Aula 3

Heaps, Heap Sort, Filas de Prioridade

Heaps

Definição [Heap]

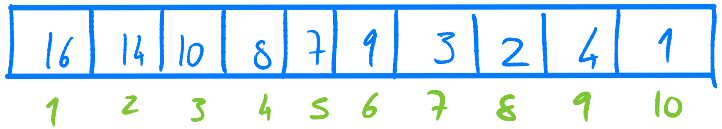
Um array $A[1..n]$ diz-se um heap se:

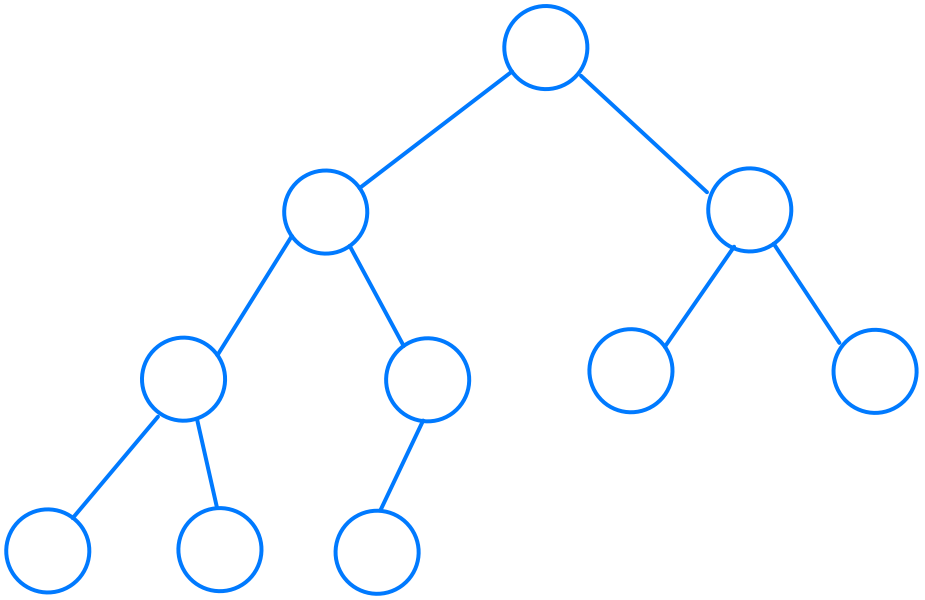
$$\forall\, 1 < i \leq n.\ A[Parent(i)] \geq A[i]$$

onde:

- $Parent(i) = \lfloor i/2 \rfloor$
- $Left(i) = 2 * i$
- $Right(i) = 2 * i + 1$

Exemplo:





Heaps

Definição [Heap]

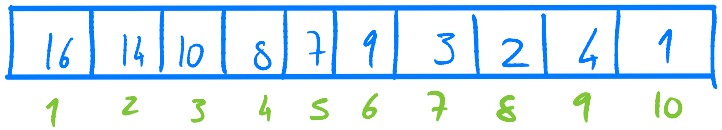
Um array $A[1..n]$ diz-se um heap se:

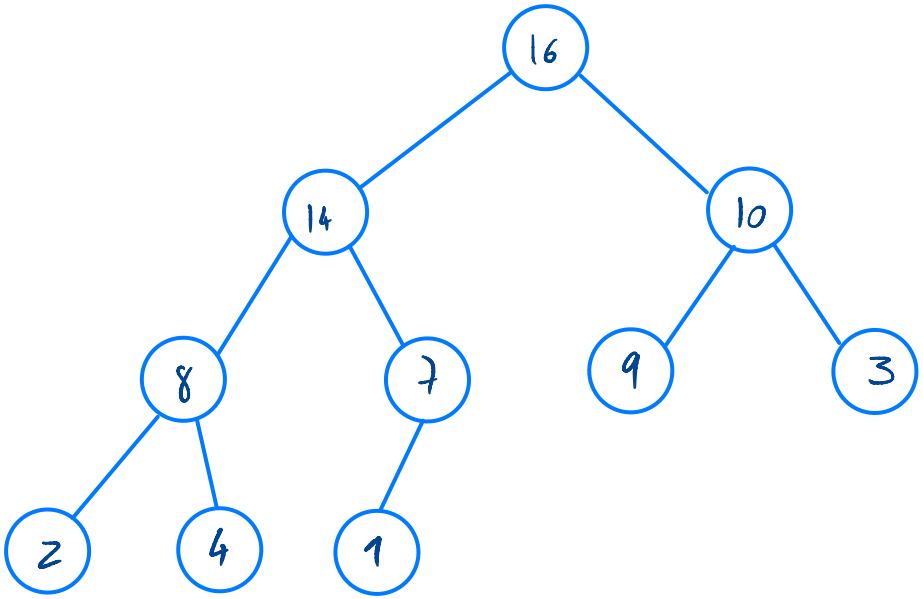
$$\forall 1 < i \leq n.\ A[Parent(i)] \geq A[i]$$

onde:

- $Parent(i) = \lfloor i/2 \rfloor$
- $Left(i) = 2 \times i$
- $Right(i) = 2 \times i + 1$

Exemplo:

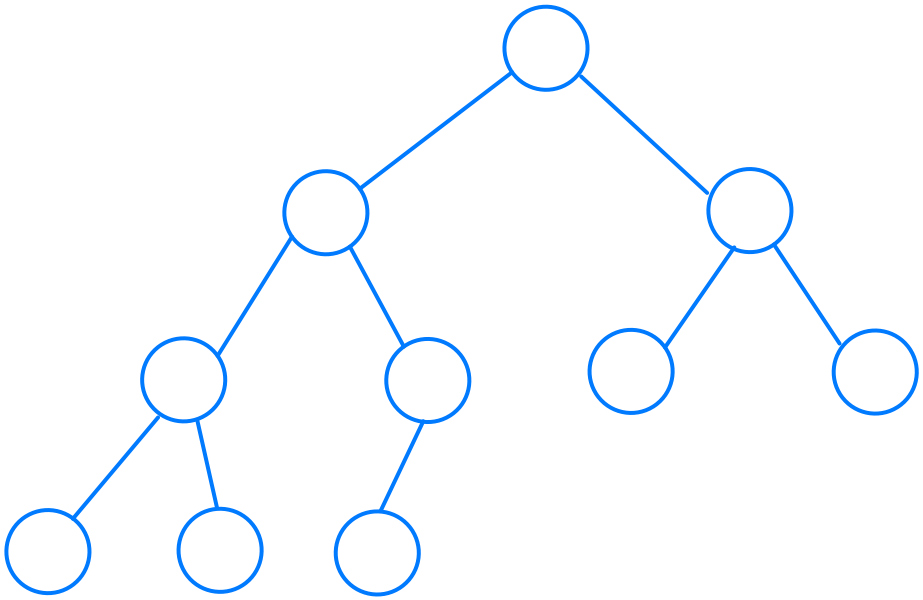




Max Heapify

Exemplo:

16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

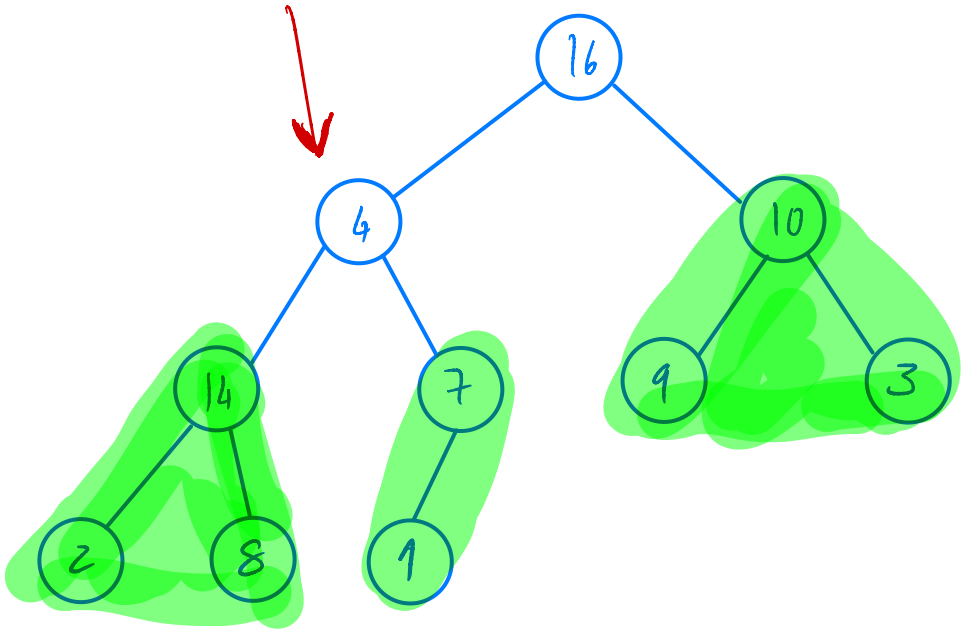


- Condição de heap falha para:

Max Heapify

Exemplo:

16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10



- Condição de heap falha para:

  - $i = 5$

    $A[Parent(i)] = A[2] = 4$

    $\neq A[i] = 7$

  - $i = 4$

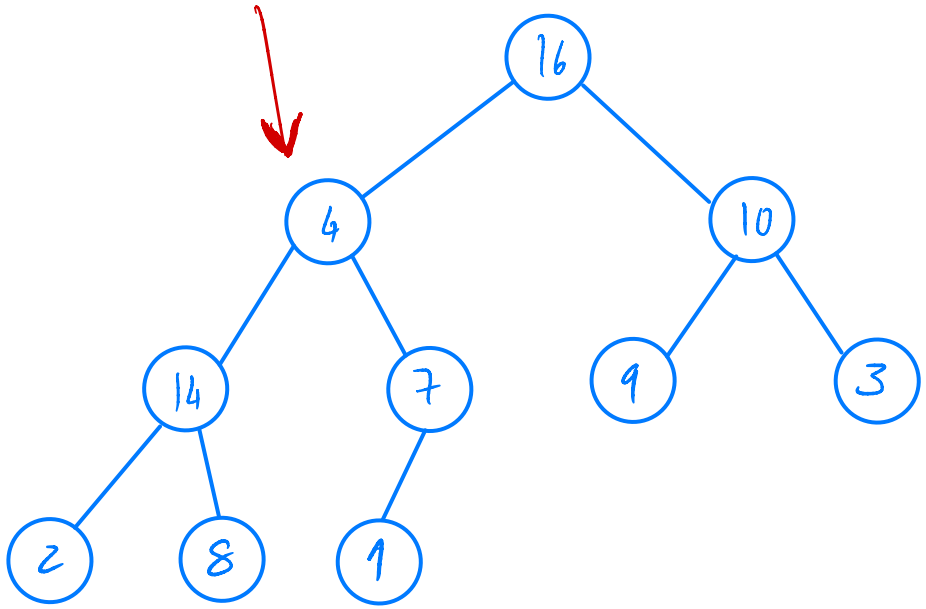
    $A[Parent(i)] = A[2] = 4$

    $\neq A[i] = 14$

Max Heapify

Exemplo:

16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10



- Condição de heap falha para:

  - $i = 5$

    $A[Parent(i)] = A[2] = 4$

    $\neq A[i] = 7$

  - $i = 4$

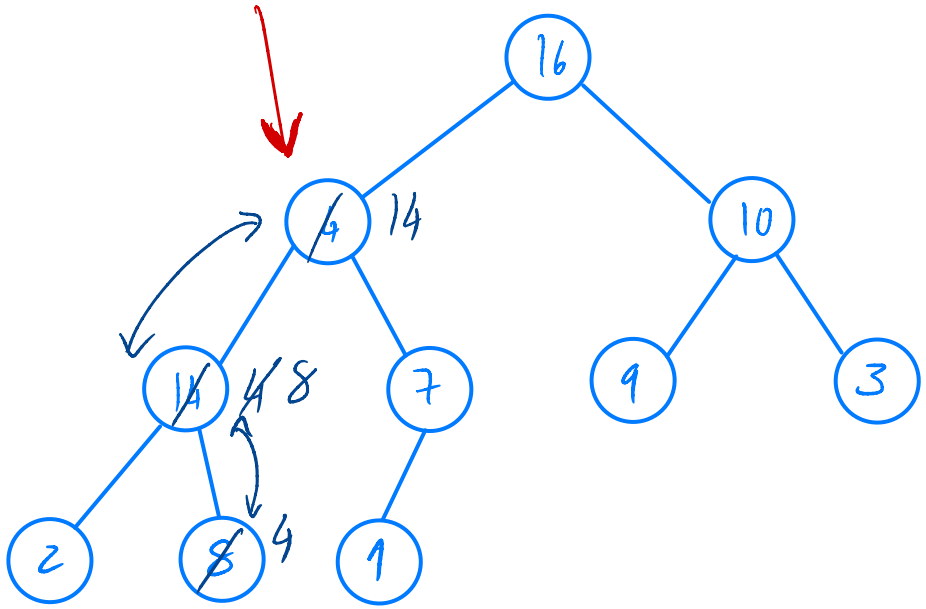
    $A[Parent(i)] = A[2] = 4$

    $\neq A[i] = 14$

Max Heapify

Exemplo:

16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10



- Condição de heap falha para:

  - $i = 5$

    $A[Parent(i)] = A[2] = 4$

    $\neq A[i] = 7$

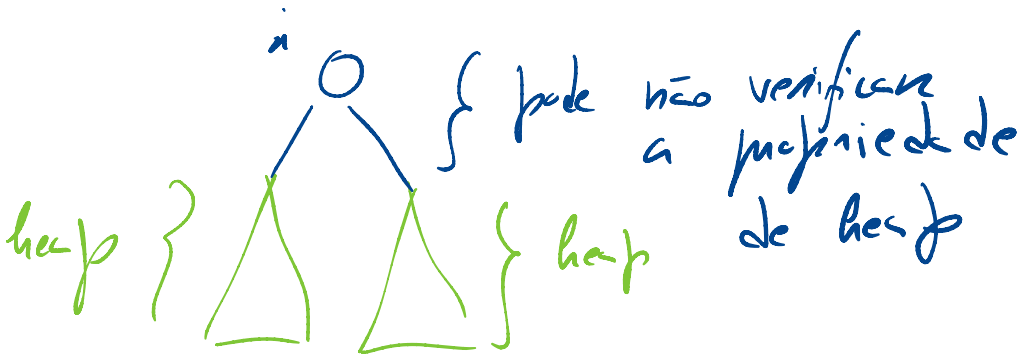
  - $i = 4$

    $A[Parent(i)] = A[2] = 4$

    $\neq A[i] = 14$

Max Heapify

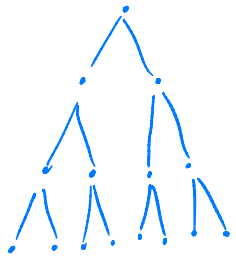
```
Max Heapify (A, i)

    l := Left (i)
    r := Right (i)

    max := i
    if ( l ≤ A.lenght && A[max] < A[l] )
        max := l
    if ( r ≤ A.lenght && A[max] < A[r] )
        max := r
    if (max ≠ i)
        Swap (A, i, max)
        Max Heapify (A, max)
```



Propriedades de Heaps

- Altura máxima de um heap com n elementos:

$$\lfloor \lg n \rfloor$$



- Heap completo de altura h: $2^{h+1} - 1$

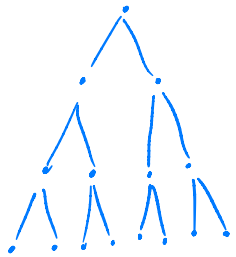
- Nº de elementos de um heap de altura h: x

$$< x \leq$$

Propriedades de Heaps

- Altura máxima de um heap com n elementos:

$$\lfloor \lg n \rfloor$$



- Heap completo de altura h: $2^{h+1} - 1$

- Nº de elementos de um heap de altura h: x

$$2^h - 1 < x \leq 2^{h+1} - 1$$

$$2^h \leq x < 2^{h+1}$$

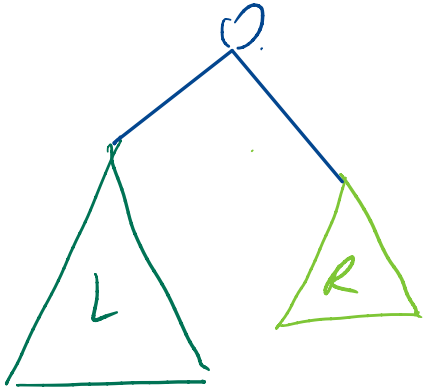
$$h \leq \lg x < h+1$$

$$\lfloor \lg x \rfloor = h$$

Max Heapify

```
Max Heapify (A, i)

    l := Left (i)
    r := Right (i)

    max := i
    if ( l ≤ A.lenght && A[max] < A[l] )
        max := l
    if ( r ≤ A.lenght && A[max] < A[r] )
        max := r
    if (max ≠ i)
        swap (A, i, max)
        Max Heapify (A, max)
```

- Complexidade



- Qual é o maior nº de vezes que podemos invocar recursivamente a função Max Heapify?

Max Heapify

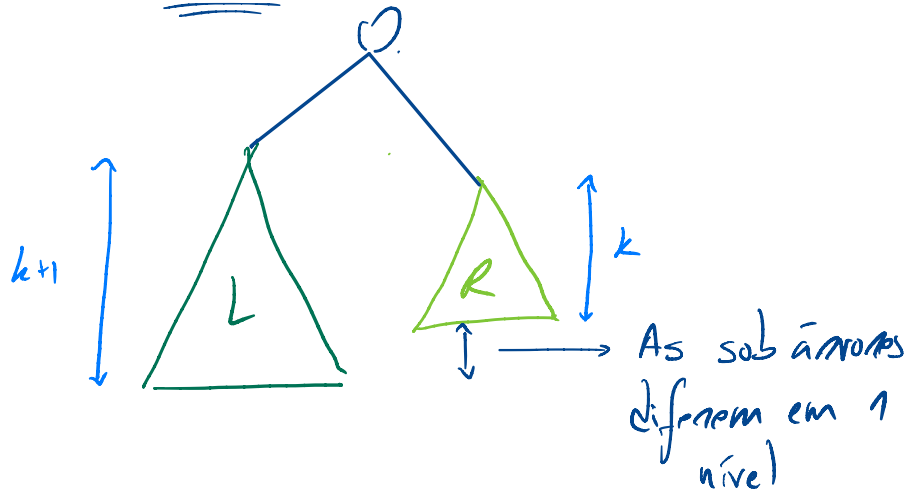
```
Max Heapify (A, i)

    l := Left (i)
    r := Right (i)

    max := i
    if ( l ≤ A.lenght && A[max] < A[l] )
        max := l
    if ( r ≤ A.lenght && A[max] < A[r] )
        max := r
    if (max ≠ i)
        swap (A, i, max)
        Max Heapify (A, max)
```

- Complexidade

$$T(n) = T(\alpha \cdot n) + O(1)$$

↓

Quem é α?



Nº de elementos de L:

Nº de elementos de R:

Nº total de elementos:

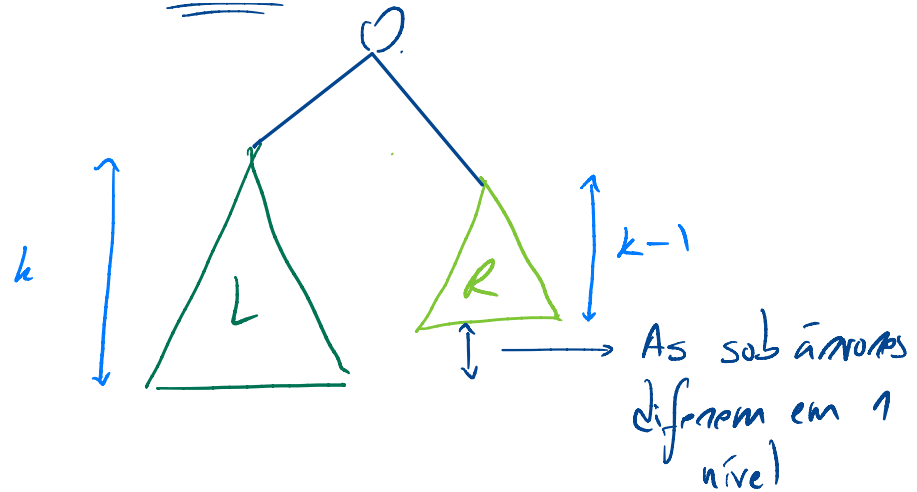
Max Heapify

```
Max Heapify (A, i)

    l := Left (i)
    r := Right (i)

    max := i
    if ( l ≤ A.lenght && A[max] < A[l] )
        max := l
    if ( r ≤ A.lenght && A[max] < A[r] )
        max := r
    if (max ≠ i)
        swap (A, i, max)
        Max Heapify (A, max)
```

- Complexidade

$$T(n) = T(\alpha \cdot n) + O(1)$$

Quem é α?



Nº de elementos de L: $2^{k+1} - 1$

Nº de elementos de R: $2^k - 1$

Nº total de elementos: $(2^{k+1} - 1) + 2^k - 1 + 1$

$= 2^{k+1} + 2^k - 2 + 1$

$= 2^k (2+1) - 1$

$= 3 \cdot 2^k - 1$

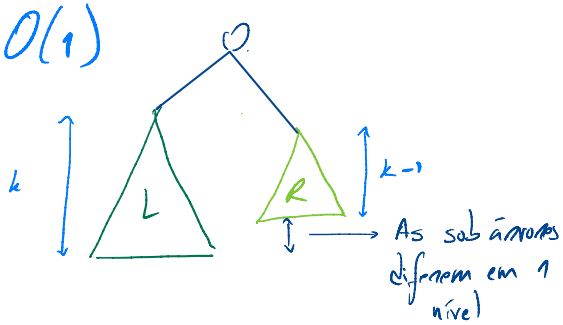
Max Heapify

```
MaxHeapify(A, i)

    l := Left(i)
    r := Right(i)

    max := i
    if( l ≤ A.lenght && A[max] < A[l] )
        max := l
    if ( r ≤ A.lenght && A[max] < A[r] )
        max := r
    if (max ≠ i)
        Swap(A, i, max)
        MaxHeapify(A, max)
```

- Complexidade

Quem é α?

$$T(n) = T(\alpha \cdot n) + O(1)$$



Nº de elementos de L: $2^{k+1} - 1$

Nº de elementos de R: $2^k - 1$

Nº total de elementos: $3 \cdot 2^k - 1$

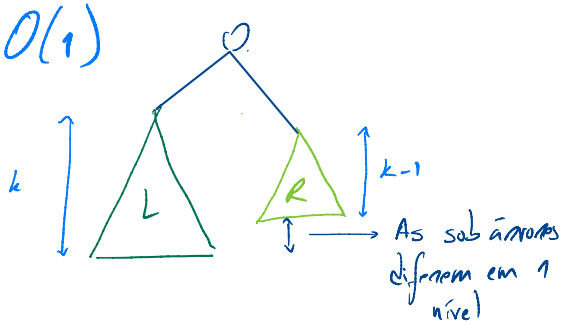
$$\alpha = \frac{\text{Nº de elementos de } L}{\text{Nº total de elementos}}$$

Max Heapify

```
Max Heapify (A, i)

    l := Left (i)
    r := Right (i)

    max := i
    if ( l ≤ A.lenght && A[max] < A[l] )
        max := l
    if ( r ≤ A.lenght && A[max] < A[r] )
        max := r
    if (max ≠ i)
        swap (A, i, max)
        Max Heapify (A, max)
```

- Complexidade → Quem é a?

$$T(n) = T(\alpha \cdot n) + O(1)$$



Nº de elementos de L: $2^{k+1} - 1$

Nº de elementos de R: $2^k - 1$

Nº total de elementos: $3 \cdot 2^k - 1$

$$\alpha = \frac{\text{Nº de elementos de } L}{\text{Nº total de elementos}}$$

$$\alpha(k) = \frac{2^{k+1} - 1}{3 \cdot 2^k - 1}$$

$$\lim_{k \to \infty} \frac{2^{k+1} - 1}{3 \cdot 2^k - 1} = \frac{2}{3} \cdot \lim_{n \to \infty} \frac{2^k - 1/2}{2^k - 1/3}$$

$$= 2/3$$

$$T(n) = T(2/3\, n) + O(1)$$

$a = 1$

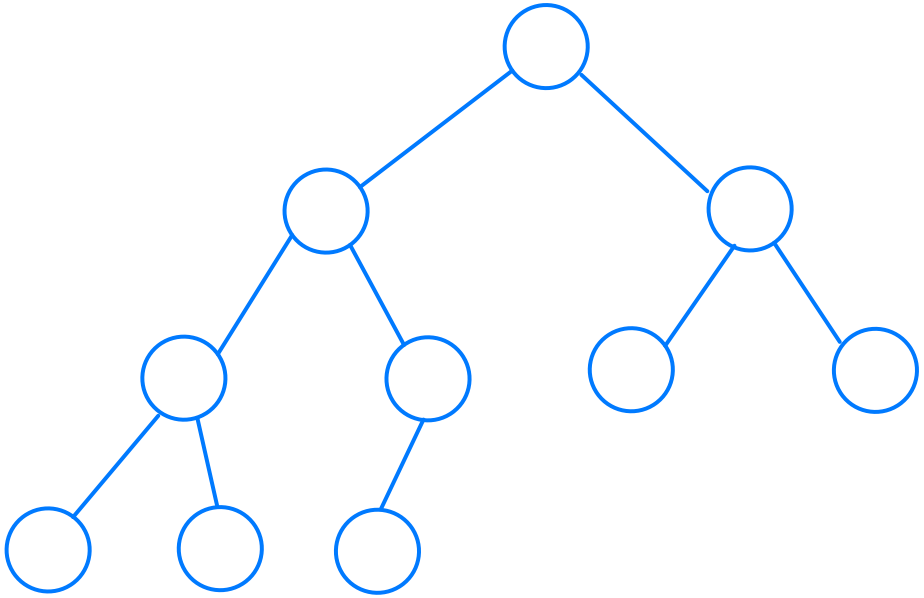
$b = 3/2$ $\quad \log_b a = 0 = d$ $\quad T(n) = O(\log n)$

$d = 0$

Construção de Heaps

Exemplo:

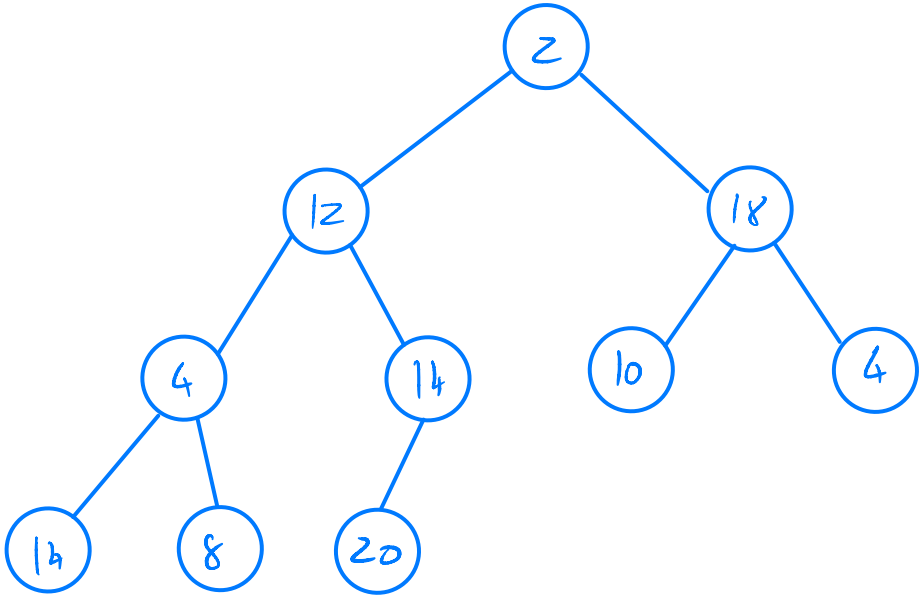
2	12	18	4	14	10	4	14	8	20
1	2	3	4	5	6	7	8	9	10



Construção de Heaps

Exemplo:

2	12	18	4	14	10	4	14	8	20
1	2	3	4	5	6	7	8	9	10

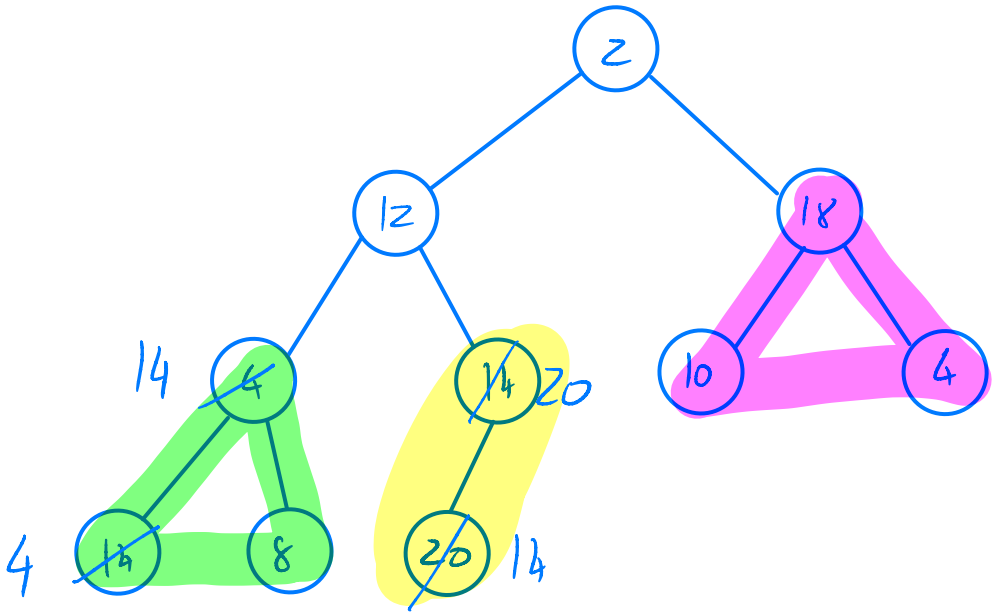


- Construímos um heap aplicando a operação de Max Heapify de baixo para cima da direita para a esquerda.

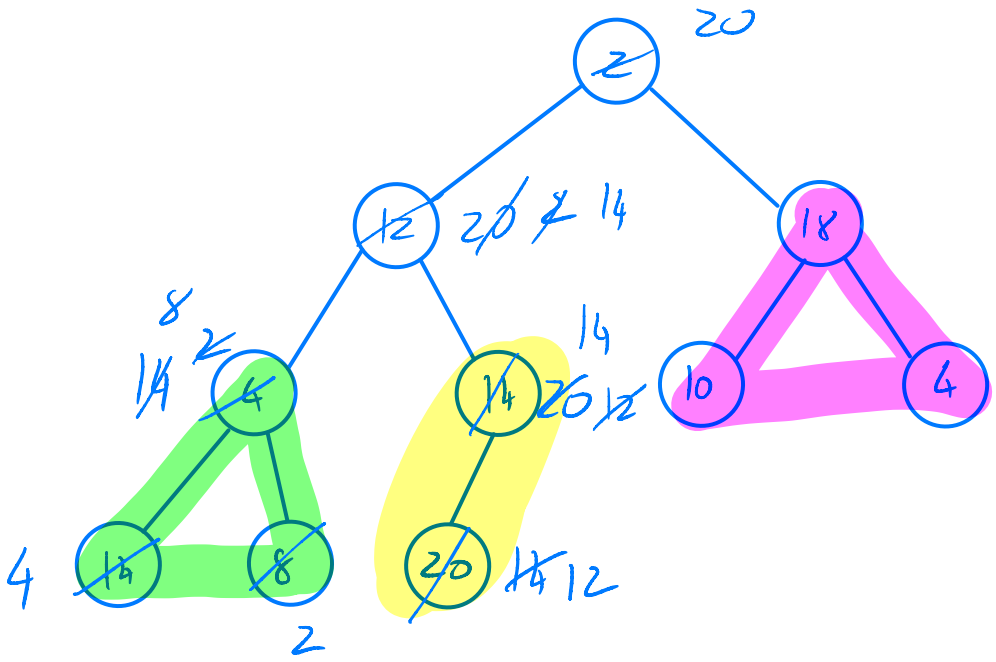
Construção de Heaps

Exemplo:

2	12	18	4	14	10	4	14	8	20
1	2	3	4	5	6	7	8	9	10



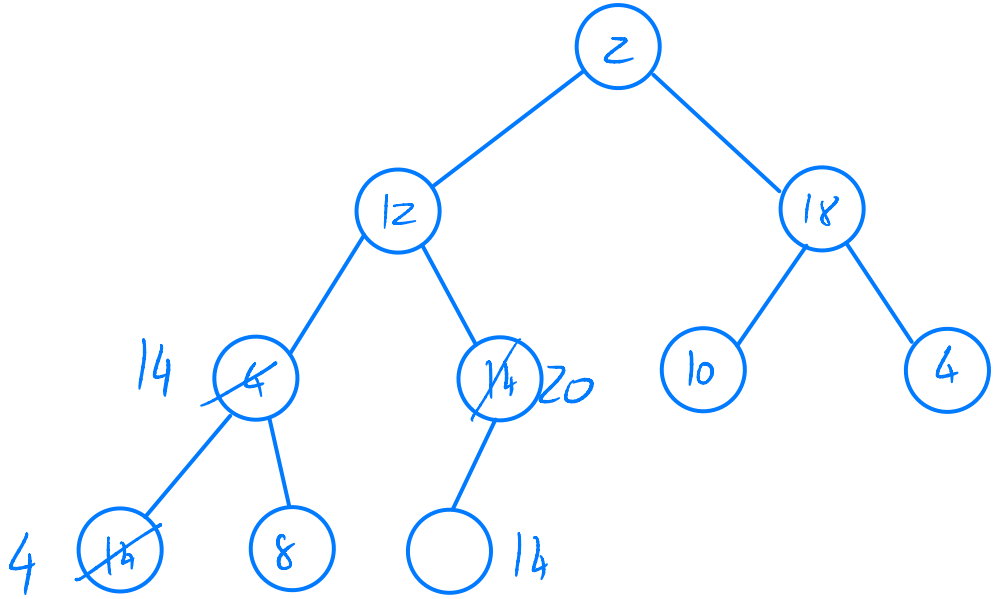
20	14	18	8	14	10	4	4	2	12
1	2	3	4	5	6	7	8	9	10



Construção de Heaps

Exemplo:

2	12	18	4	14	10	4	14	8	20
1	2	3	4	5	6	7	8	9	10

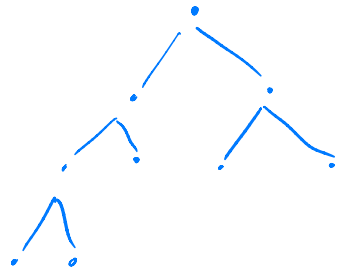


- Construímos um heap aplicando a operação de Max Heapify de baixo para cima da direita para a esquerda.

Propriedades de Heaps

$n = 5$

$i = 2$



$n = 9$

$i = 4$

- Índice do primeiro nó com filhos num heap com n elementos

- O índice do 1º nó com filhos é $\lfloor n/2 \rfloor$

Prova

- 2 casos: $i \leq \lfloor n/2 \rfloor \Rightarrow$ i tem filhos
  $i > \lfloor n/2 \rfloor \Rightarrow$ i não tem filhos

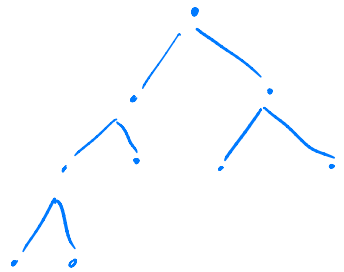
(A) $i \leq \lfloor n/2 \rfloor \Rightarrow$ i tem filhos

$$Left(i) = 2 * i = 2 * \lfloor n/2 \rfloor \leq \frac{2 * n}{2}$$

Propriedades de Heaps

$n = 5$

$i = 2$



$n = 9$

$i = 4$

- Indice do primeiro nó com filhos num heap com n elementos

- O indice do 1º nó com filhos é $\lfloor n/2 \rfloor$

Prova

- 2 casos: $i \leq \lfloor n/2 \rfloor \Rightarrow$ i tem filhos
  $i > \lfloor n/2 \rfloor \Rightarrow$ i não tem filhos

(b) $i > \lfloor n/2 \rfloor \Rightarrow$ i não tem filhos

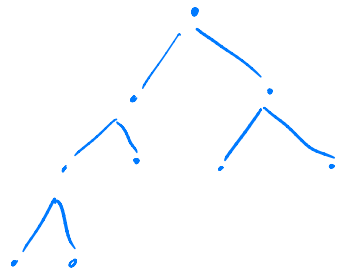
(b.1) n é par: $\exists m.\ n = 2 * m$

(b.2) i é impar: $\exists m.\ i = 2 * m + 1$

Propriedades de Heaps

$n = 5$

$i = 2$



$n = 9$

$i = 4$

- Índice do primeiro nó com filhos num heap com n elementos

- O índice do 1º nó com filhos é $\lfloor n/2 \rfloor$

Prova

- 2 casos: $i \leq \lfloor n/2 \rfloor \Rightarrow i$ tem filhos
  $i > \lfloor n/2 \rfloor \Rightarrow i$ não tem filhos

(b) $i > \lfloor n/2 \rfloor \Rightarrow i$ não tem filhos

(b.1) n é par: $\exists m.\ n = 2 \times m$

$i > \lfloor n/2 \rfloor \Leftrightarrow i > \lfloor \frac{2m}{2} \rfloor$

$i > m$

$Left(i) = 2 \times i > 2 \times m = n$

(b.2) i é ímpar: $\exists m.\ i = 2 \times m + 1$

$i > \lfloor n/2 \rfloor \Leftrightarrow i > \lfloor \frac{2m+1}{2} \rfloor$

$\Rightarrow i > m$

$Left(i) = 2 \times i \Rightarrow Left(i) > 2 \times m$

$> 2 \times m + 1$

porque $Left(i)$ é par

Construção de Heaps - Build Max Heap

```
Build Max Heap (A)
    k := ⌊A.length/2⌋
    for i := k to 1
        Max Heapify (A, i)
```

Complexidade [1ª Aproximação]

Complexidade [Limite apertado]

Construção de Heaps - Build Max Heap

```
Build Max Heap (A)
    k := ⌊A.length/2⌋
    for i := k to 1
        Max Heapify (A, i)
```

Complexidade [1ª Aproximação]

$O(n \lg n)$

Complexidade [Limite apertado]

$$\sum_{h=0}^{\lfloor \lg n \rfloor} trees(n,h) \times O(h)$$

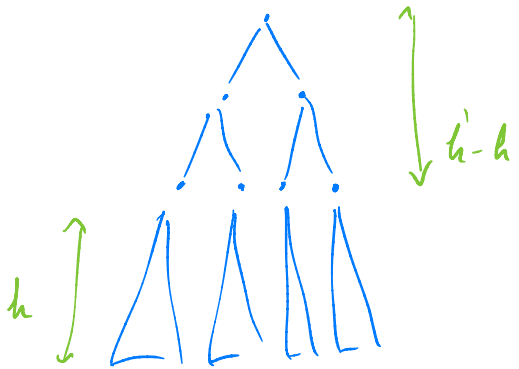
$trees(n,h)$ → nº de árvores de altura h

$O(h)$ → complexidade do Max Heapify numa árvore de altura h

Propriedades de Heaps

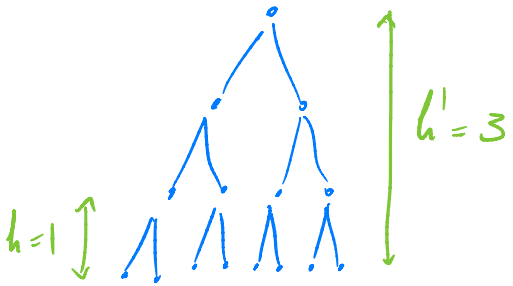
- Nº de árvores de altura h é no máximo $\left\lceil \frac{n}{2^{h+1}} \right\rceil$

Prova:



nº total de elementos:

Nº de árvores de altura h?

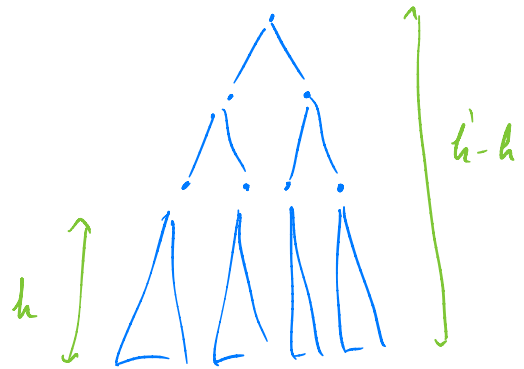


Nº de árvores de altura h:

Propriedades de Heaps

- Nº de árvores de altura h é no máximo $\left\lceil \frac{n}{2^{h+1}} \right\rceil$

Prova:



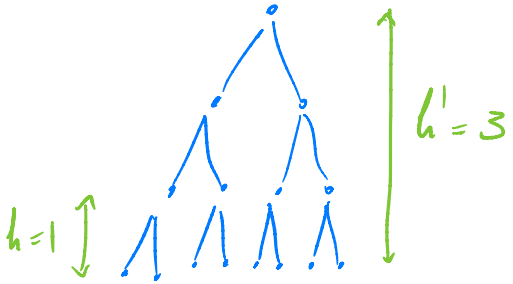
nº total de elementos: $2^{h'+1} - 1$

Nº de árvores de altura h? $2^{h'-h}$

$$2^{h'-h} = \frac{2^{h'}}{2^h} = \frac{2^{h'+1}}{2^{h+1}} = \frac{2^{h'+1} - 1}{2^{h+1}}$$

$$= \frac{n-1}{2^{h+1}}$$

$$\leq \left\lceil \frac{n}{2^{h+1}} \right\rceil$$



Nº de árvores de altura h: 4

Construção de Heaps - Build Max Heap

```
Build Max Heap (A)
    k := ⌊A.length/2⌋
    for i := k to 1
        Max Heapify (A, i)
```

Complexidade [limite apertado]

$$\sum_{h=0}^{\lfloor \lg n \rfloor} trees(n,h) \times O(h)$$

$$\leq \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h)$$

Construção de Heaps - Build Max Heap

```
Build Max Heap (A)
    k := ⌊A.length/2⌋
    for i := k to 1
        Max Heapify (A, i)
```

Complexidade [limite apertado]

$$\sum_{h=0}^{\lfloor \lg n \rfloor} trees(n,h) \times O(h)$$

$$\leq \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h)$$

$$\leq O\left(\sum_{h=0}^{\lfloor \lg n \rfloor} \frac{n}{2^h} \cdot h \right)$$

$$= O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

$$\leq O\left(n \cdot \frac{1/2}{(1-1/2)^2} \right)$$

$$= O(2n)$$

$$= O(n)$$

$$\sum_{i=0}^{n} a^i = \frac{1}{1-a}$$

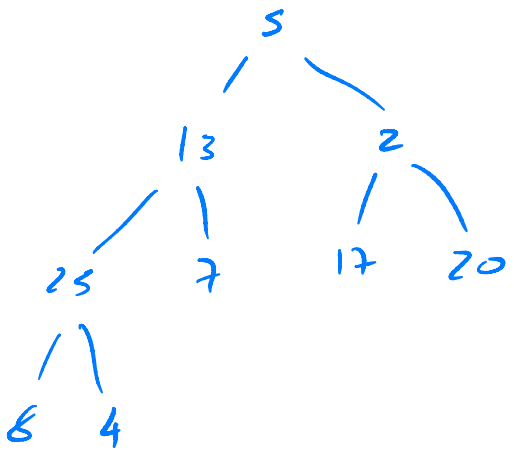
$$\sum_{i=0}^{n} i\, a^{i-1} = \frac{+1}{(1-a)^2}$$

$$\sum_{i=0}^{\infty} i\, a^i = \frac{a}{(1-a)^2}$$

Heap Sort - Exemplo

$\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

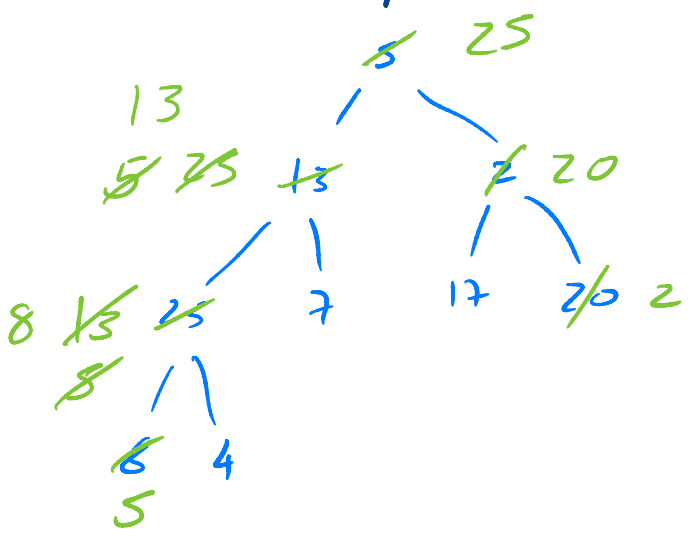
① Build Max Heap



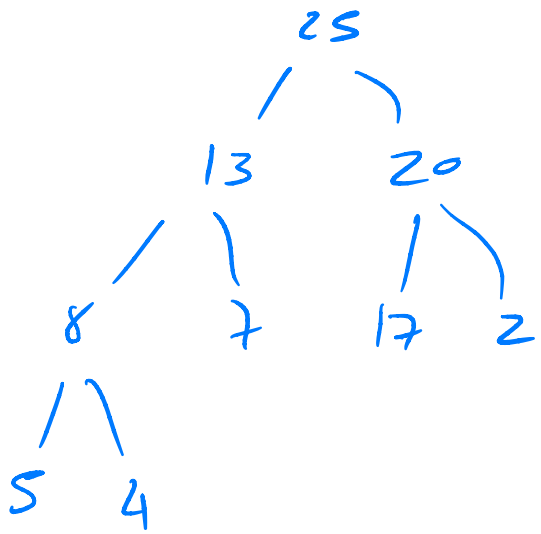
Heap Sort - Exemplo

$\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

① Build Max Heap



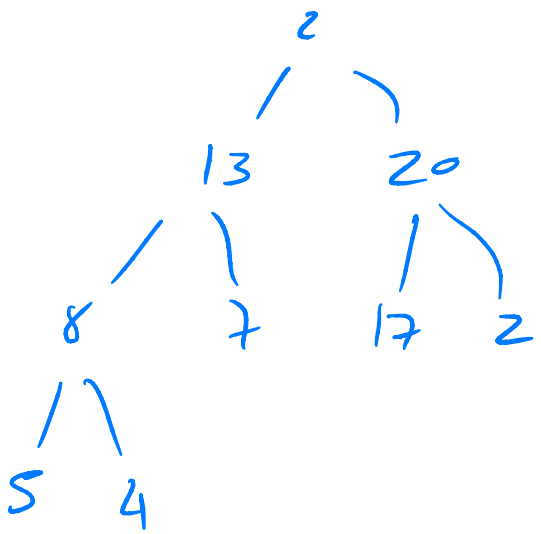
$\langle 25, 13, 20, 8, 7, 17, 2, 5, 4 \rangle$



Heap Sort - Exemplo

$\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

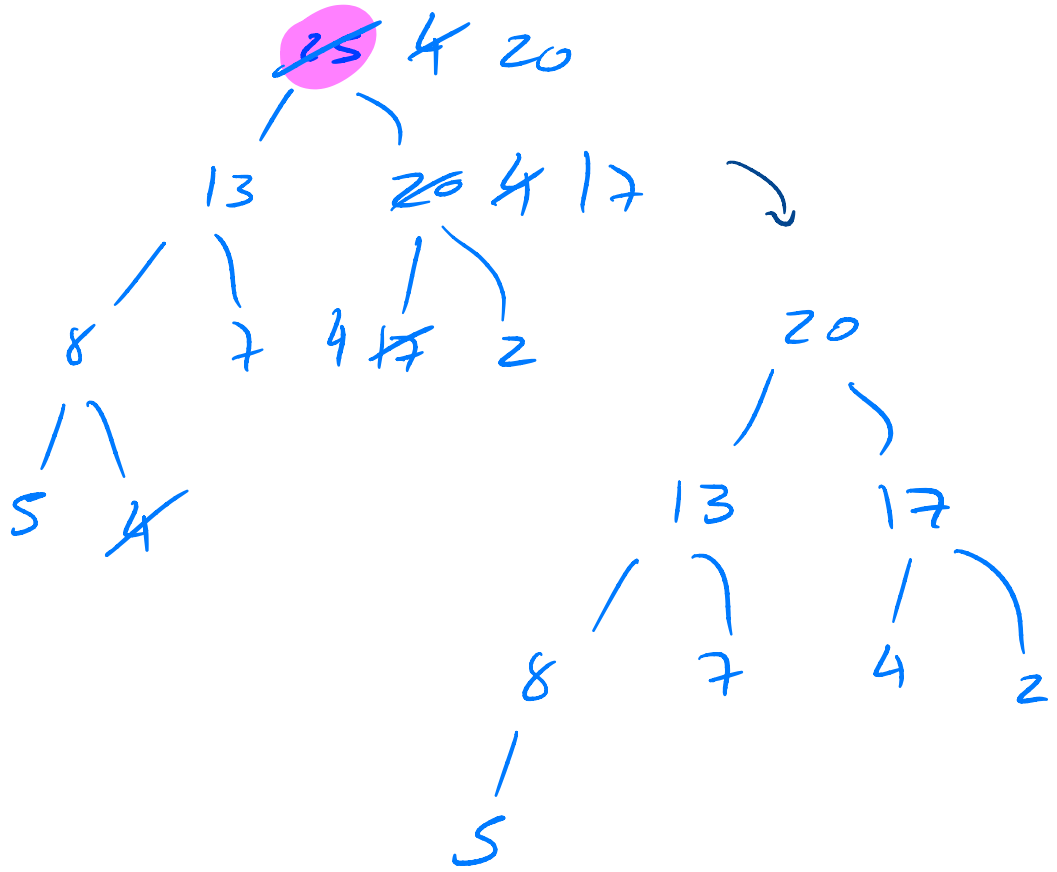
② Sequência de Max Heapify's



Heap Sort - Exemplo

$\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

② Sequência de Max Heapify's

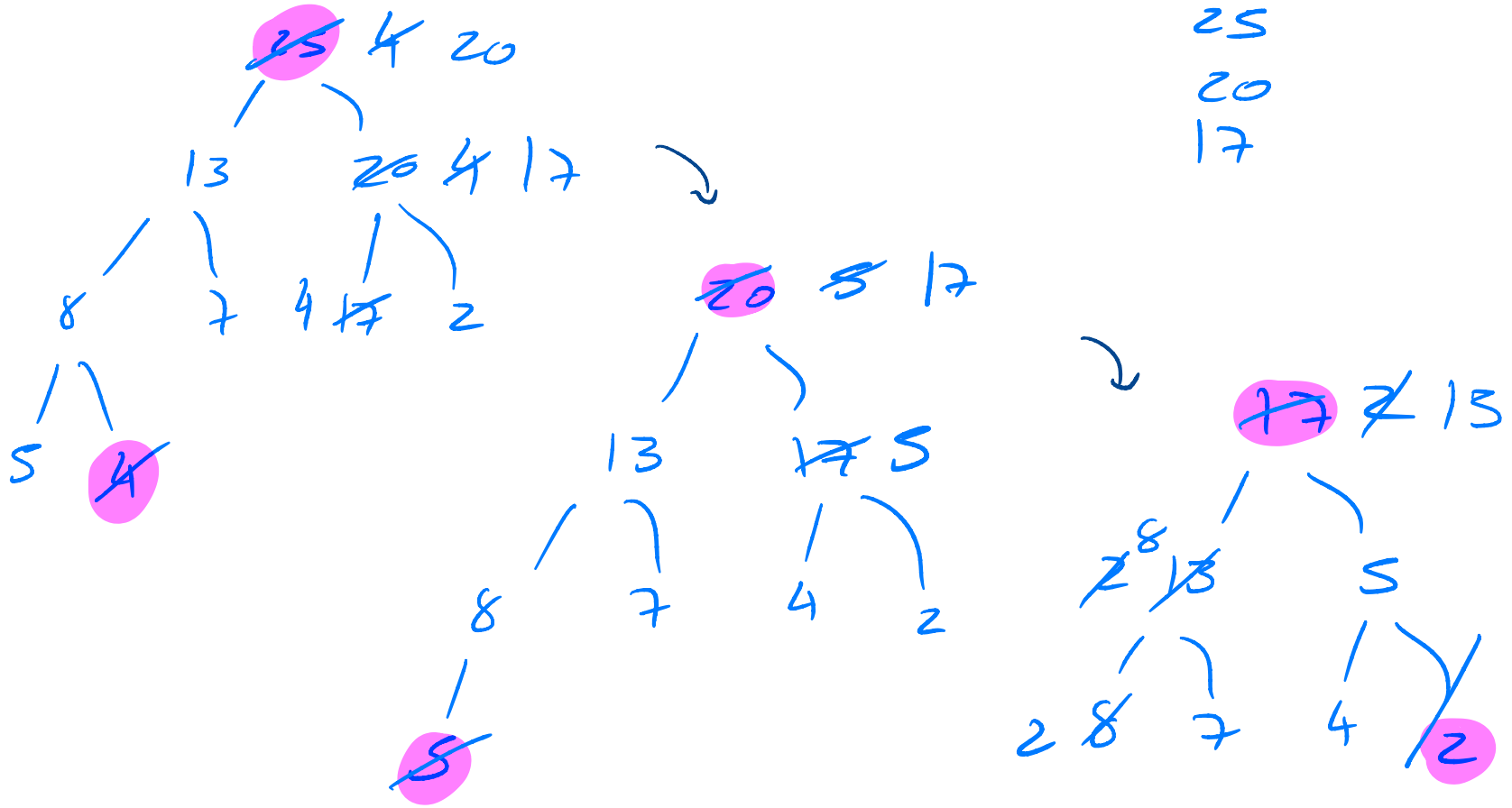


25

Heap Sort - Exemplo

$\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

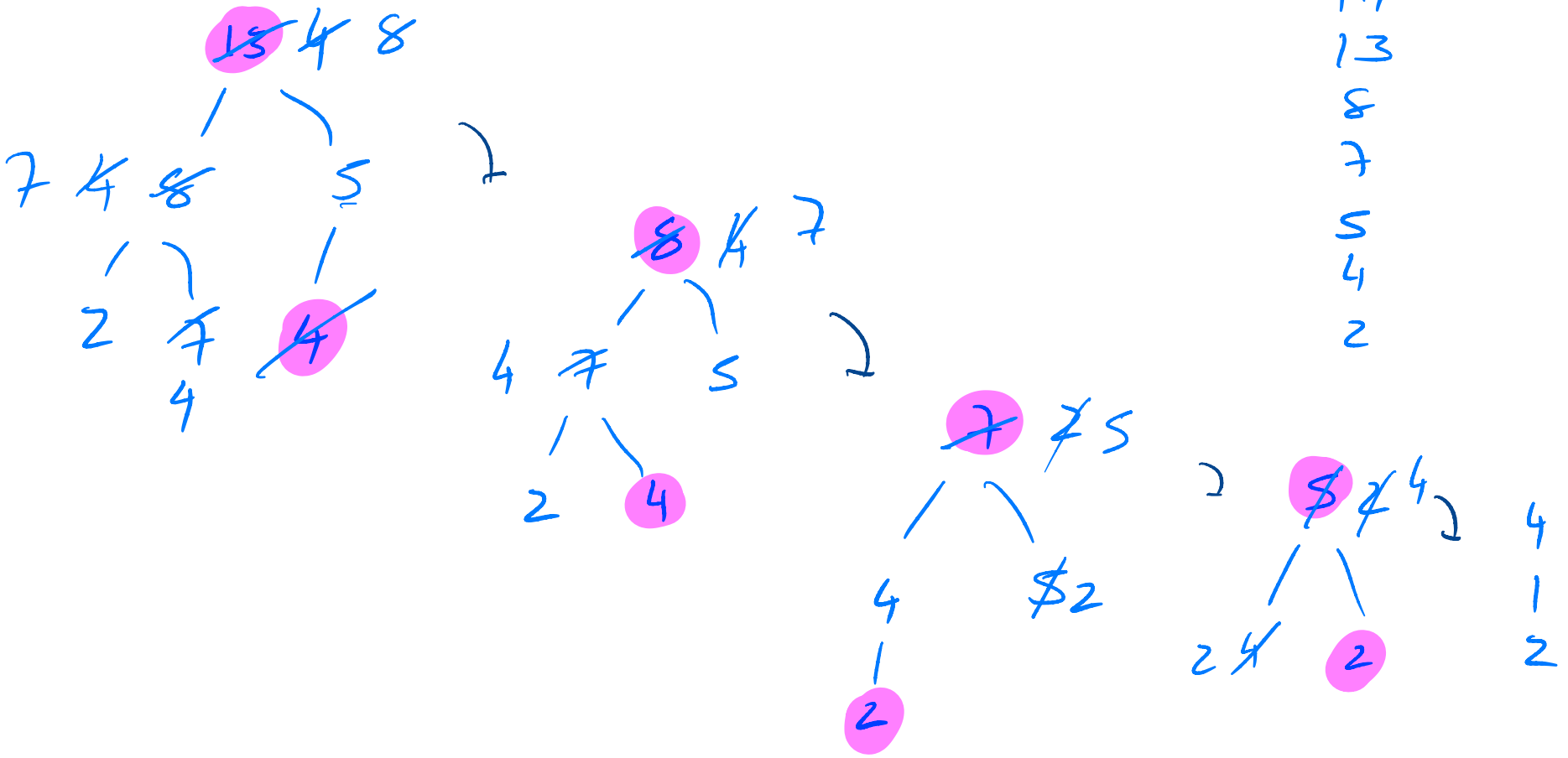
② Sequência de Max Heapify's



Heap Sort - Exemplo

$\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

② Sequência de Max Heapify's



Heap Sort

HeapSort(A)

for i ← 1 to A.length

Complexidade

Heap Sort

```
HeapSort(A)
    BuildMaxHeap(A)
    for A.length to i=2
        swap(A, 1, i)
        A.length--
        MaxHeapify(A,1)
```

Complexidade

$O(n) + O(n \log n)$

$= O(n \log n)$

Filas de Prioridade

- Max(A): Retorna a chave máxima da fila de prioridade
- Extract(A): retorna a chave máxima da fila de prioridade e remove a chave da fila
- Increase key (A, i, k): actualizar o valor da chave associado ao índice i para k
  Supondo que: $A[i] \leq k$
- Insert key (A, k): introduz a chave k no heap

Filas de Prioridade

- Max(A): retorna a chave máxima da fila de prioridade

  Max(A)

- Extract(A): retorna a chave máxima da fila de prioridade e remove a chave da fila

  Extract(A)

Filas de Prioridade

- Max(A): Retorna a chave máxima da fila de prioridade

```
Max(A)
    return A[1]
```

$O(1)$

- Extract(A): retorna a chave máxima da fila de prioridade e remove a chave da fila

```
Extract(A)
    v := A[1];
    A[1] := A[A.size()];
    A.size --;
    MaxHeapify(A, 1)
    return v
```

$O(\lg n)$

Filas de Prioridade

- IncreaseKey (A, i, k) : actualizar o valor da chave associado ao índice i para k

  Supondo que: $A[i] \leq k$

```
IncreaseKey (A, i, k)
    assert( A[i] ≤ k )
    if A[Parent(i)] ≥ k
       A[i] := k
    else {
       A[i] := A[Parent(i)]
       IncreaseKey (A, Parent(i), k)
    }
```

$O(\log n)$

Filas de Prioridade

- InsertKey (A, k) : introduz a chave k no heap

```
InsertKey(A, k)
    A.size + 1;
    A[A.size] := -∞
    IncreaseKey(A, A.size, k)
```

$O(\lg n)$