = i; esquerda = 2 * i;
    direita = 2 * i + 1;
    se array[esquerda] > array[maior] então
        maior = esquerda;
    se array[direita] > array[maior] então
        maior = direita;
    se (maior != i) então
        troca(i, maior);
        restaura(maior);
}
```

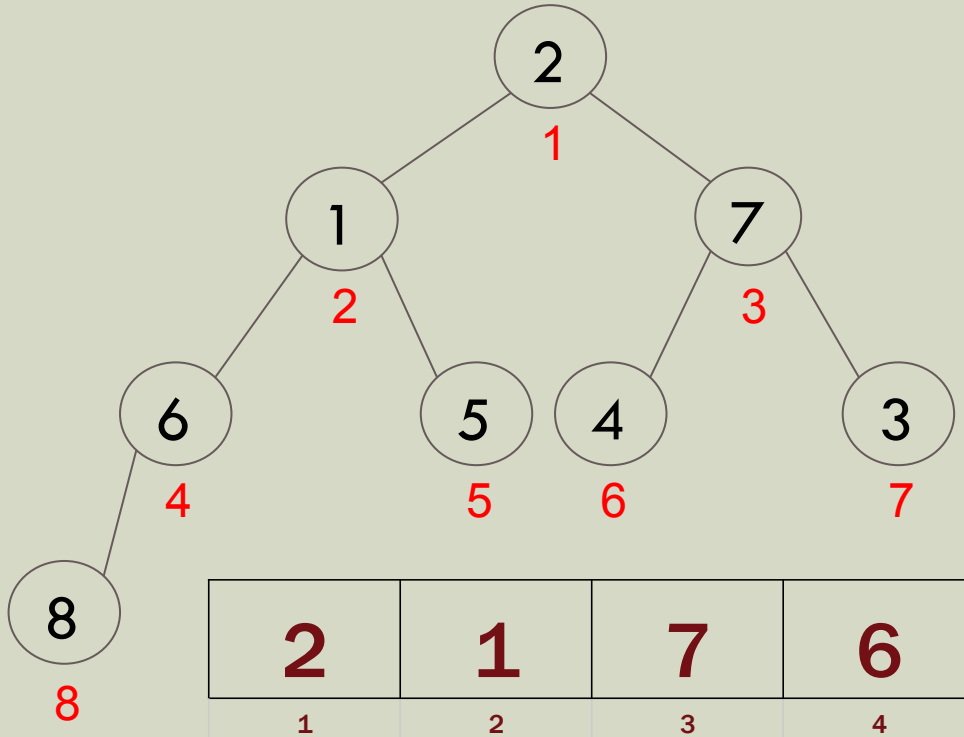
RESTAURAÇÃO DA *HEAP*

```
restaura(i) {
    maior = i; esquerda = 2 * i;
    direita = 2 * i + 1;
    se array[esquerda] > array[maior] então
        maior = esquerda;
    se array[direita] > array[maior] então
        maior = direita;
    se (maior != i) então
        troca(i, maior);
        restaura(maior);
}
```

CONSTRUÇÃO DA *HEAP*



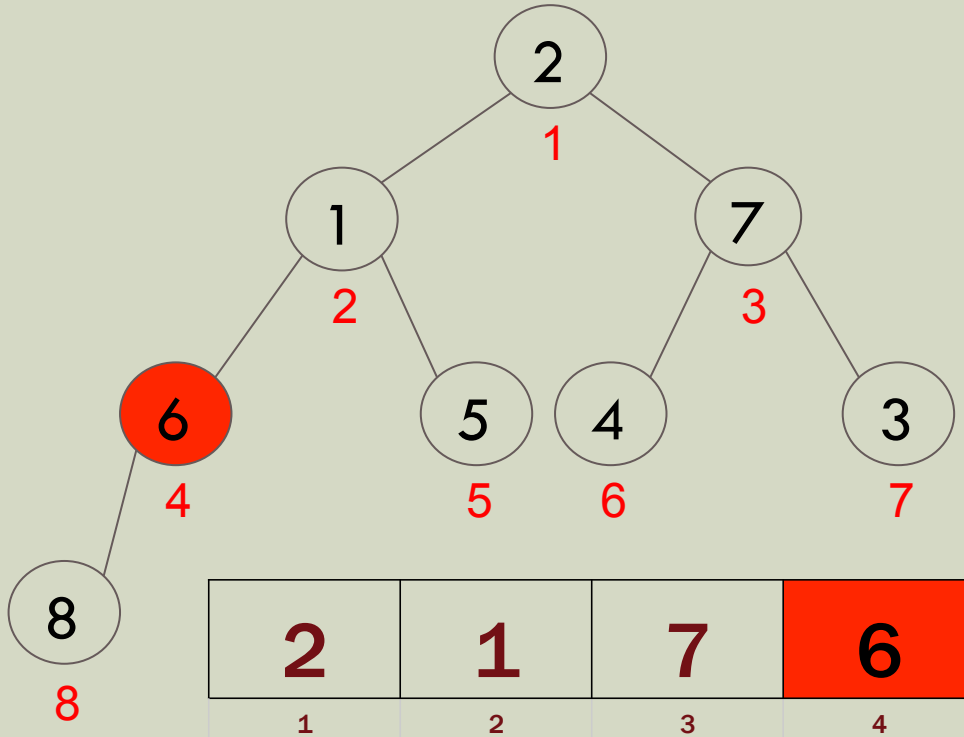
CONSTRUÇÃO DA *HEAP*



- Construindo a *Heap* a partir de $i = n/2$.

2	1	7	6	5	4	3	8
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*

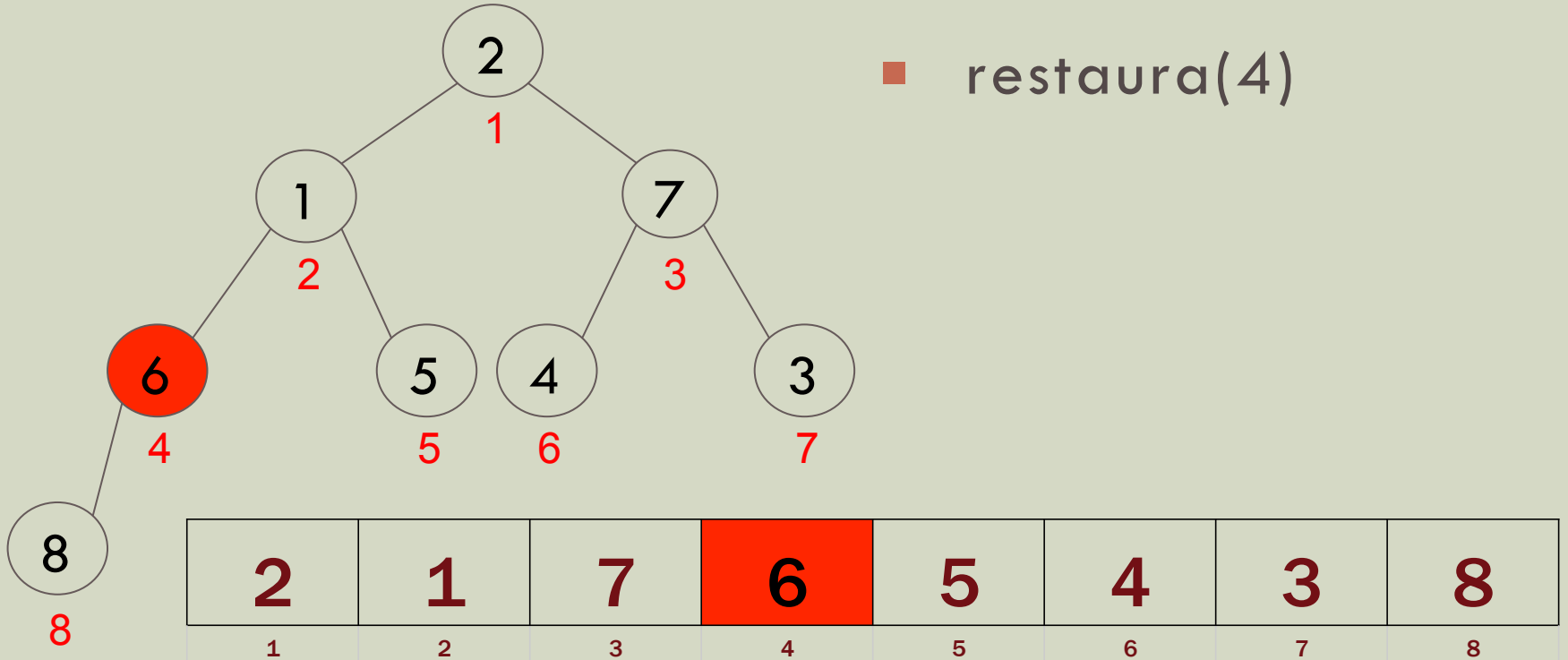


- Construindo a *Heap* a partir de $i = 4$.

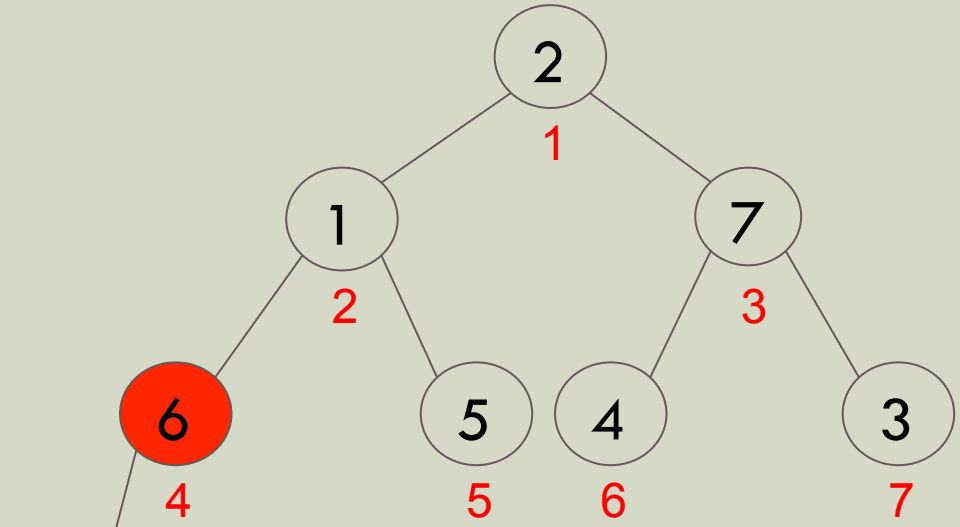
2	1	7	6	5	4	3	8
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*

- restaura(4)



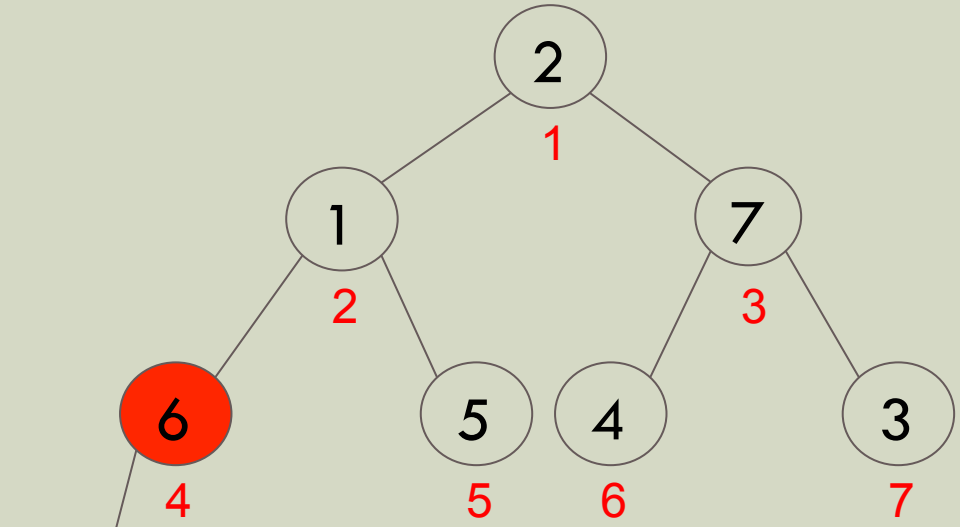
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]

2	1	7	6	5	4	3	8
1	2	3	4	5	6	7	8

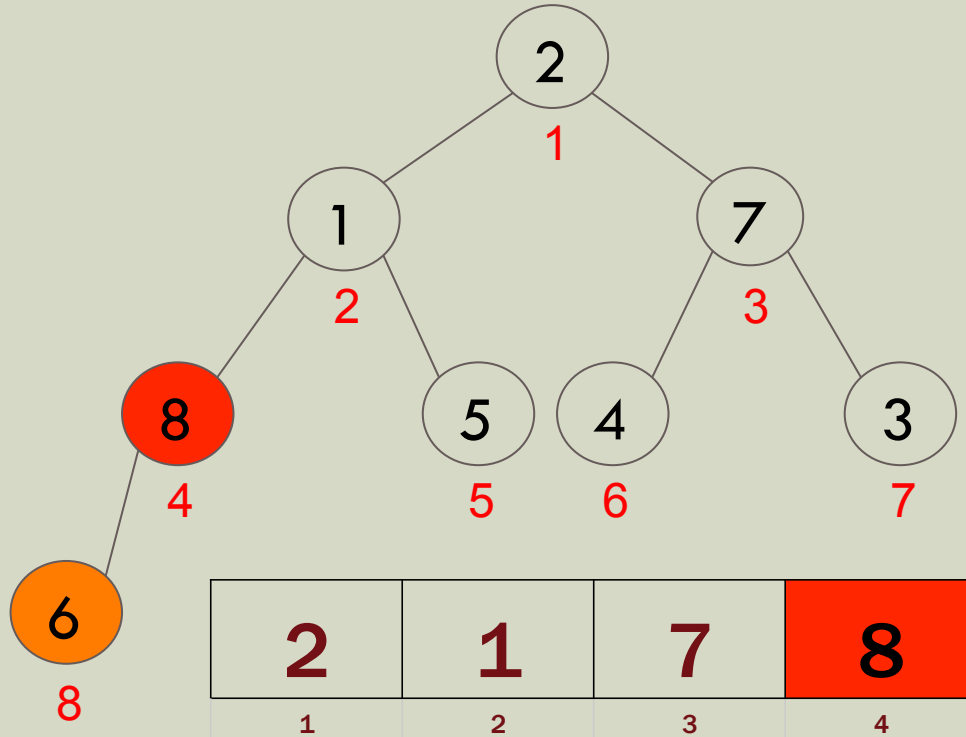
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]
  - troca(8, 4)

2	1	7	6	5	4	3	8
1	2	3	4	5	6	7	8

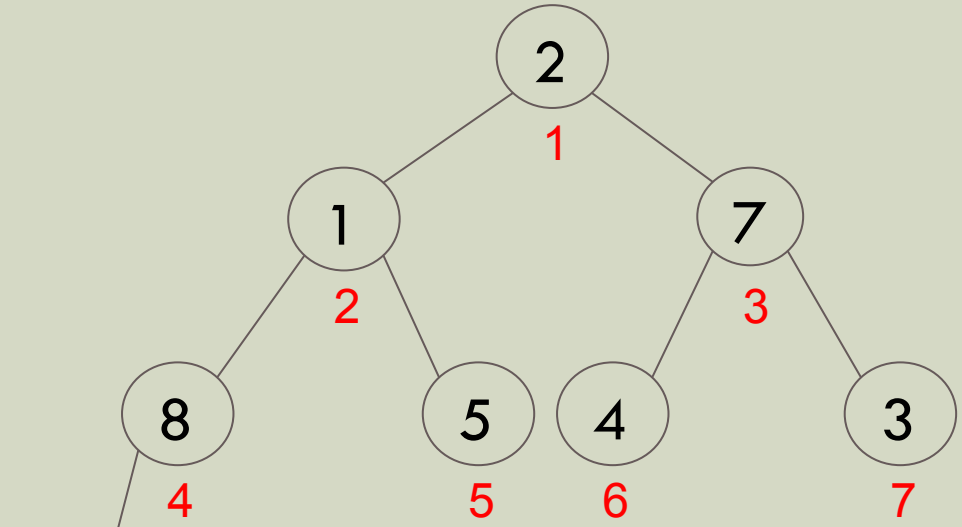
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]
  - troca(8, 4)

2	1	7	8	5	4	3	6
1	2	3	4	5	6	7	8

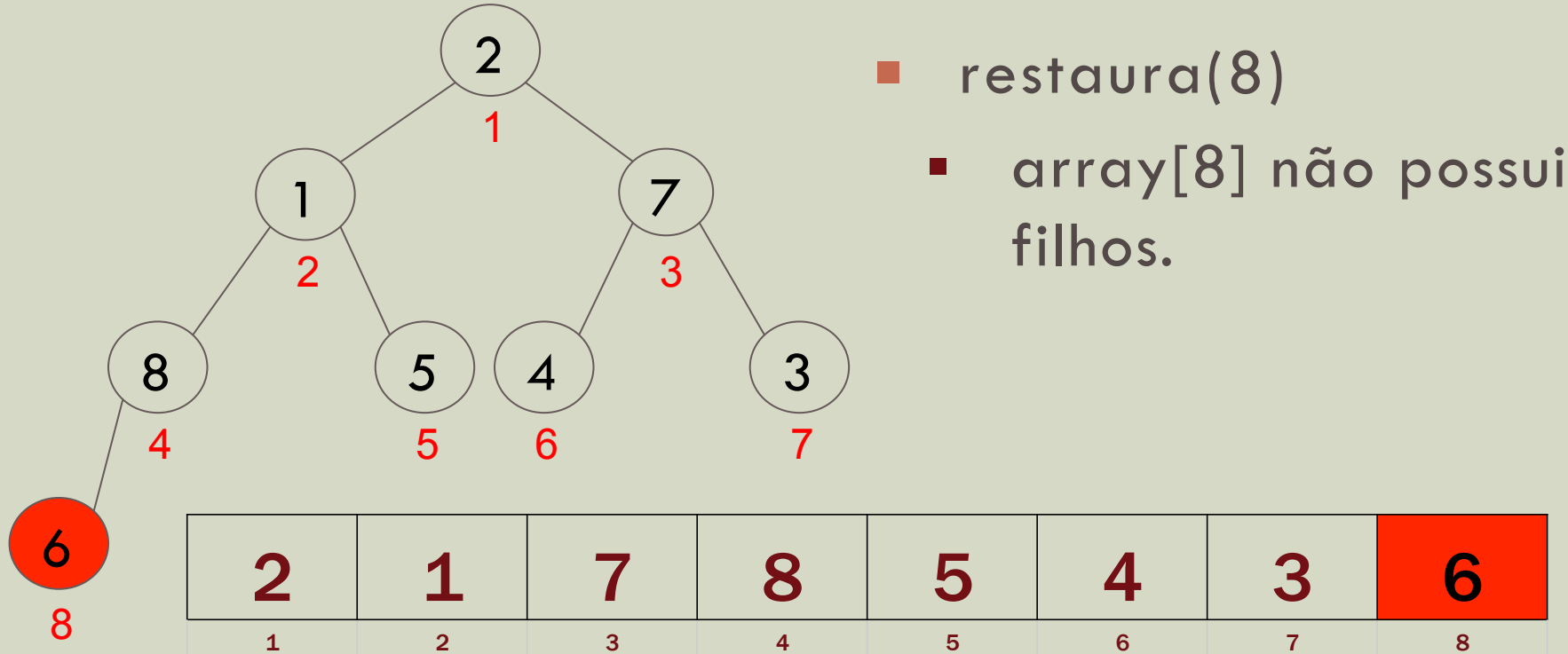
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]
  - troca(8, 4)
  - restaura(8)

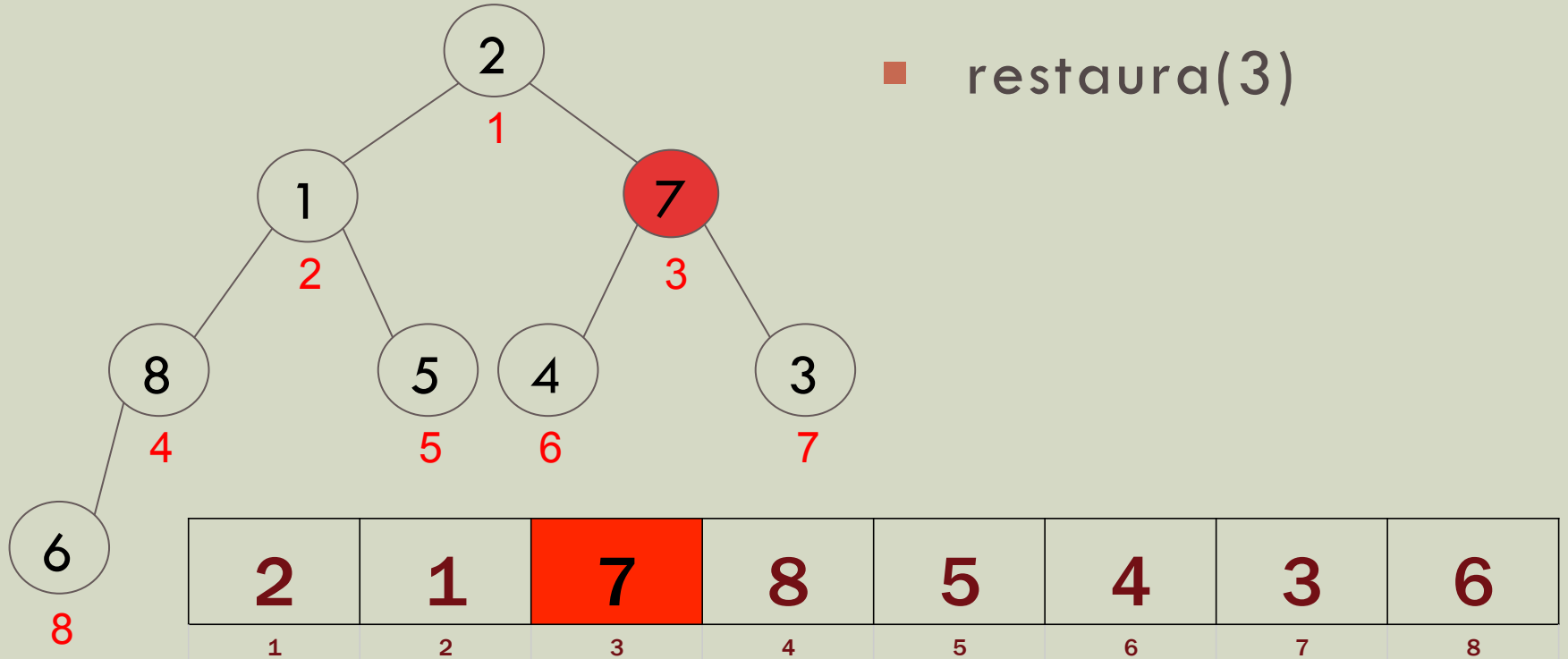
2	1	7	8	5	4	3	6
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*



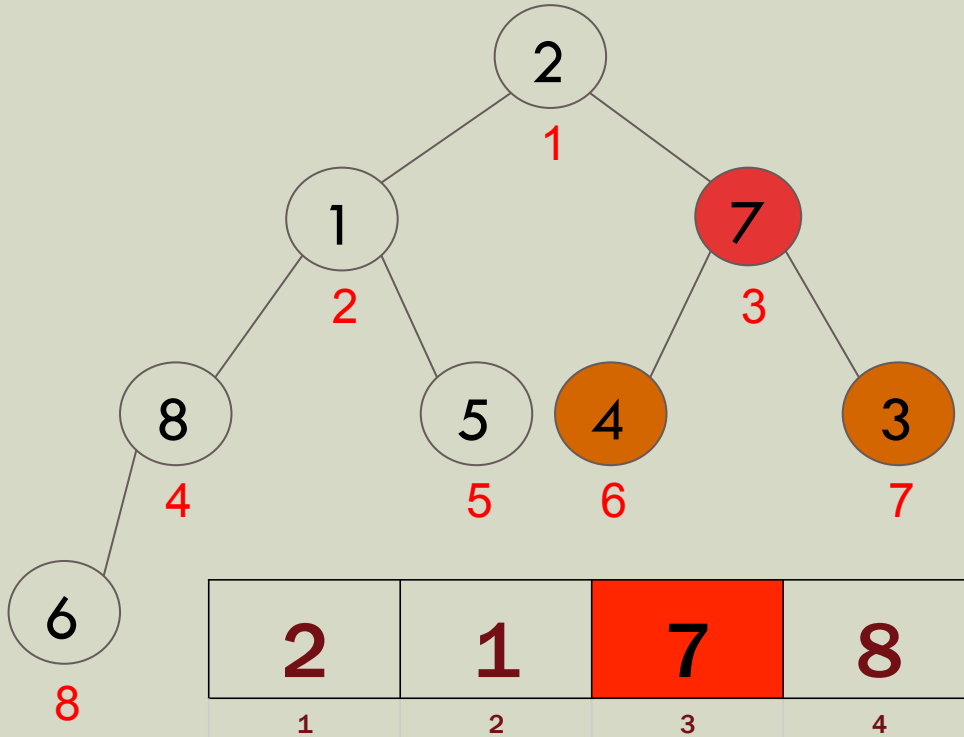
- restaura(8)
  - array[8] não possui filhos.

CONSTRUÇÃO DA *HEAP*



- restaura(3)

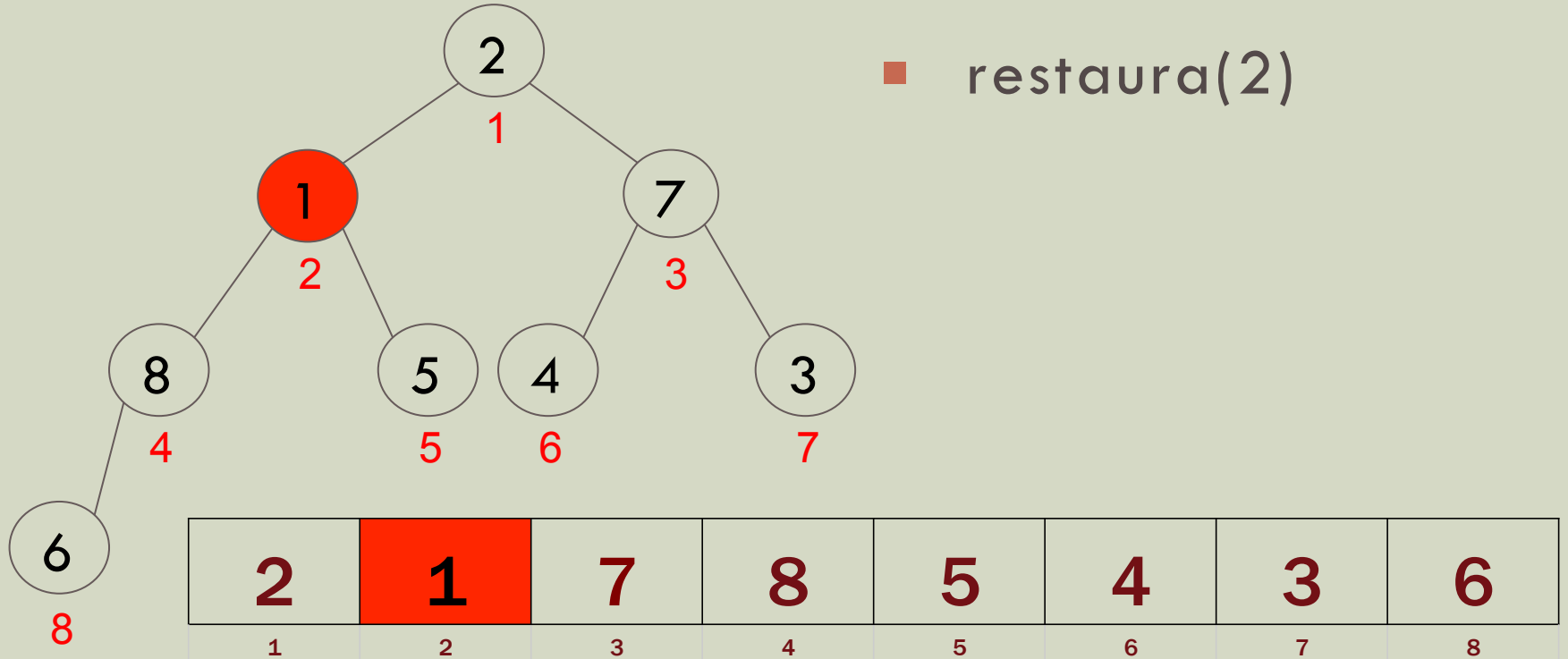
CONSTRUÇÃO DA *HEAP*



- restaura(3)
  - array[3] > array[6]
  - array[3] > array[7]

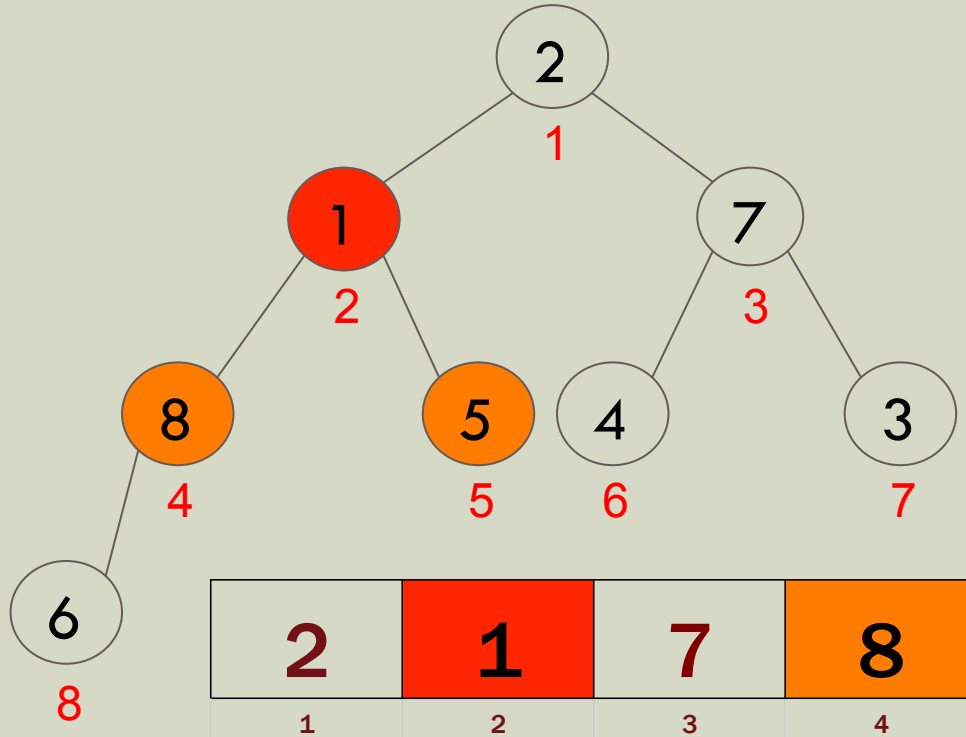
2	1	7	8	5	4	3	6
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*



- restaura(2)

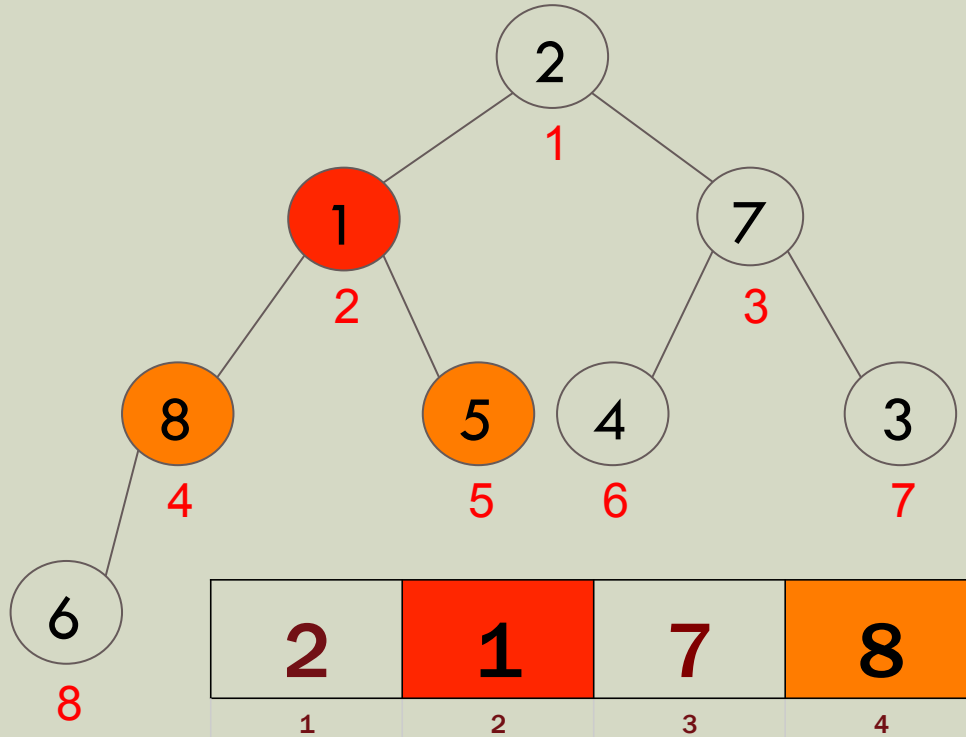
CONSTRUÇÃO DA *HEAP*



- restaura(2)
  - array[4] > array[2]

2	1	7	8	5	4	3	6
1	2	3	4	5	6	7	8

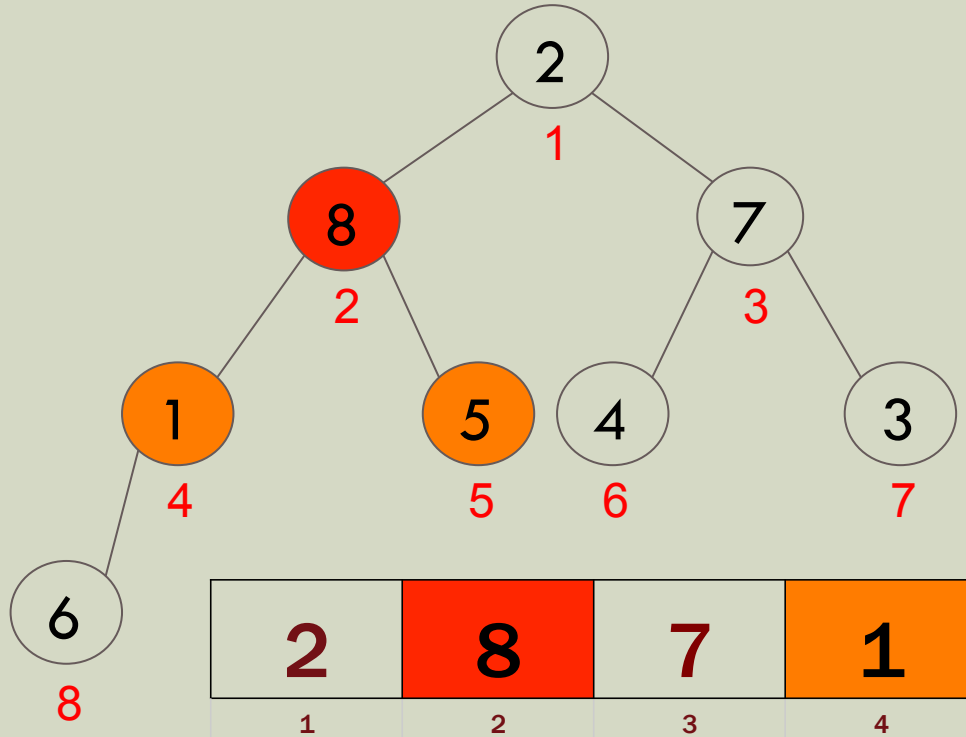
CONSTRUÇÃO DA *HEAP*



- restaura(2)
  - array[4] > array[2]
  - troca(4, 2)

2	1	7	8	5	4	3	6
1	2	3	4	5	6	7	8

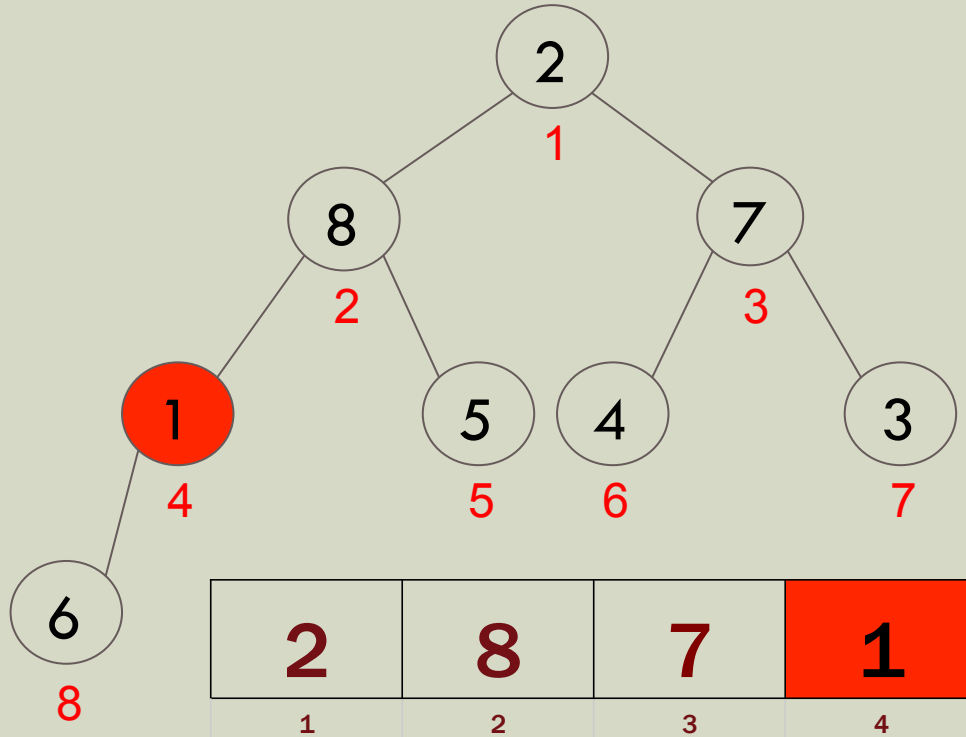
CONSTRUÇÃO DA *HEAP*



- restaura(2)
  - array[4] > array[2]
  - troca(4, 2)

2	8	7	1	5	4	3	6
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*

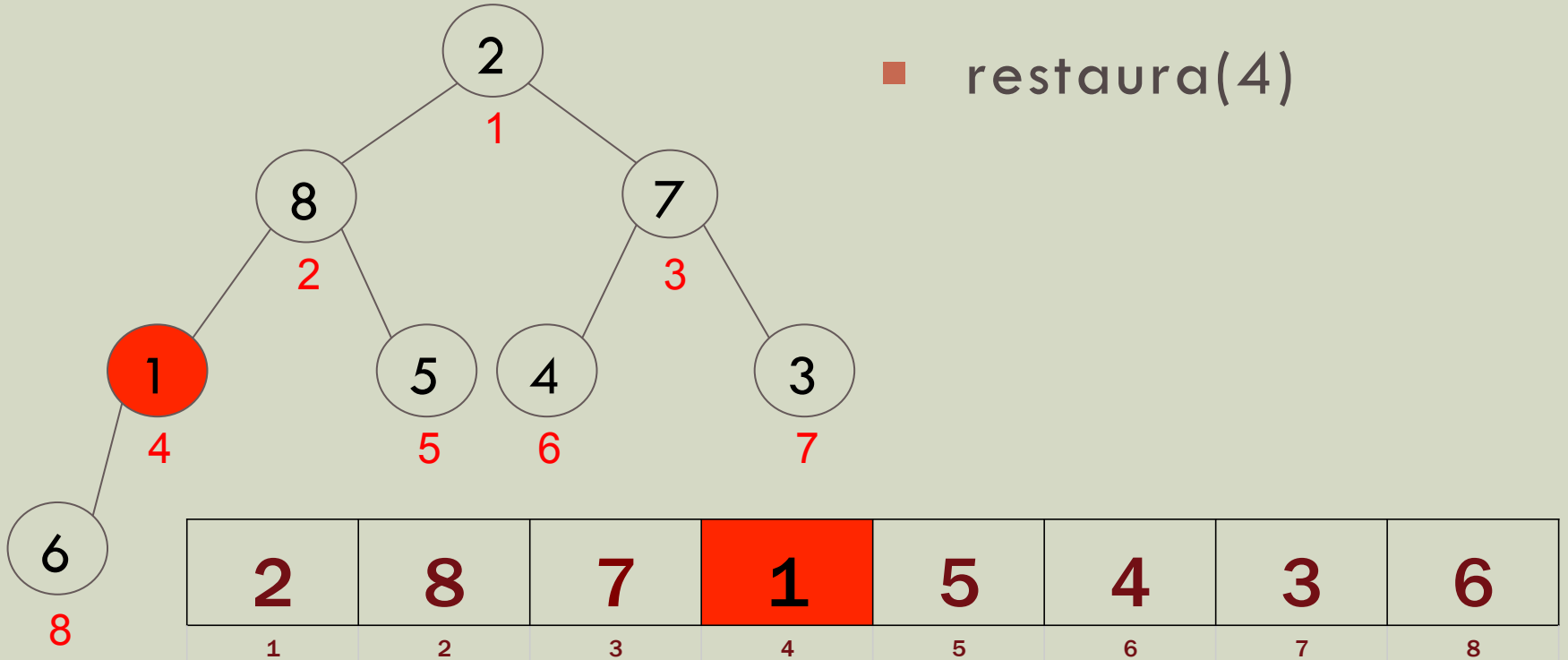


- restaura(2)
  - array[4] > array[2]
  - troca(4, 2)
  - restaura(4)

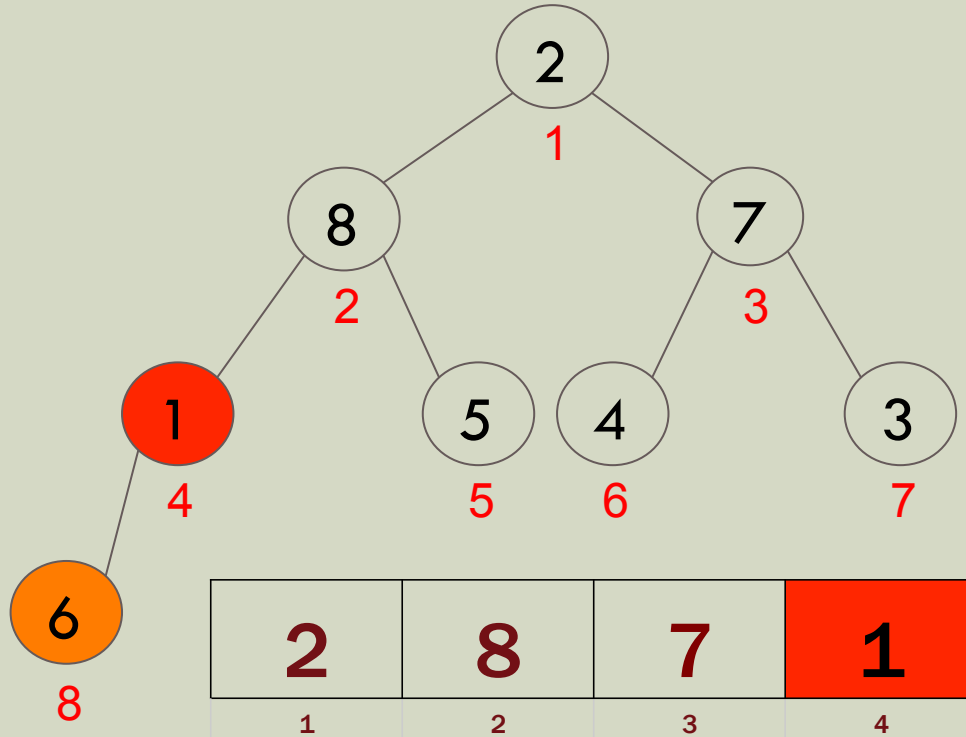
2	8	7	1	5	4	3	6
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*

- restaura(4)



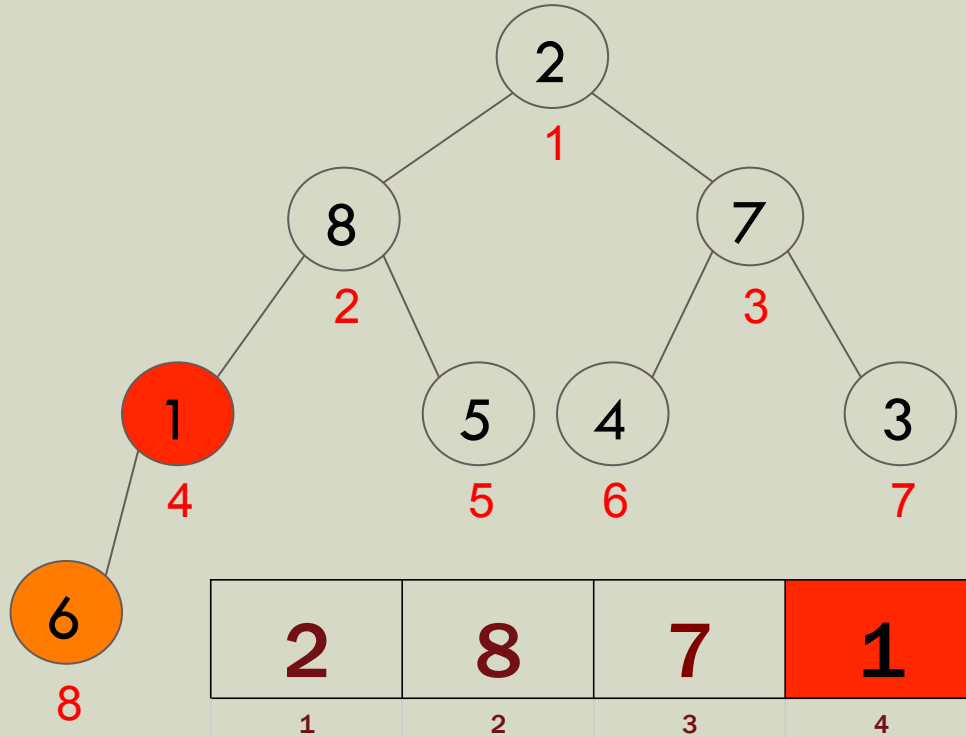
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]

2	8	7	1	5	4	3	6
1	2	3	4	5	6	7	8

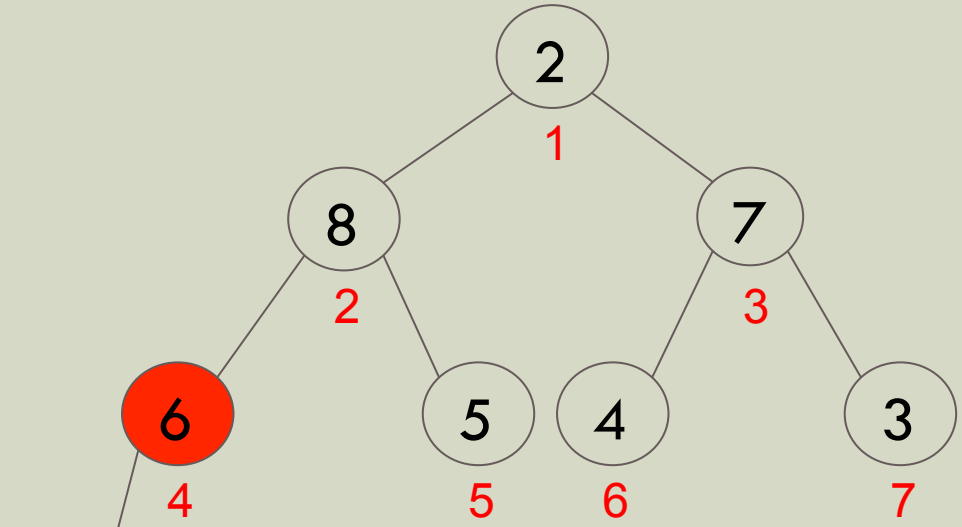
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]
  - troca(8, 4)

2	8	7	1	5	4	3	6
1	2	3	4	5	6	7	8

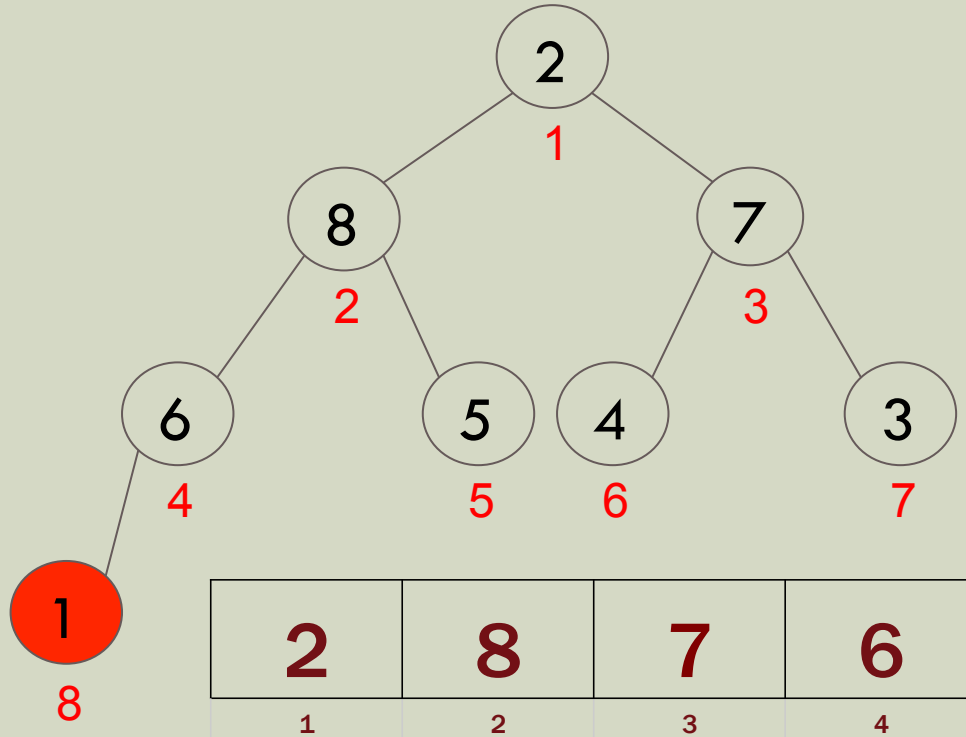
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]
  - troca(8, 4)

2	8	7	6	5	4	3	1
1	2	3	4	5	6	7	8

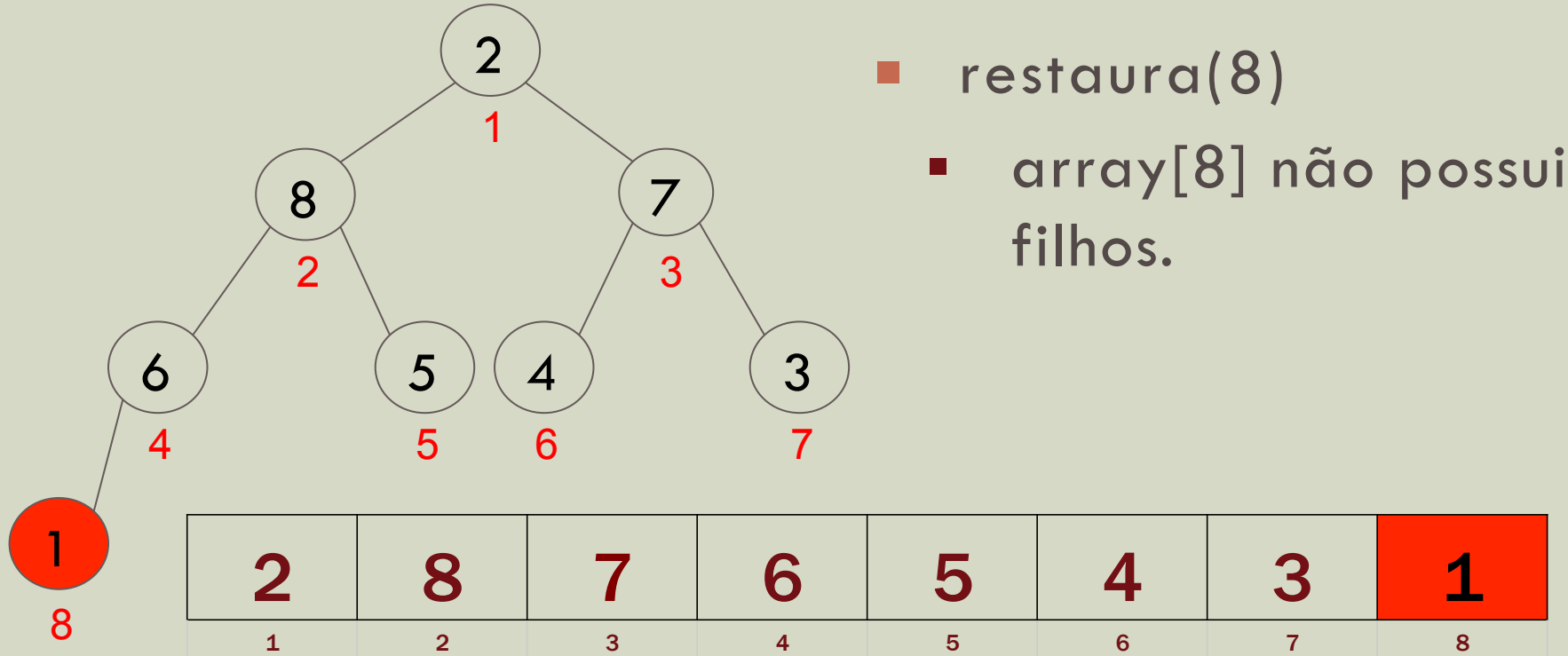
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[8] > array[4]
  - troca(8, 4)
  - restaura(8)

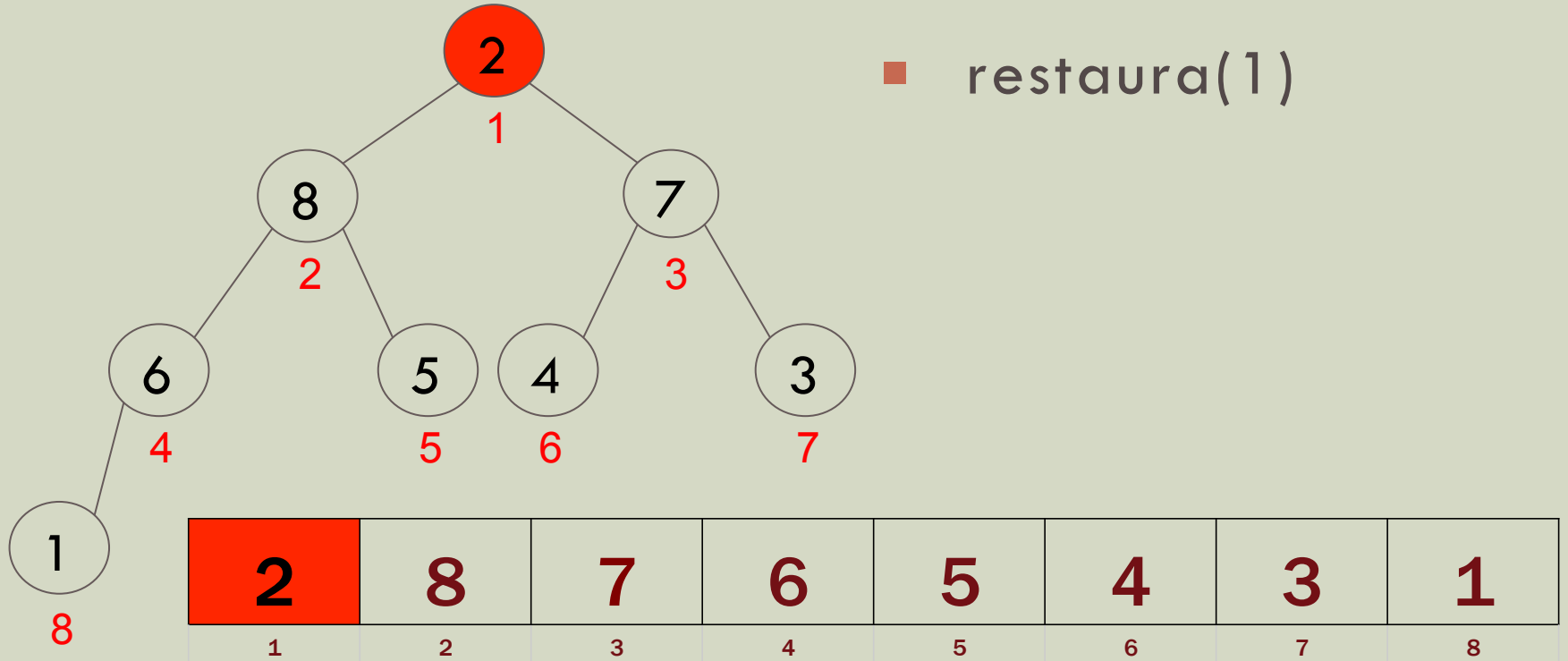
2	8	7	6	5	4	3	1
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*



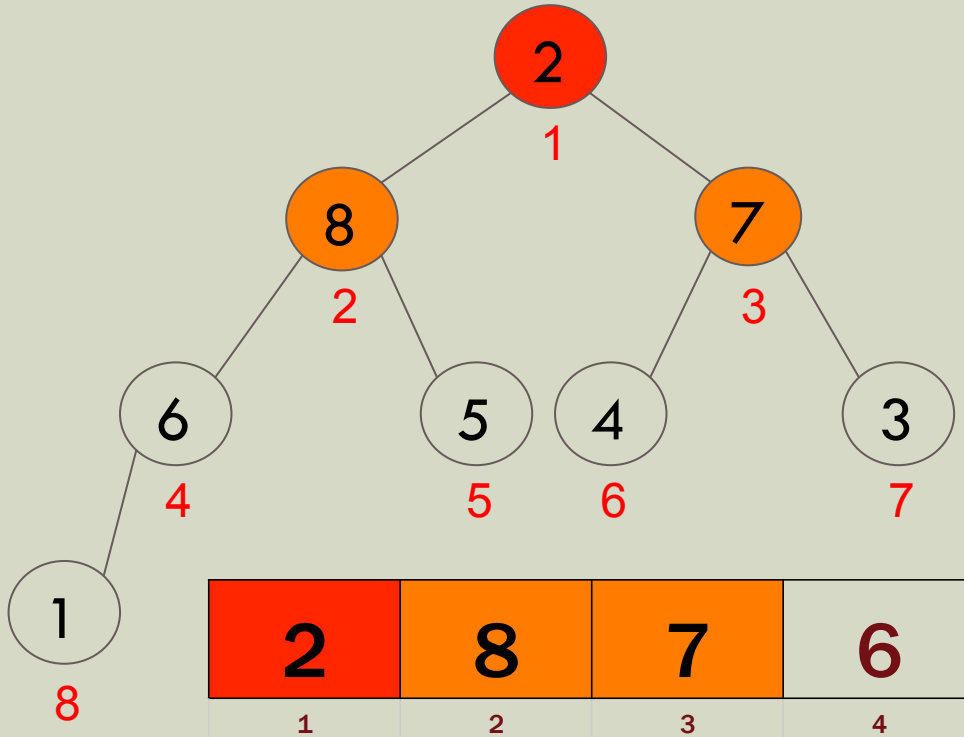
- restaura(8)
  - array[8] não possui filhos.

CONSTRUÇÃO DA *HEAP*



- restaura(1)

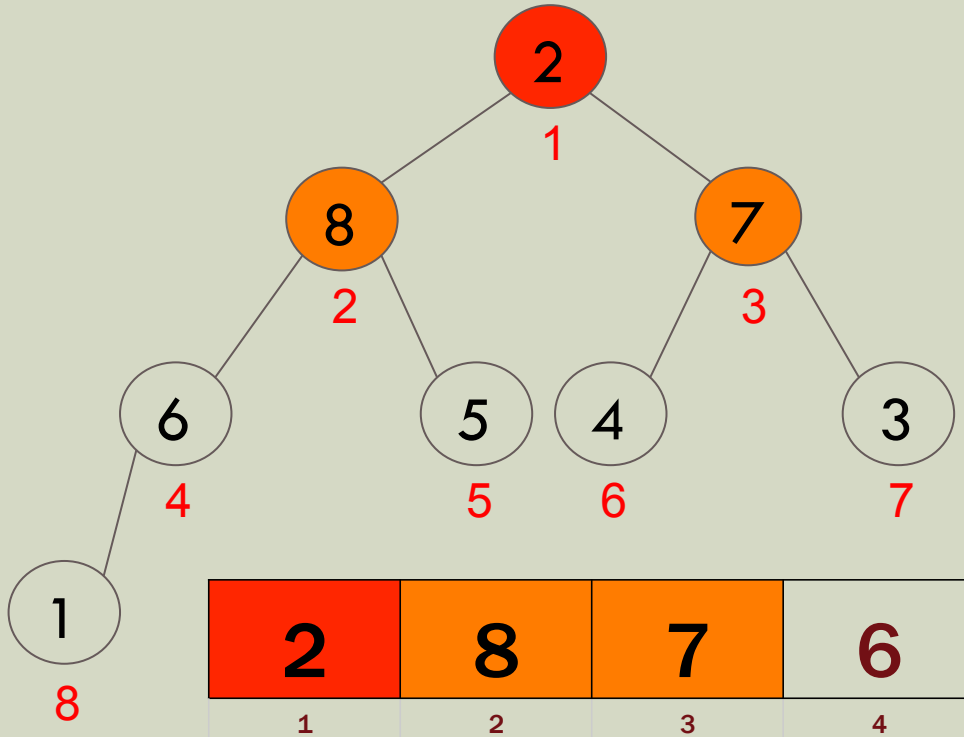
CONSTRUÇÃO DA *HEAP*



- restaura(1)
  - array[2] > array[1]

2	8	7	6	5	4	3	1
1	2	3	4	5	6	7	8

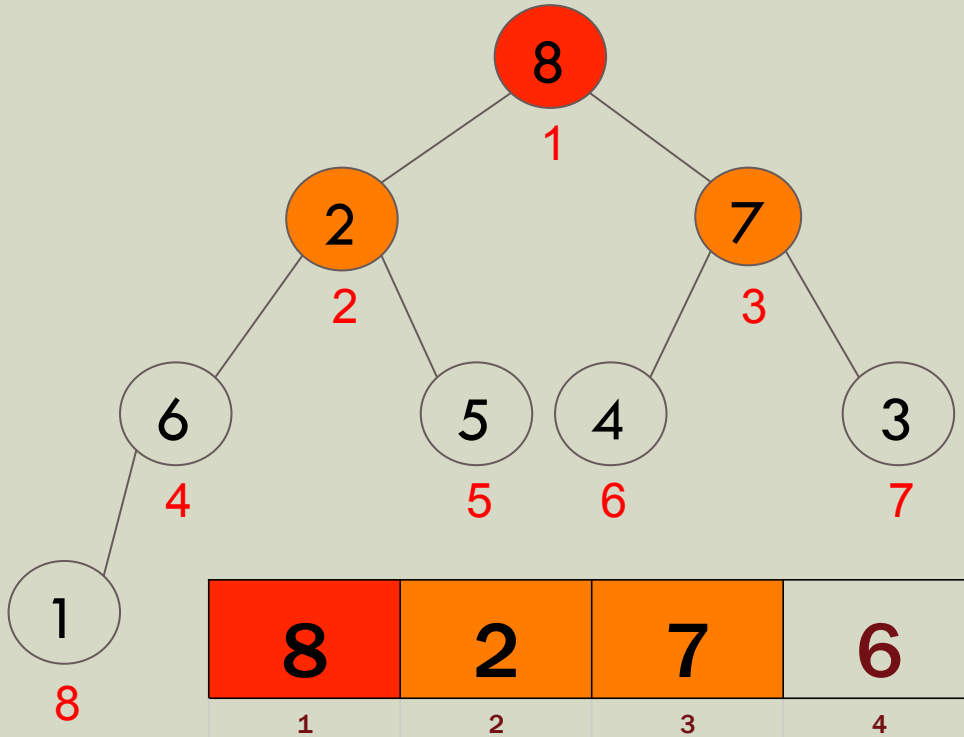
CONSTRUÇÃO DA *HEAP*



- restaura(1)
  - array[2] > array[1]
  - troca(2, 1)

2	8	7	6	5	4	3	1
1	2	3	4	5	6	7	8

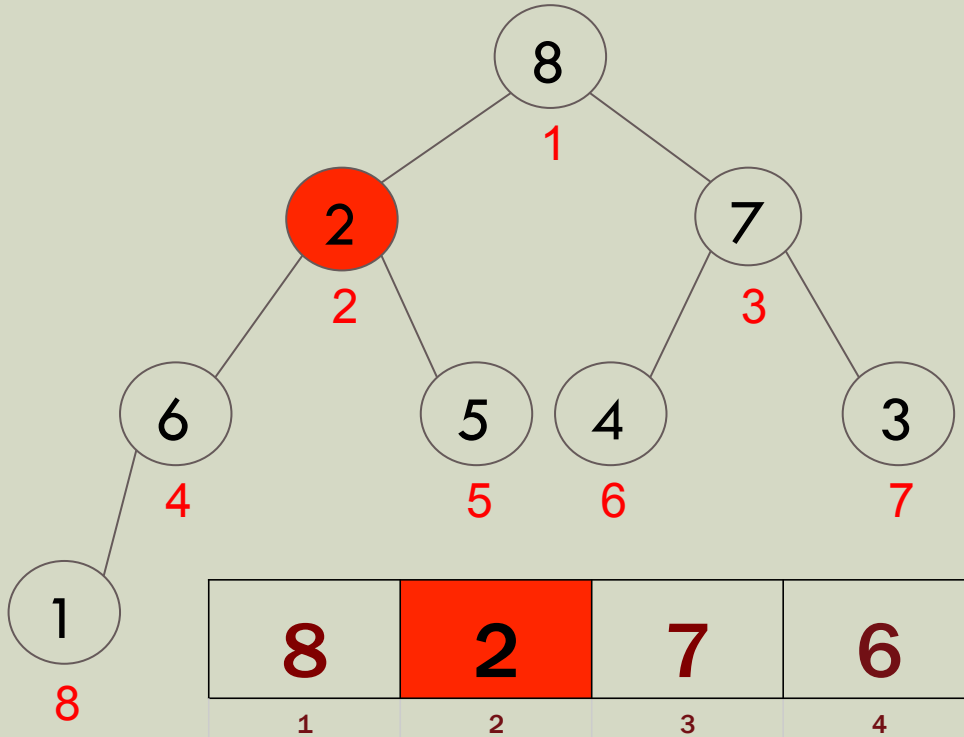
CONSTRUÇÃO DA *HEAP*



- restaura(1)
  - array[2] > array[1]
  - troca(2, 1)

8	2	7	6	5	4	3	1
1	2	3	4	5	6	7	8

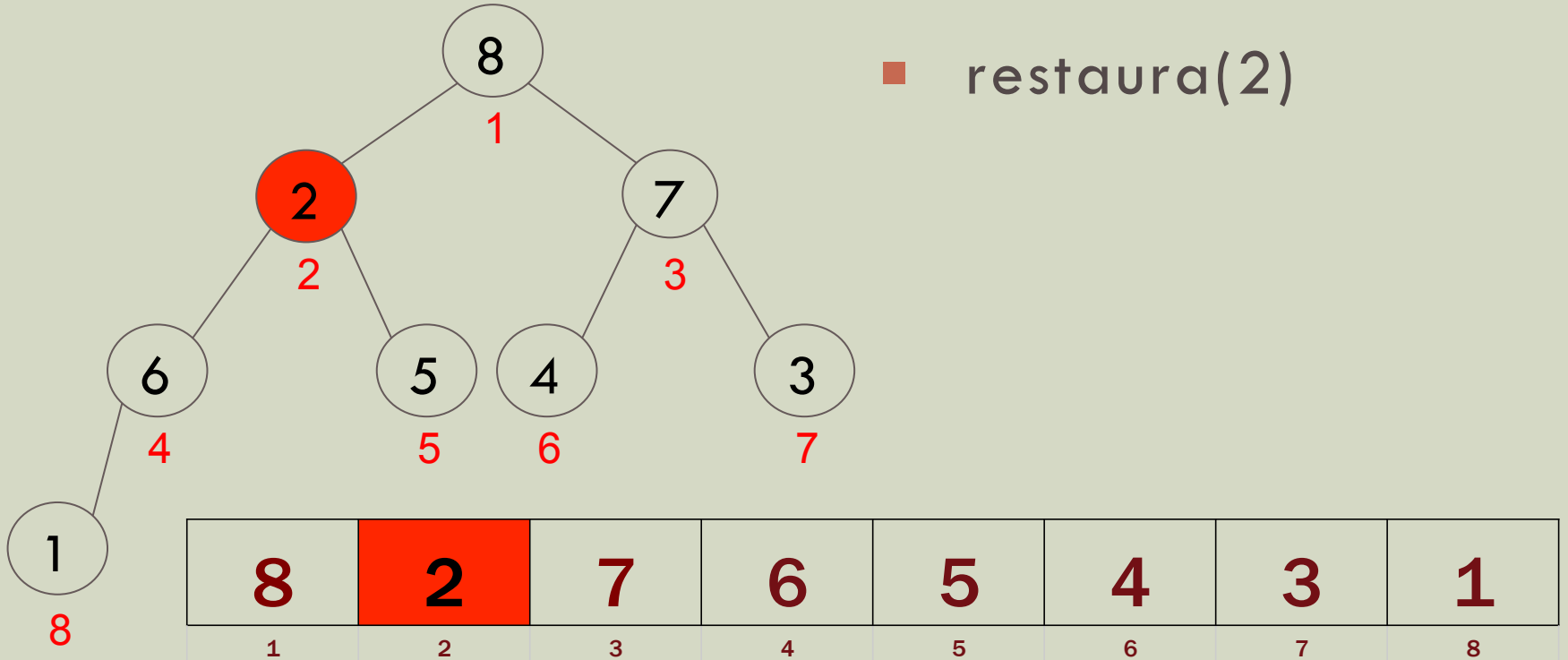
CONSTRUÇÃO DA *HEAP*



- restaura(1)
  - array[2] > array[1]
  - troca(2, 1)
  - restaura(2)

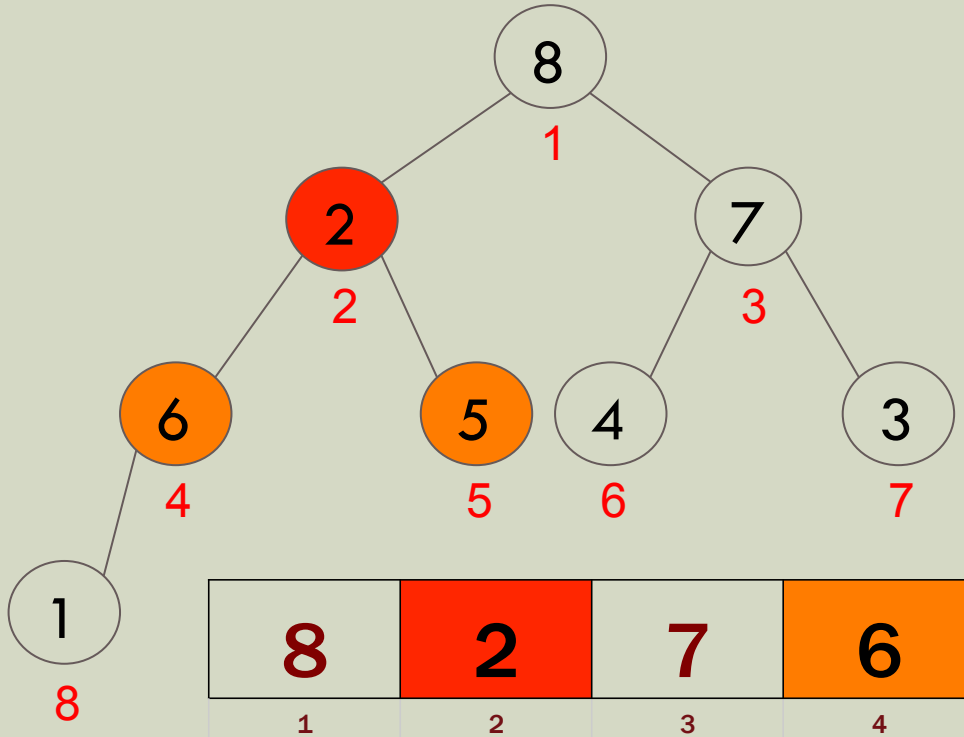
8	2	7	6	5	4	3	1
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*



- restaura(2)

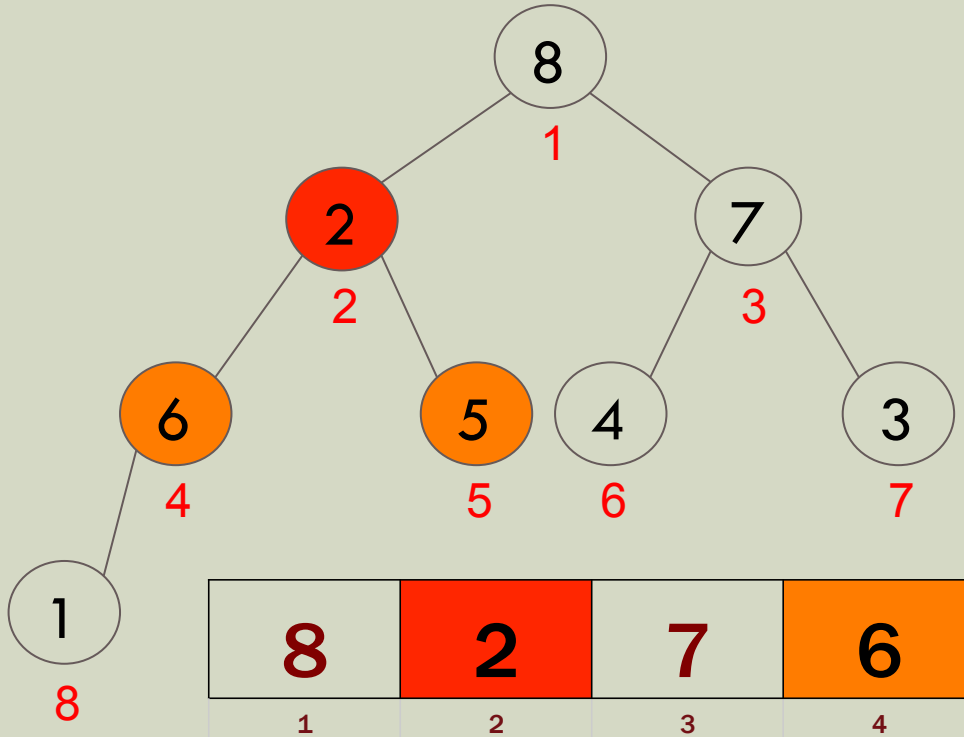
CONSTRUÇÃO DA *HEAP*



- restaura(2)
  - array[4] > array[2]

8	2	7	6	5	4	3	1
1	2	3	4	5	6	7	8

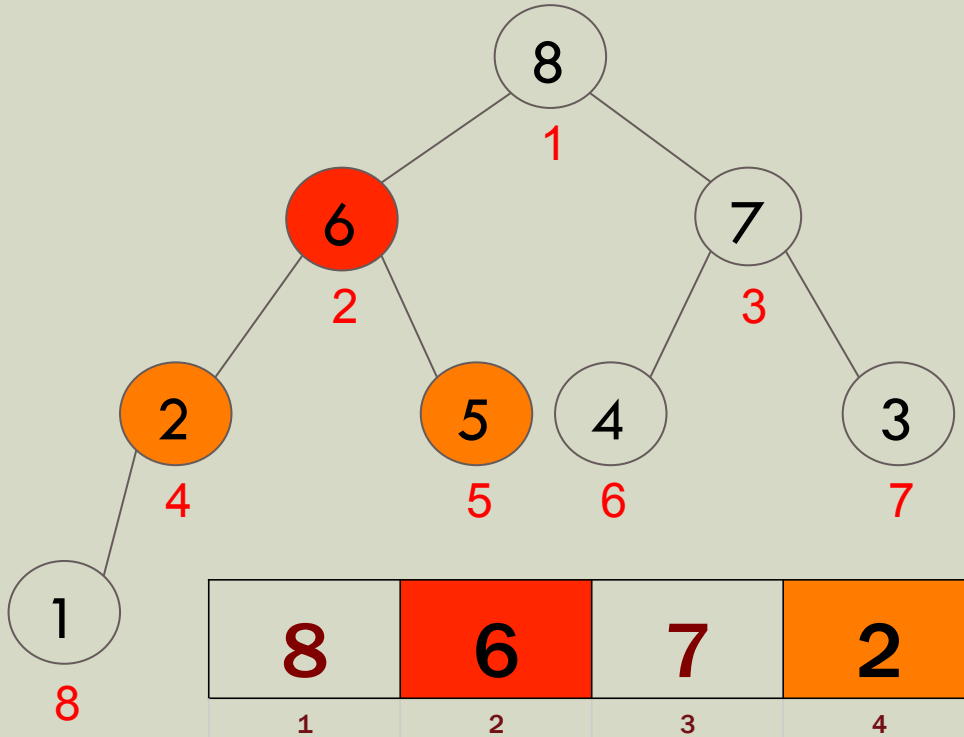
CONSTRUÇÃO DA *HEAP*



- restaura(2)
  - array[4] > array[2]
  - troca(4, 2)

8	2	7	6	5	4	3	1
1	2	3	4	5	6	7	8

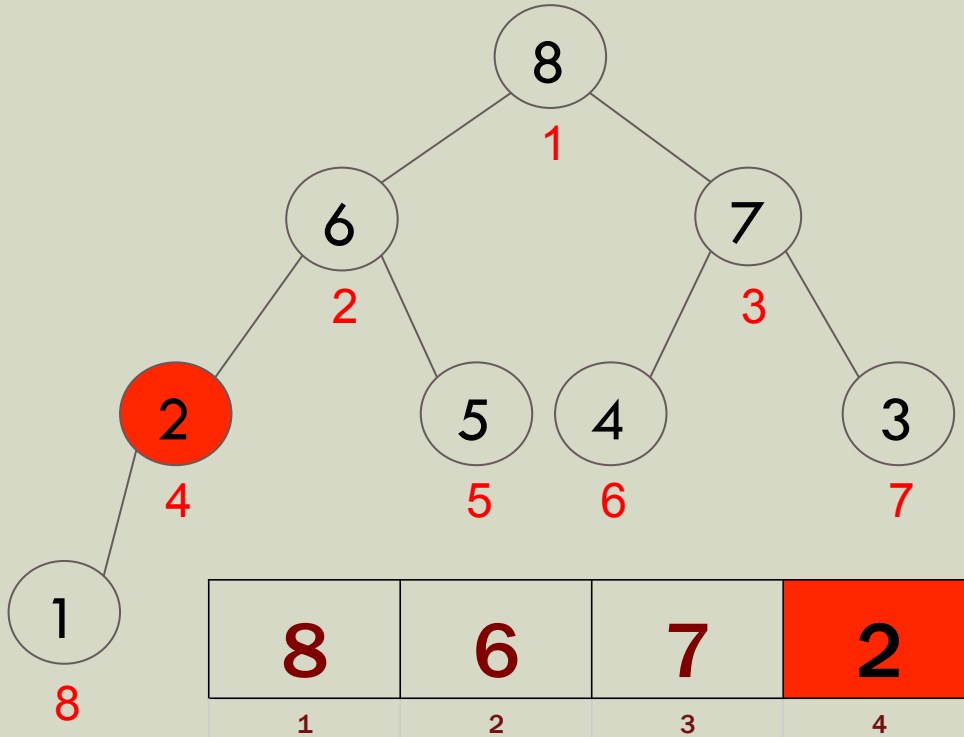
CONSTRUÇÃO DA *HEAP*



- restaura(2)
  - array[4] > array[2]
  - troca(4, 2)

8	6	7	2	5	4	3	1
1	2	3	4	5	6	7	8

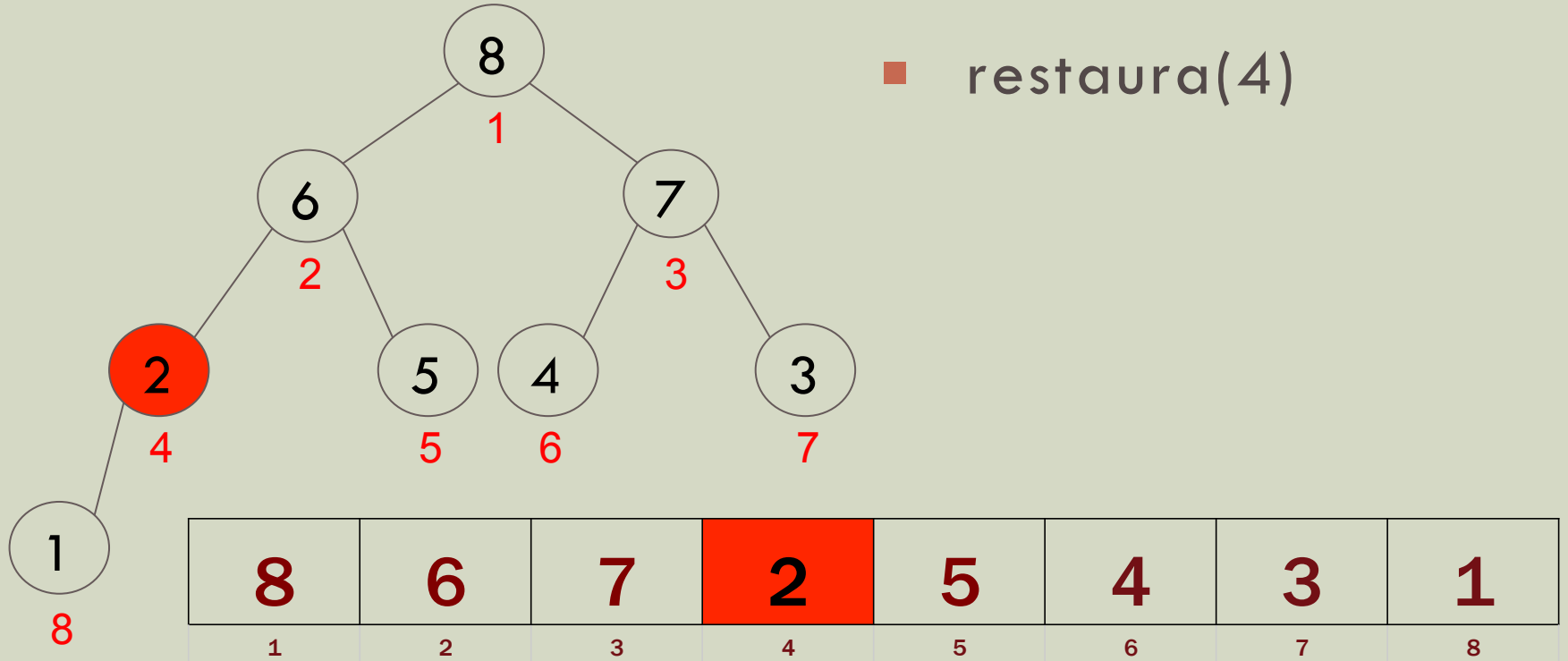
CONSTRUÇÃO DA *HEAP*



- restaura(2)
  - array[4] > array[2]
  - troca(4, 2)
  - restaura(4)

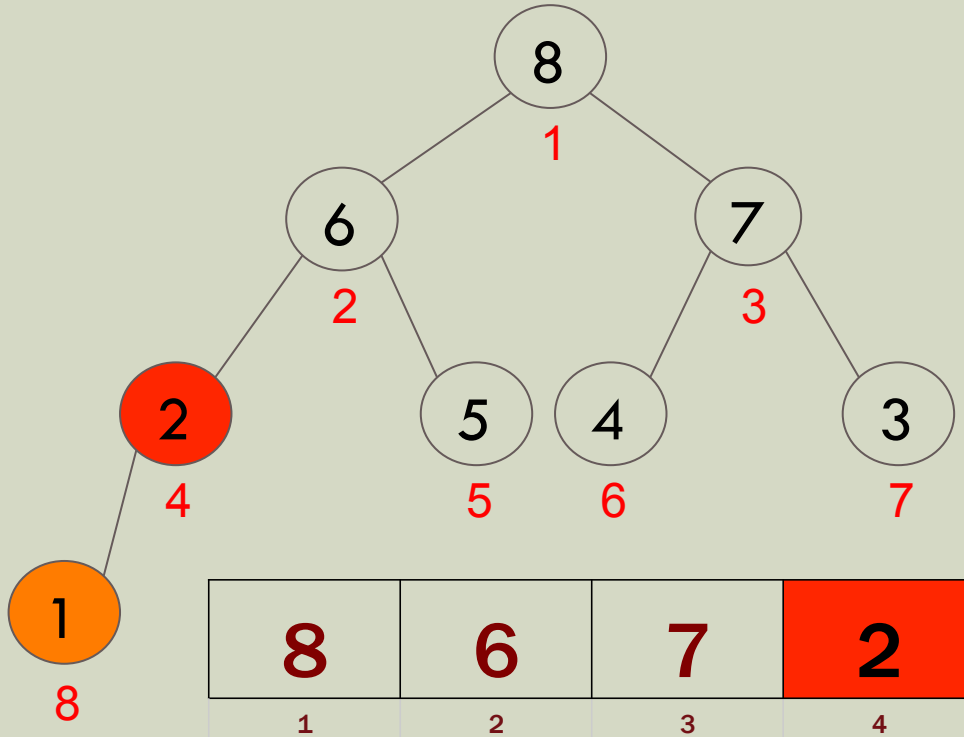
8	6	7	2	5	4	3	1
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*



- restaura(4)

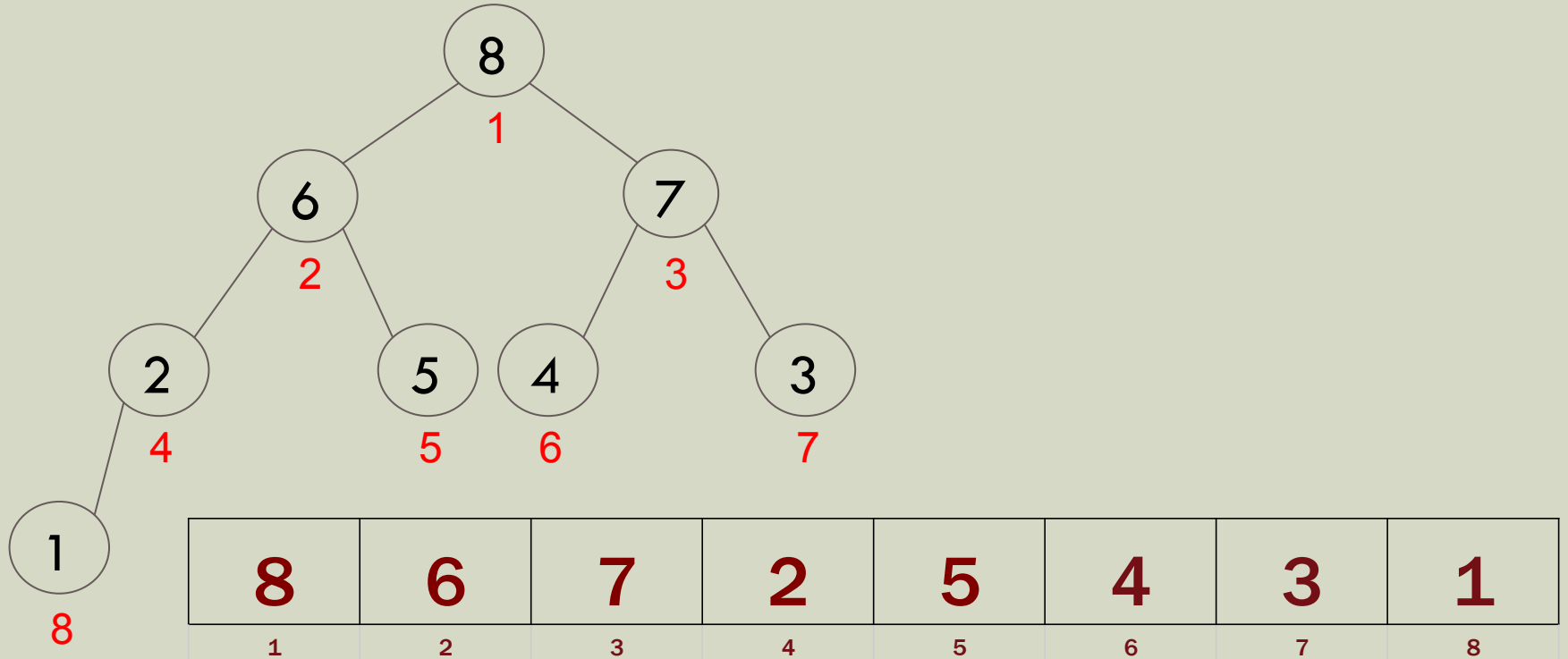
CONSTRUÇÃO DA *HEAP*



- restaura(4)
  - array[4] > array[8]

8	6	7	2	5	4	3	1
1	2	3	4	5	6	7	8

CONSTRUÇÃO DA *HEAP*



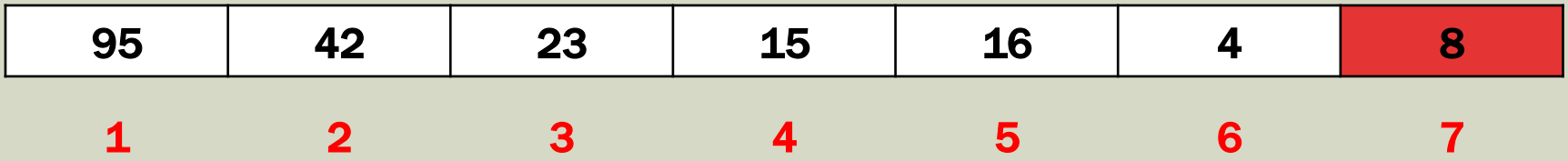
ALGORITMO *HEAPSORT*

- Construir a *heap*;
- Para cada posição de referência, a partir do fim do conjunto:
  - trocar o elemento que ocupa a posição de referência com o primeiro;
  - restaurar a *heap* a partir da primeira posição.

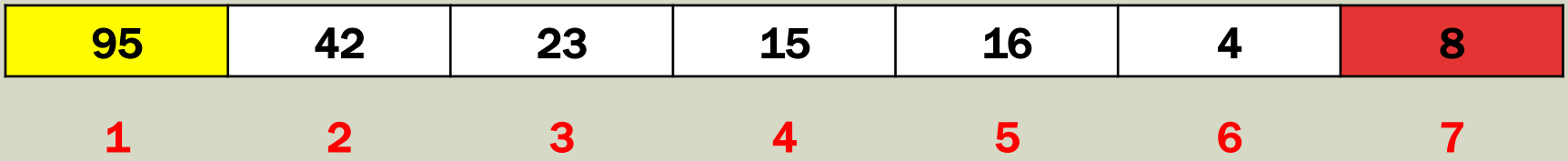
ORDENAÇÃO COM *HEAPSORT*

95	42	23	15	16	4	8
1	2	3	4	5	6	7

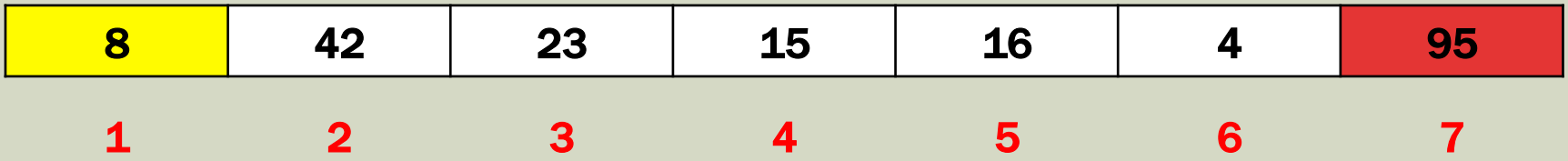
ORDENAÇÃO COM *HEAPSORT*



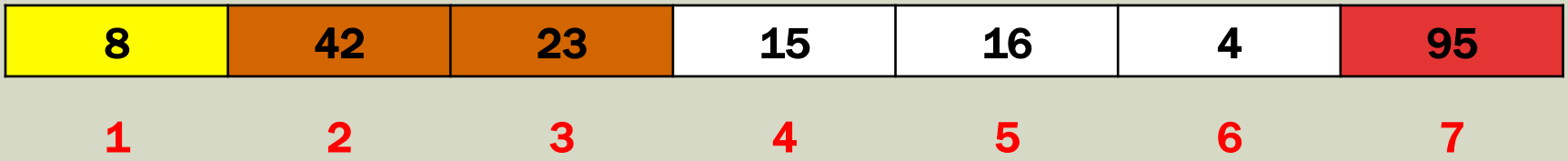
ORDENAÇÃO COM *HEAPSORT*



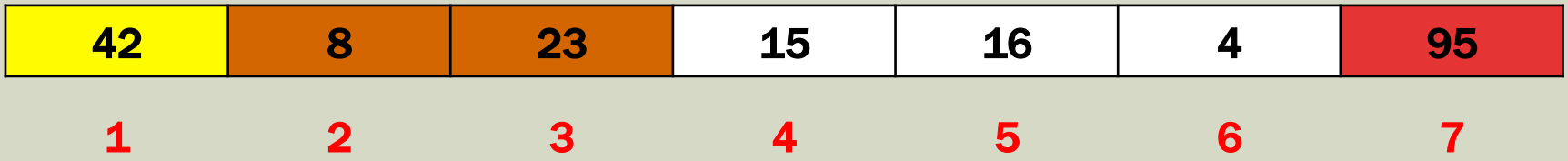
ORDENAÇÃO COM *HEAPSORT*



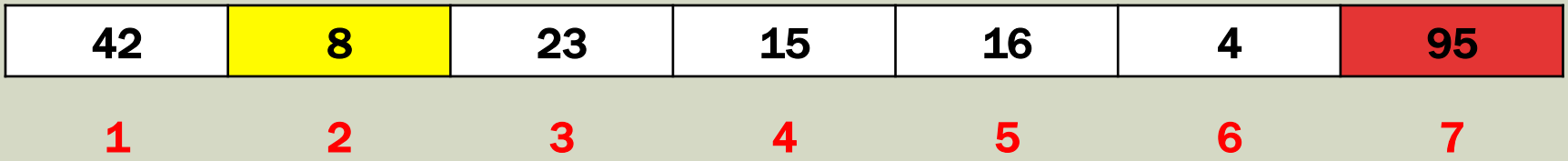
ORDENAÇÃO COM *HEAPSORT*



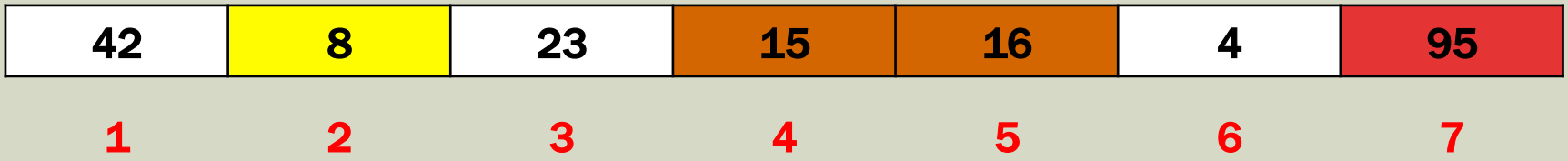
ORDENAÇÃO COM *HEAPSORT*



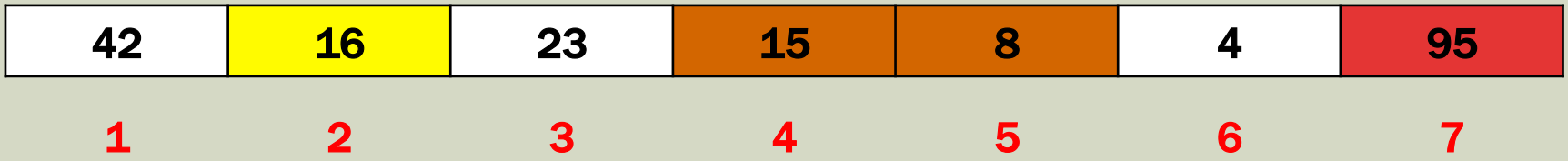
ORDENAÇÃO COM *HEAPSORT*



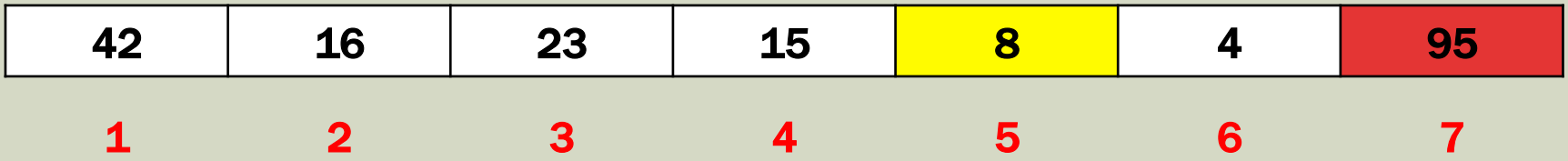
ORDENAÇÃO COM *HEAPSORT*



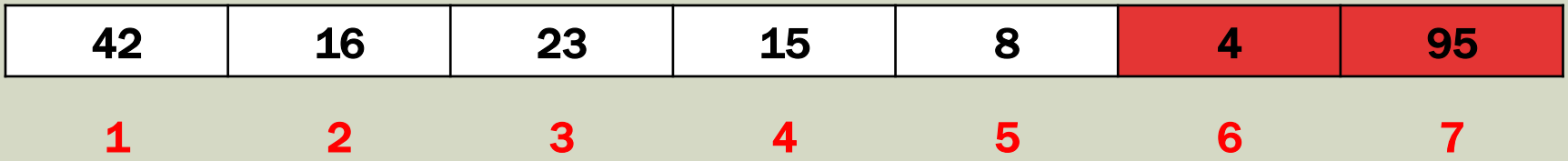
ORDENAÇÃO COM *HEAPSORT*



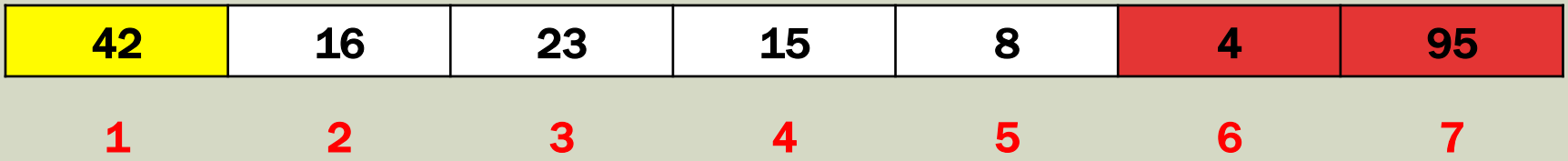
ORDENAÇÃO COM *HEAPSORT*



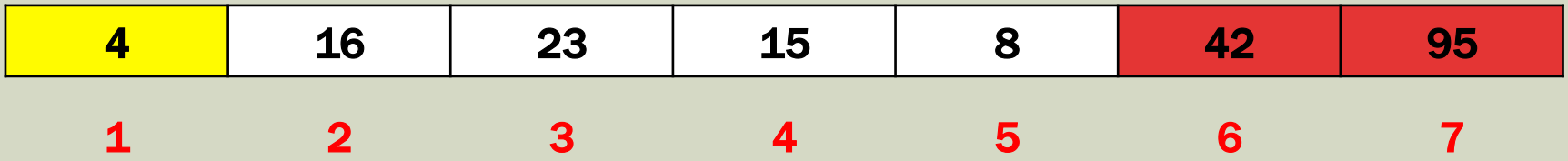
ORDENAÇÃO COM *HEAPSORT*



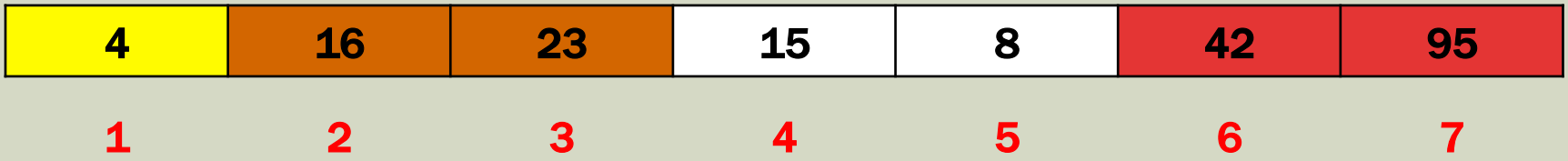
ORDENAÇÃO COM *HEAPSORT*



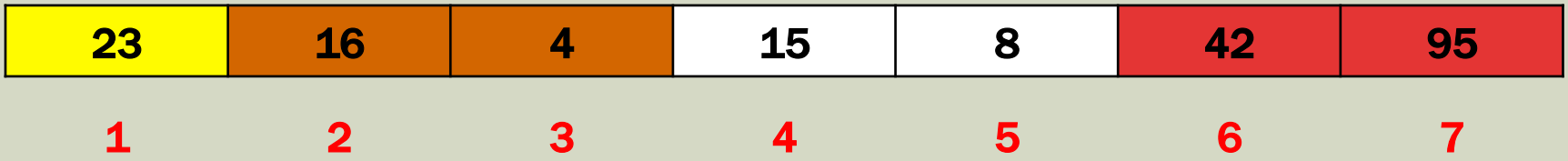
ORDENAÇÃO COM *HEAPSORT*



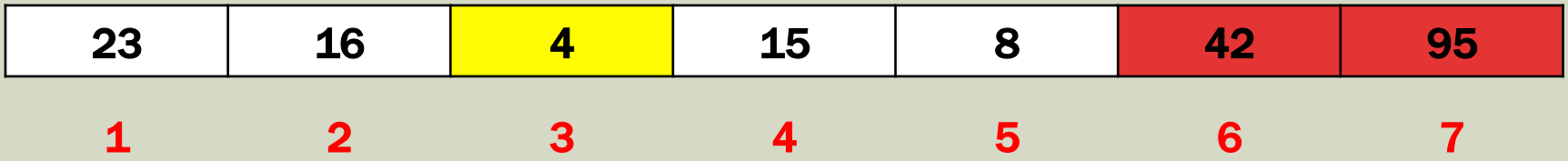
ORDENAÇÃO COM *HEAPSORT*



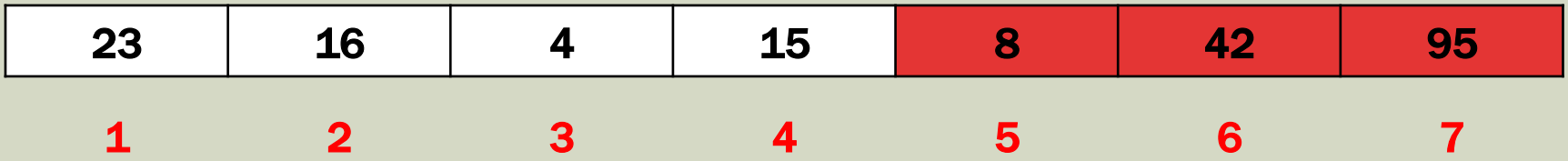
ORDENAÇÃO COM *HEAPSORT*



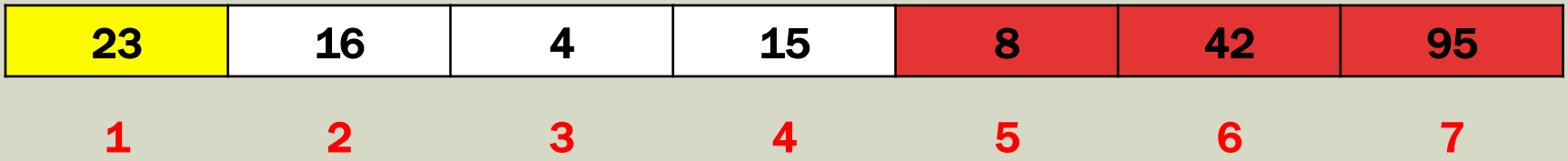
ORDENAÇÃO COM *HEAPSORT*



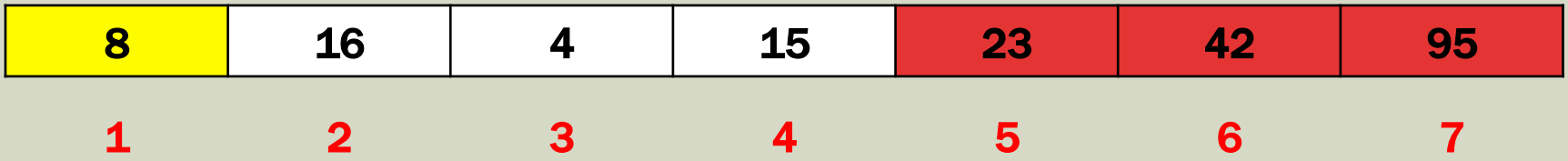
ORDENAÇÃO COM *HEAPSORT*



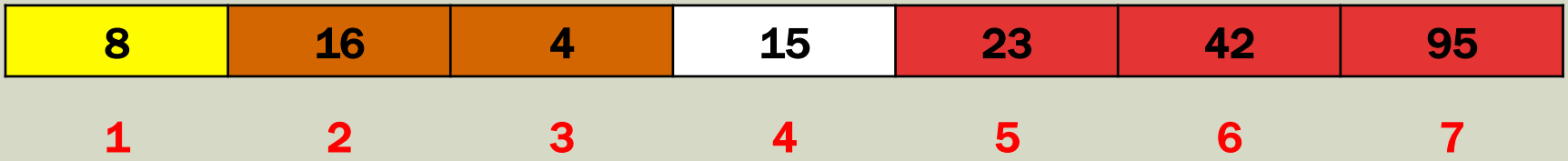
ORDENAÇÃO COM *HEAPSORT*



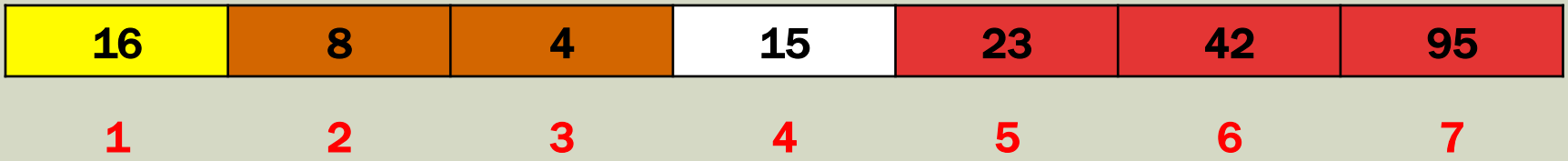
ORDENAÇÃO COM *HEAPSORT*



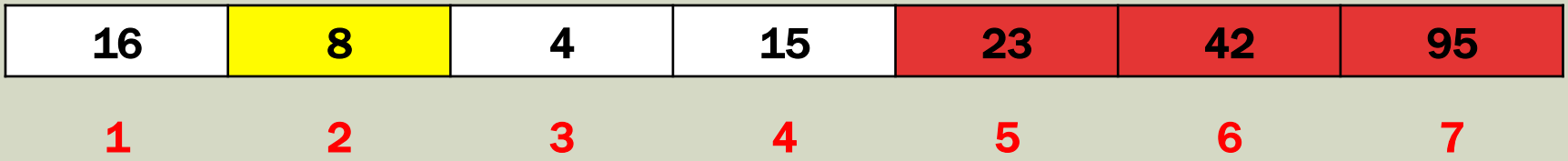
ORDENAÇÃO COM *HEAPSORT*



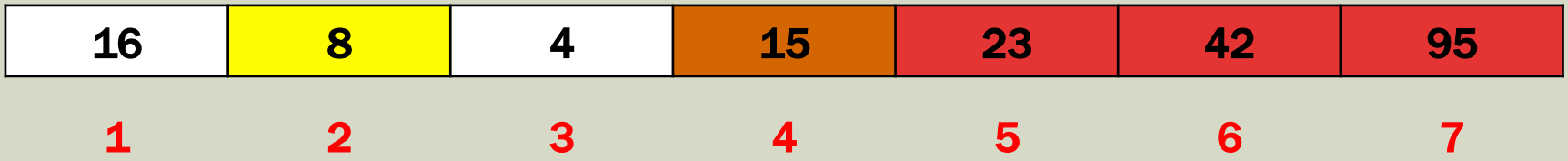
ORDENAÇÃO COM *HEAPSORT*



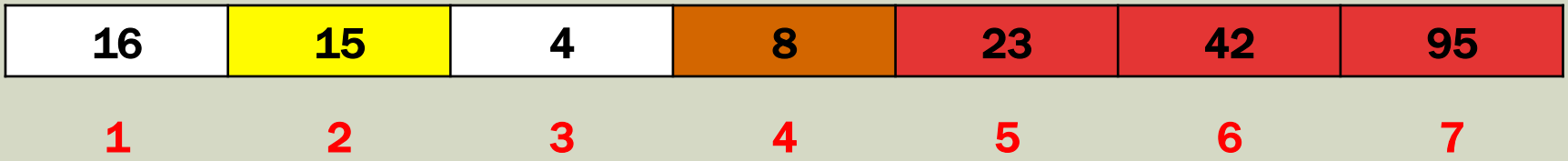
ORDENAÇÃO COM *HEAPSORT*



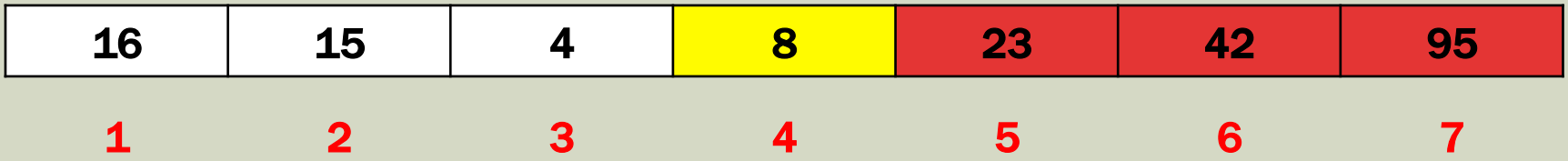
ORDENAÇÃO COM *HEAPSORT*



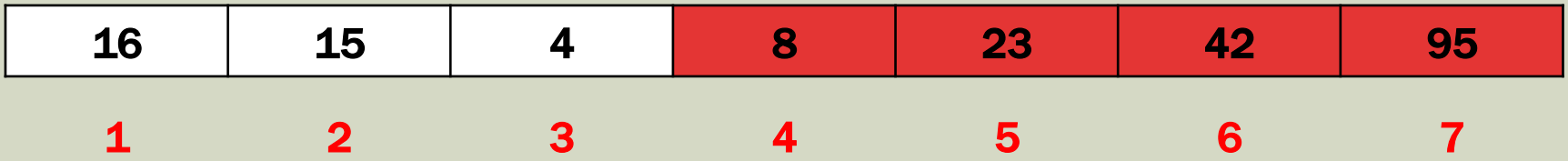
ORDENAÇÃO COM *HEAPSORT*



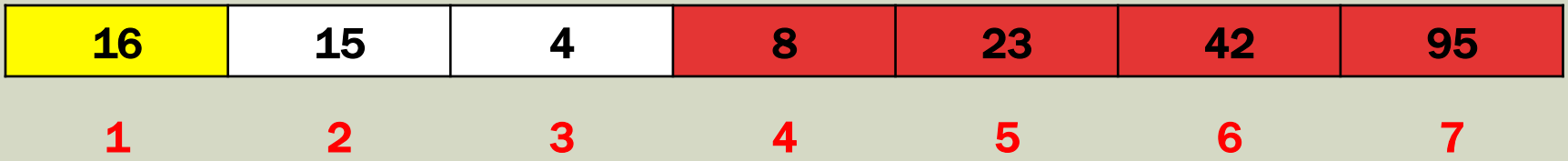
ORDENAÇÃO COM *HEAPSORT*



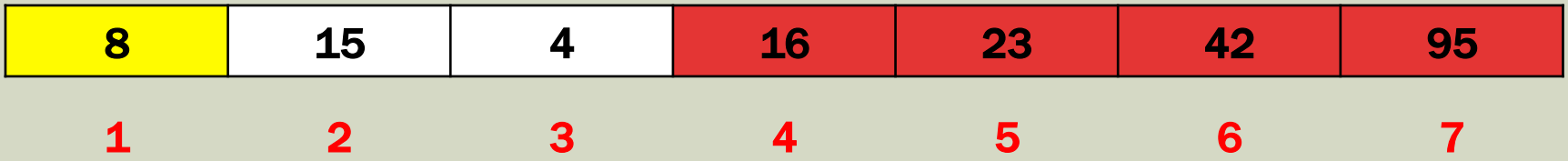
ORDENAÇÃO COM *HEAPSORT*



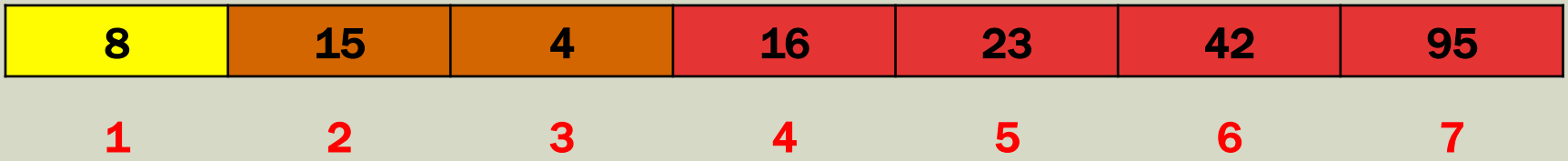
ORDENAÇÃO COM *HEAPSORT*



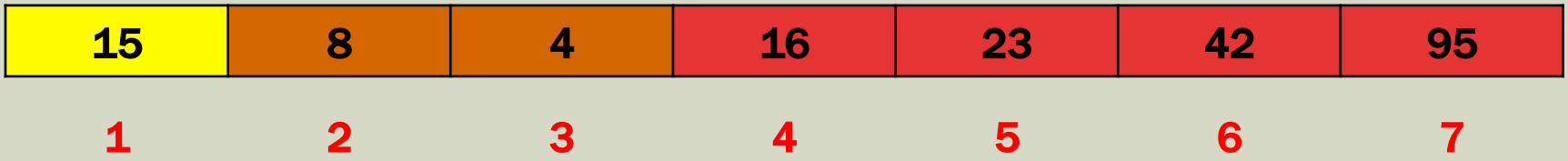
ORDENAÇÃO COM *HEAPSORT*



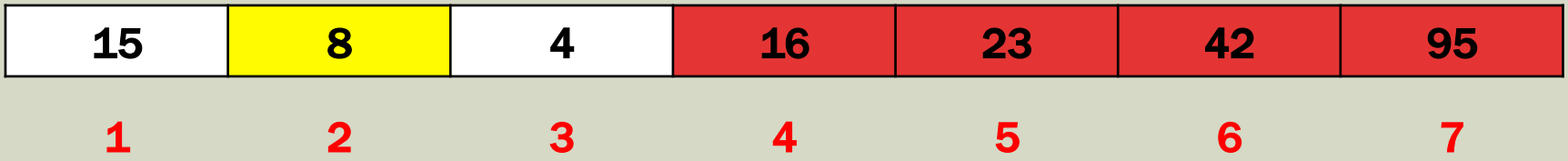
ORDENAÇÃO COM *HEAPSORT*



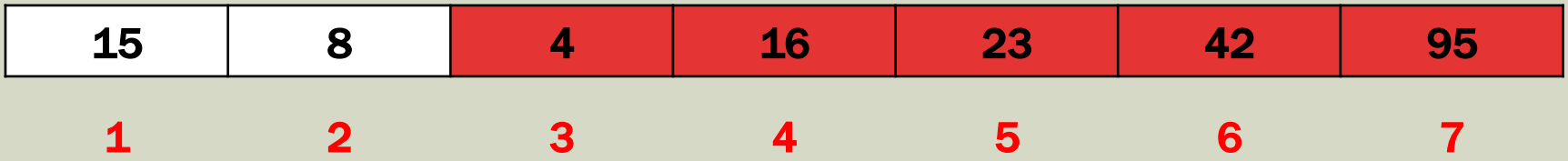
ORDENAÇÃO COM *HEAPSORT*



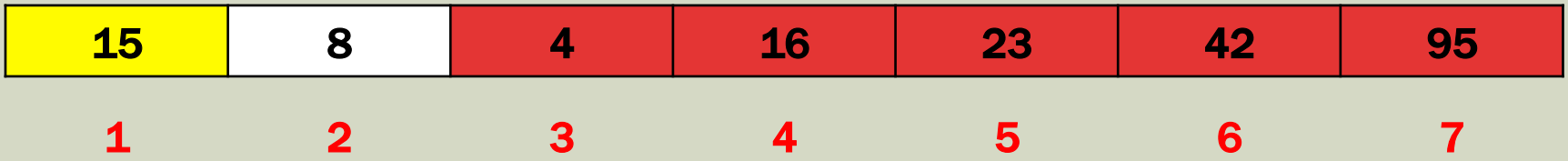
ORDENAÇÃO COM *HEAPSORT*



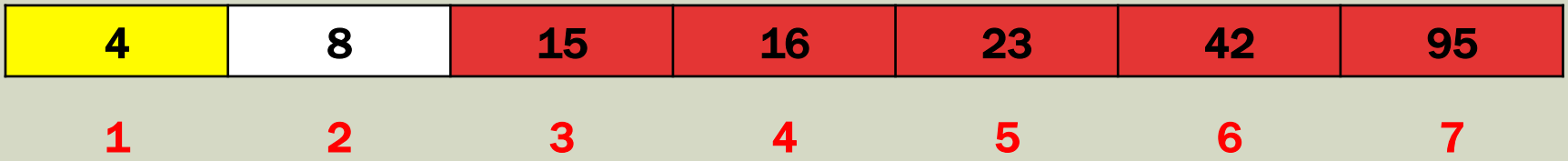
ORDENAÇÃO COM *HEAPSORT*



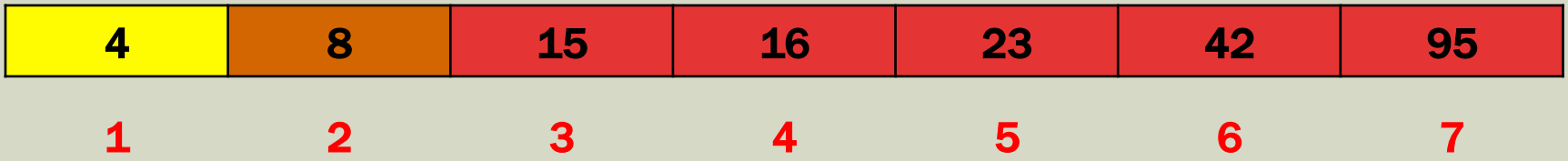
ORDENAÇÃO COM *HEAPSORT*



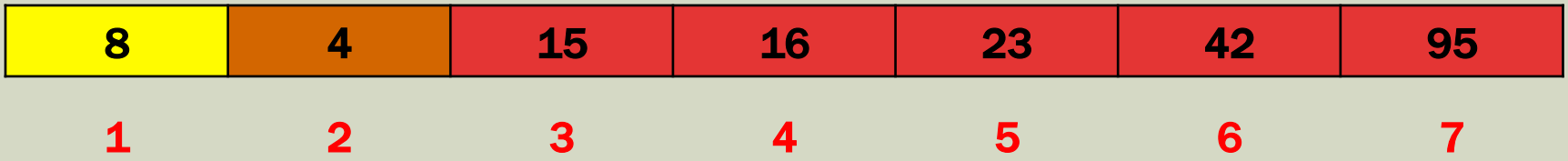
ORDENAÇÃO COM *HEAPSORT*



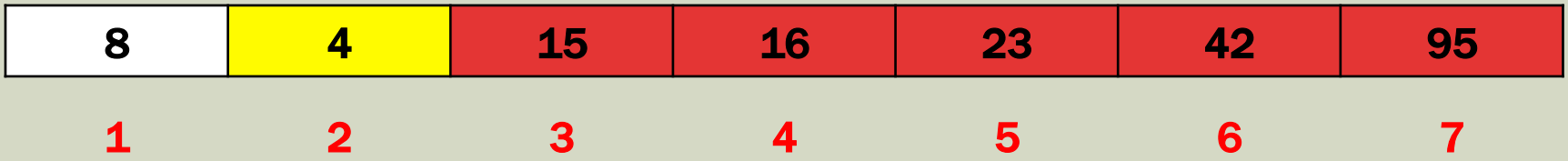
ORDENAÇÃO COM *HEAPSORT*



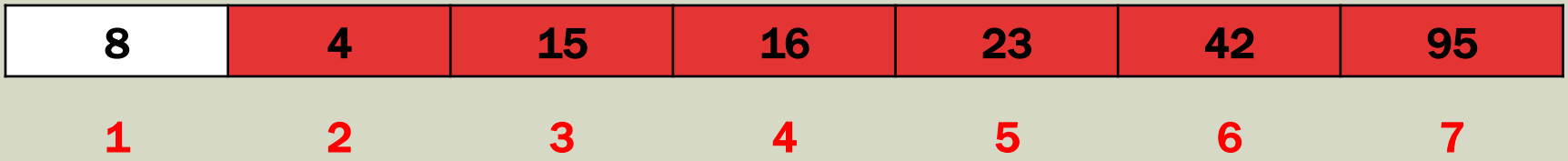
ORDENAÇÃO COM *HEAPSORT*



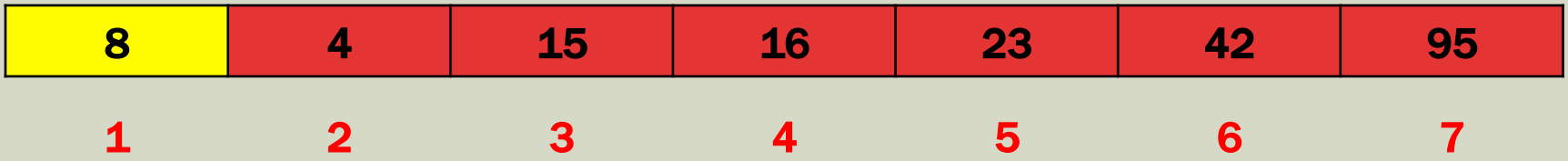
ORDENAÇÃO COM *HEAPSORT*



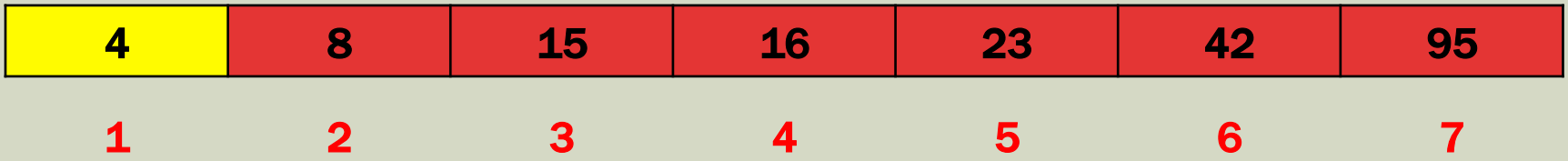
ORDENAÇÃO COM *HEAPSORT*



ORDENAÇÃO COM *HEAPSORT*



ORDENAÇÃO COM *HEAPSORT*



ORDENAÇÃO COM *HEAPSORT*

4	8	15	16	23	42	95
1	2	3	4	5	6	7

CONSIDERAÇÕES

- Método não-estável.
- Custo alto para construir o *heap*.