Aula 3

1. Teorema Mestre Generalizado

2. Heaps
    - Max Heapify
    - Build Max Heap
    - Heap Sort
    - Filas de Prioridade

Teorema Mestre (Simplificado)

Se $T(n) = a \cdot T(\lceil n/b \rceil) + O(n^d)$ p/ constantes $a > 0$, $b > 1$ e $d \geq 0$

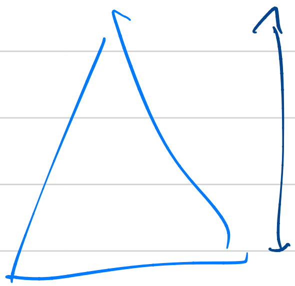
então:

$$T(n) = \begin{cases} O(n^d) & \text{se } d > \log_b a & \mathrm{I} \\ O(n^d \log n) & \text{se } d = \log_b a & \mathrm{II} \\ O(n^{\log_b a}) & \text{se } d < \log_b a & \mathrm{III} \end{cases}$$

I

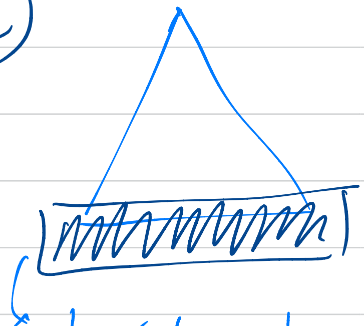
Custo da raiz domina o custo do problema

II



Custo distribuído

III



custo é dominado pelo nº de folhas

Teorema Mestre - Exemplo Código

V

```
int f(int n) {
  int i = 0, j = n;

  if (n <= 1) return 1;

  while(j > 0) {
    i++;
    j = j / 2;
  }

  for (int k = 0; k < 4; k++)
    j += f(n/2);

  while (i > 0) {
    j = j + 2;
    i--;
  }
  return j;
}
```

Teorema Mestre Generalizado

Se $T(n) = a\,T(n/b) + f(n)$ p/ constantes $a \geq 1$ e $b > 1$
então:

Condição de Regularidade

(I) Se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ p/ algum $\varepsilon > 0$, e se $a\,f(n/b) \leq c \cdot f(n)$ p/ algum $c < 1$ e n suficientemente grande,
então: $T(n) = \Theta(f(n))$

(II) Se $f(n) = \Theta(n^{\log_b a})$, então $T(n) = \Theta(n^{\log_b a} \cdot \log n)$

(III) Se $f(n) = O(n^{\log_b a - \varepsilon})$ p/ algum $\varepsilon > 0$, então: $T(n) = \Theta(n^{\log_b a})$

Teorema Mestre - Exemplo Código

V

```
int f(int n) {
  int i = 0, j = n;

  if (n <= 1) return 1;

  while(j > 0) {
    i++;
    j = j / 2;
  }

  for (int k = 0; k < 4; k++)
    j += f(n/2);

  while (i > 0) {
    j = j + 2;
    i--;
  }
  return j;
}
```

Teorema Mestre - Exemplo Código

V

```
int f(int n) {
  int i = 0, j = n;

  if (n <= 1) return 1;

  while(j > 0) {
    i++;
    j = j / 2;
  }

  for (int k = 0; k < 4; k++)
    j += f(n/2);

  while (i > 0) {
    j = j + 2;
    i--;
  }
  return j;
}
```

$O(\log n)$ (first while loop)

$O(\log n)$ (second while loop)

k	i	j
0	0	n
1	1	$n/2$
2	2	$n/4$
$\vdots$		
k	k	$n/2^k$

Cond. de paragem:

$$\frac{n}{2^k} = 1$$

$$\Leftrightarrow n = 2^k \Leftrightarrow k = \log_2 n$$

$$T(n) = 4 \cdot T(n/2) + O(\log n)$$

$$\log(n) \in O(n^{2-\varepsilon})$$

$$T(n) = O(n^2)$$

Teorema Mestre (Simplificado)

Se $T(n) = a \cdot T(\lceil n/b \rceil) + O(n^d)$ p/ constantes $a>0$, $b>1$ e $d \geq 0$

então:

$$T(n) = \begin{cases} O(n^d) & \text{se } d > \log_b a \\ O(n^d \log n) & \text{se } d = \log_b a \\ O(n^{\log_b a}) & \text{se } d < \log_b a \end{cases}$$

Teorema Mestre (Simplificado)

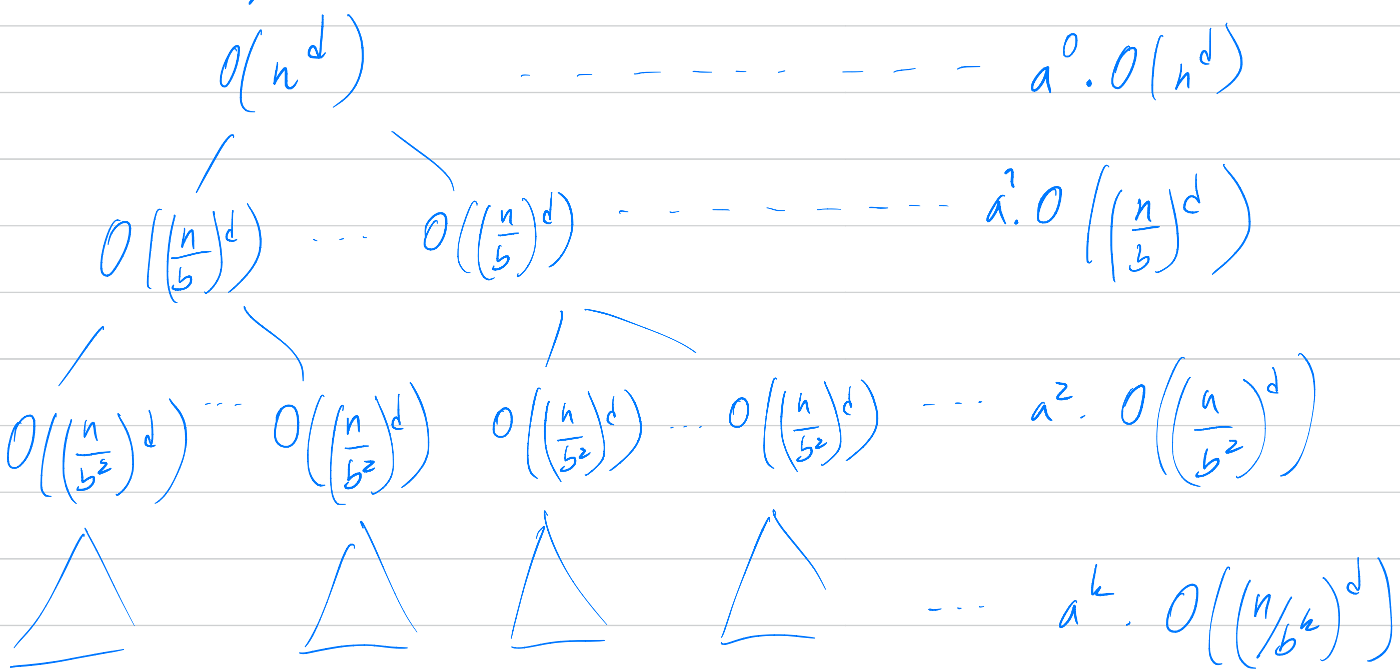
Se $T(n) = a \cdot T(\lceil n/b \rceil) + O(n^d)$ p/ constantes $a > 0$, $b > 1$ e $d \geq 0$

então:

$$T(n) = \begin{cases} O(n^d) & \text{se } d > \log_b a \\ O(n^d \log n) & \text{se } d = \log_b a \\ O(n^{\log_b a}) & \text{se } d < \log_b a \end{cases}$$

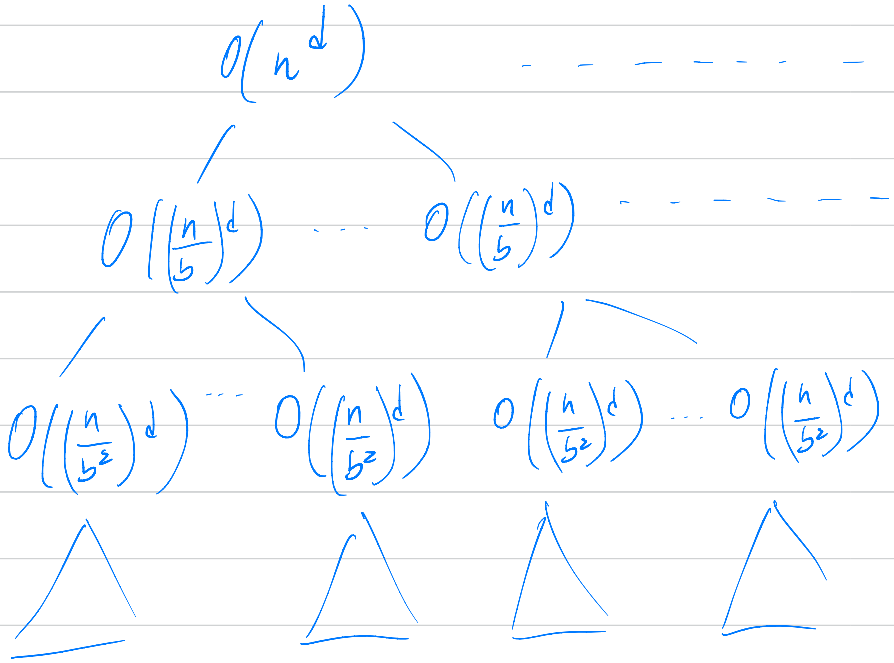
Prova [Sketch]:

- Suponhamos q̄ n é uma potência de b ($n = b^k$, para algum k)



Prova [Sketch]:

• Suponhamos q n é uma potência de b ($n = b^k$, para algum k)



$a^0 \cdot O(n^d)$

$a^1 \cdot O\left(\left(\frac{n}{b}\right)^d\right)$

$a^2 \cdot O\left(\left(\frac{n}{b^2}\right)^d\right)$

... $a^k \cdot O\left(\left(n/b^k\right)^d\right)$

Altura da árvore:

$$\frac{n}{b^k} = 1$$

$$\iff n = b^k$$

$$\iff k = \log_b n$$

Custo da Árvore:

$$T(n) = \sum_{k=0}^{?} a^k \cdot O\left(\left(n/b^k\right)^d\right)$$

altura da árvore

Custo da Árvore:

$$T(n) = \sum_{k=0}^{\log_b n} a^k \cdot O\left(\left(\frac{n}{b^k}\right)^d\right)$$

Custo da Árvore:

$$T(n) = \sum_{k=0}^{\log_b n} a^k \cdot O\left(\left(n/b^k\right)^d\right)$$

$$= \sum_{k=0}^{\log_b n} O\left(a^k \cdot \left(\frac{n}{b^k}\right)^d\right)$$

$$= n^d \sum_{k=0}^{\log_b n} O\left(\left(\left(\frac{a}{b^d}\right)\right)^k\right)$$

$$= O\left(n^d \sum_{k=0}^{\log_b n} \left(\left(a/b^d\right)\right)^k\right)$$

3 casos

I - $(a/b^d) < 1$

II - $(a/b^d) = 1$

III - $(a/b^d) > 1$

Custo da Árvore:

$$T(n) = O\left(n^d \sum_{k=0}^{\log_b n} \left(\left(a/b^d\right)\right)^k\right)$$

3 casos

- I – $(a/b^d) < 1$
- II – $(a/b^d) = 1$
- III – $(a/b^d) > 1$

I $\quad a/b^d < 1 \rightsquigarrow$ A série converge!

$$T(n) = O(n^d)$$

II $\quad a/b^d = 1$

$$T(n) = O(n^d \cdot \log_b n)$$

III $\quad a/b^d > 1 \rightsquigarrow$ o custo da série é dominado pelo custo do último termo.

$$\left(a/b^d\right)^{\log_b n} = a^{\log_b n} \cdot \frac{1}{\left(b^{\log_b n}\right)^d} = \frac{a^{\log_b n}}{n^d}$$

$$T(n) = O\left(\cancel{n^d} \cdot \frac{a^{\log_b n}}{\cancel{n^d}}\right)$$

$$= O\left(a^{\log_b n}\right)$$

$$= O\left(n^{\log_b a}\right)$$

Notes

- $a^{\log_b n} = a^{\log_a n / \log_a b}$

$$= \left(a^{\log_a n}\right)^{1/\log_a b}$$

$$= n^{1/\log_a b}$$

$$= n^{\log_b a}$$

$$\log_a b = \log_b b / \log_b a$$

$$= 1/\log_b a$$

Heaps

Definição [Heap]

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

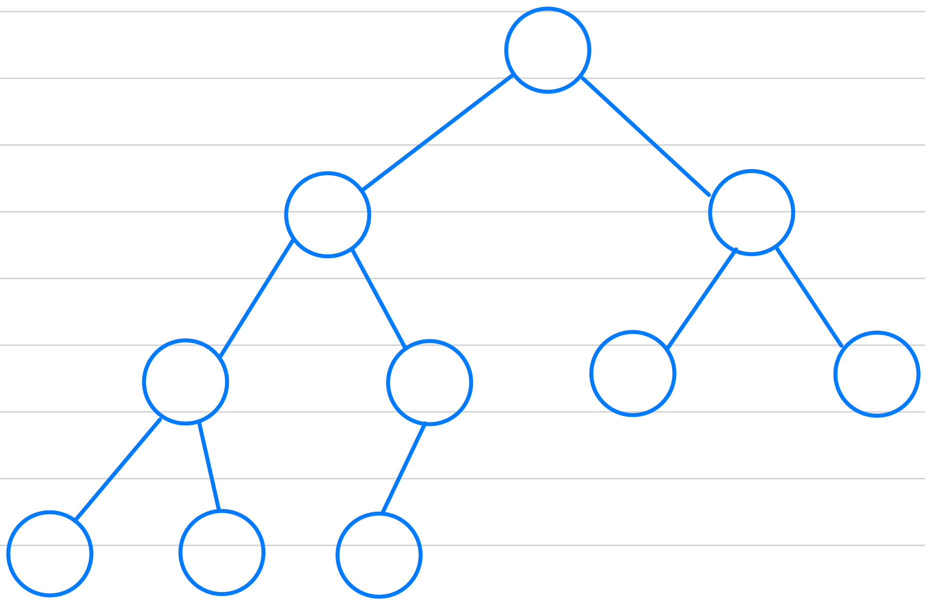
$$\forall\, 1 < i \leq n.\ A[Parent(i)] \geq A[i]$$

onde:

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

Exemplo:

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



Heaps

Definição [Heap]

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

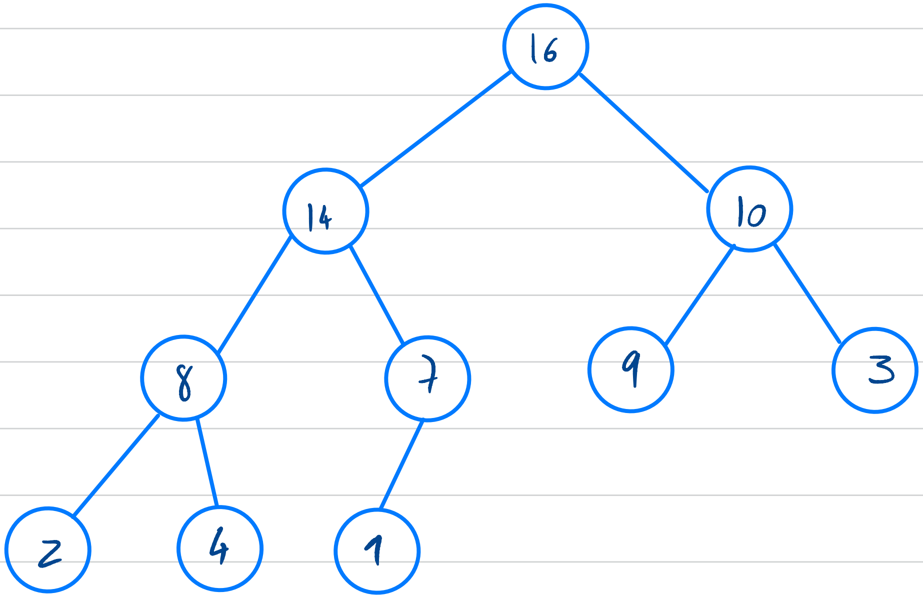
$$\forall\, 1 < i \leq n.\ A[Parent(i)] \geq A[i]$$

onde:

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

Exemplo:

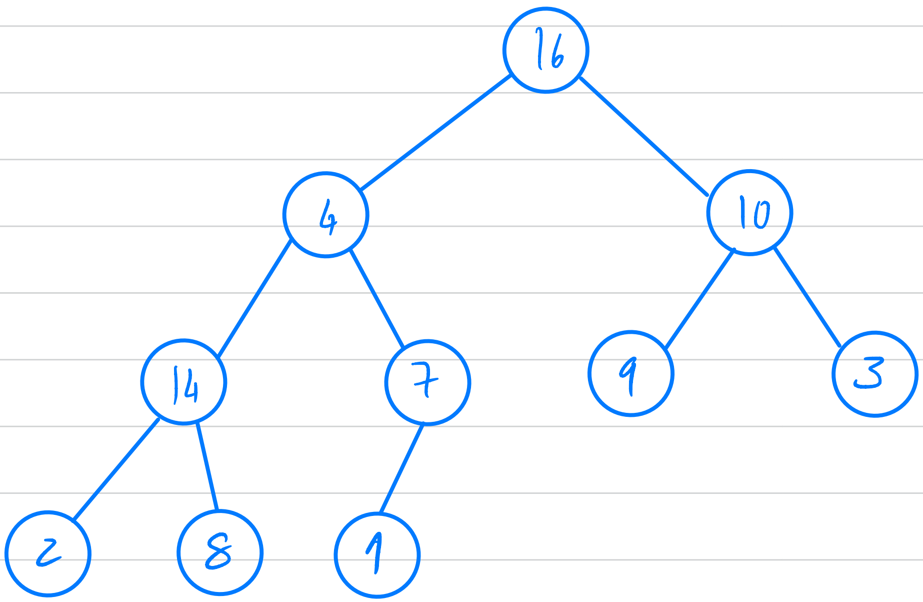
16	14	10	8	7	9	3	2	4	1
1	2	3	4	5	6	7	8	9	10



Max Heapify

Exemplo:

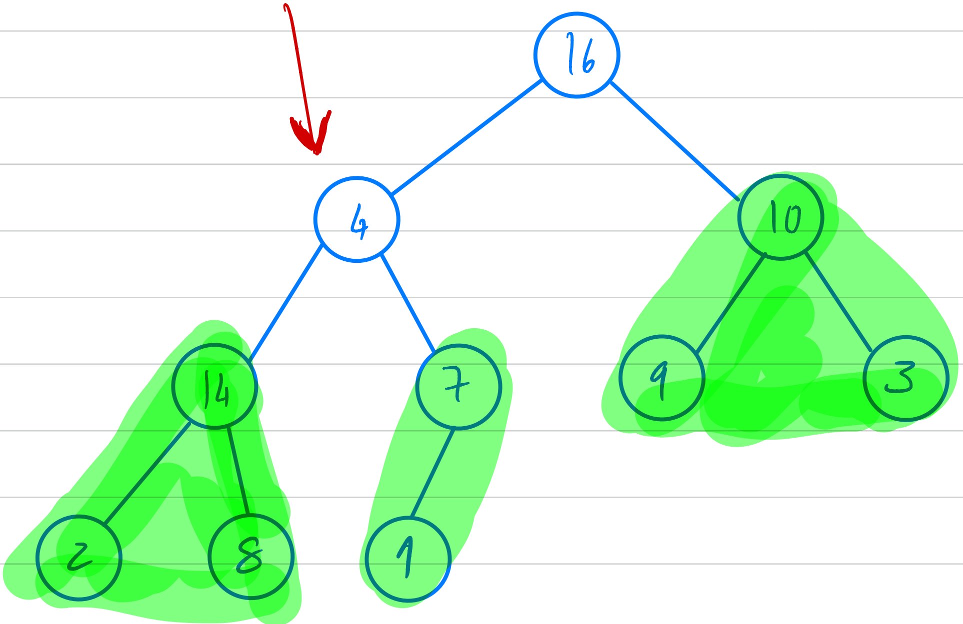
16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10



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:

  - $i = 5$

    $$A[\text{Parent}(i)] = A[2] = 4 \not\geq A[i] = 7$$

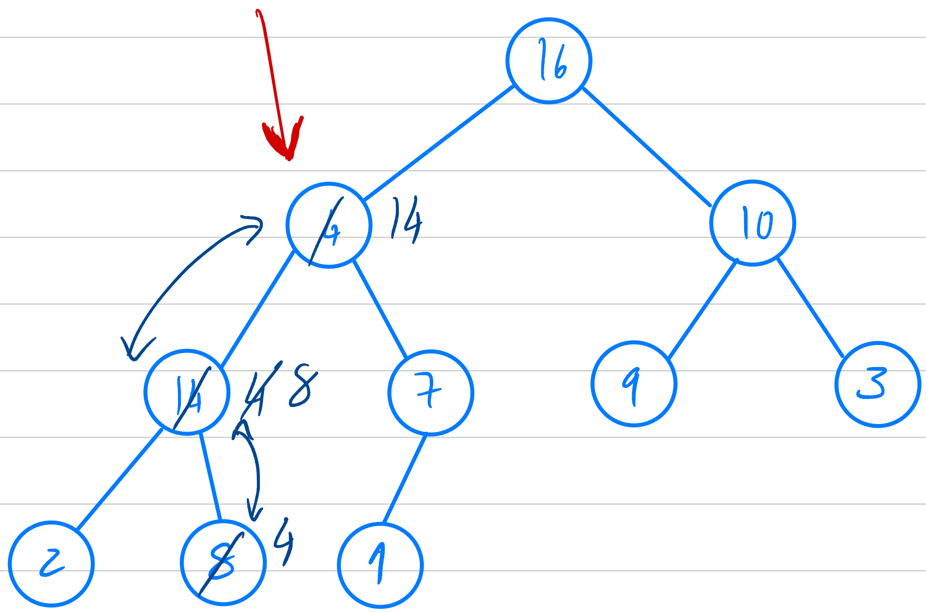
  - $i = 4$

    $$A[\text{Parent}(i)] = A[2] = 4 \not\geq A[i] = 14$$

Max Heapify

Exemplo:

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



• Condição de heap falha para:

- $i = 5$

  $A[\text{Parent}(i)] = A[2] = 4 \ngtr A[i] = 7$

- $i = 4$

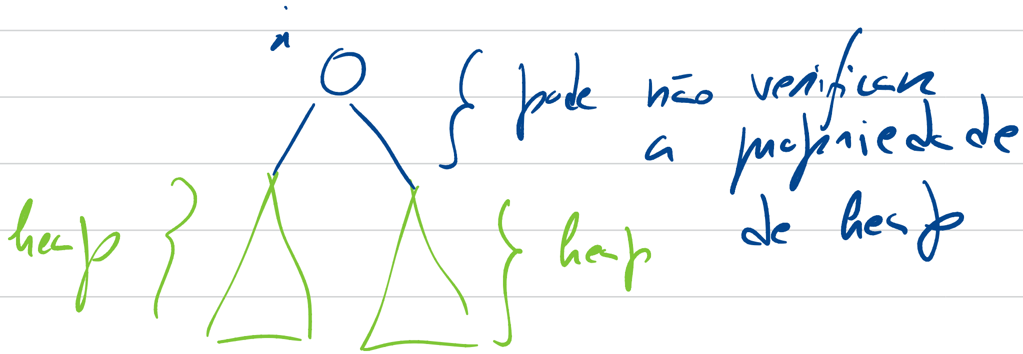
  $A[\text{Parent}(i)] = A[2] = 4 \ngtr A[i] = 14$

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)
```



Propriedades de Heaps

- Altura máxima de um heap com $\underline{n}$ elementos:

$$\lfloor \log n \rfloor$$

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

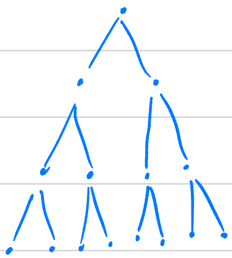
- Nº de elementos de um heap de altura h: x

$$< x \leq$$

Propriedades de Heaps

- Altura máxima de um heap com <u>n</u> elementos:

$$\lfloor \log n \rfloor$$



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

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

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

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

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

$$\lfloor \log x \rfloor = h$$

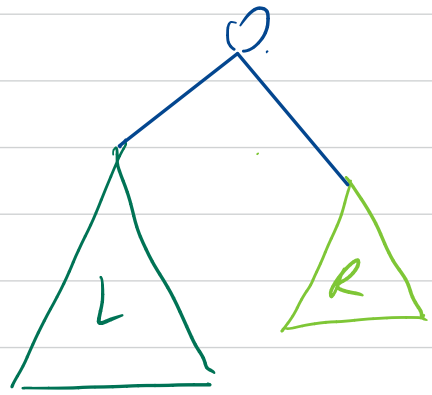
Max Heapify

```
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



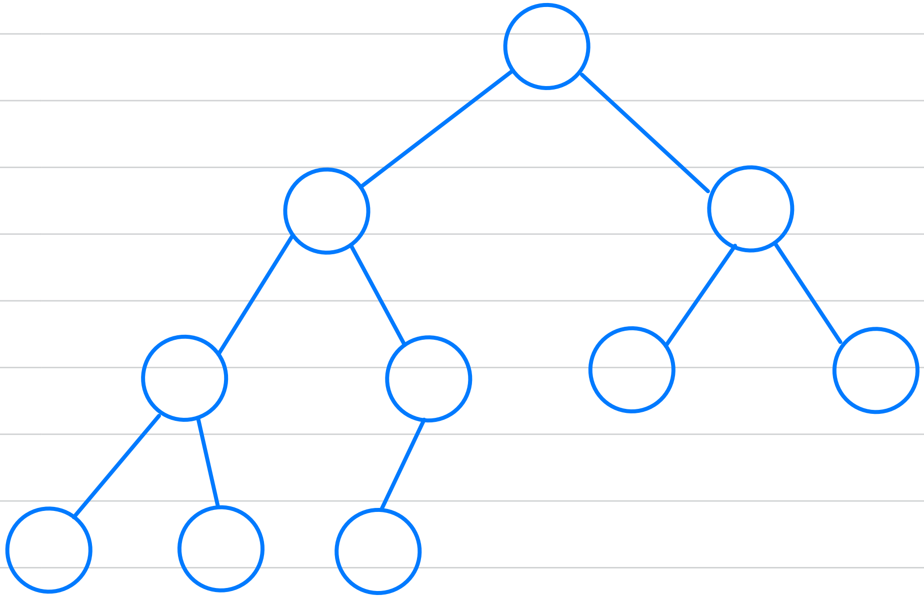
- Qual é o maior nº de vezes q podemos invocar recursivamente a função MaxHeapify?

A altura do heap: $\log n$

Construção de Heaps

Exemplo:

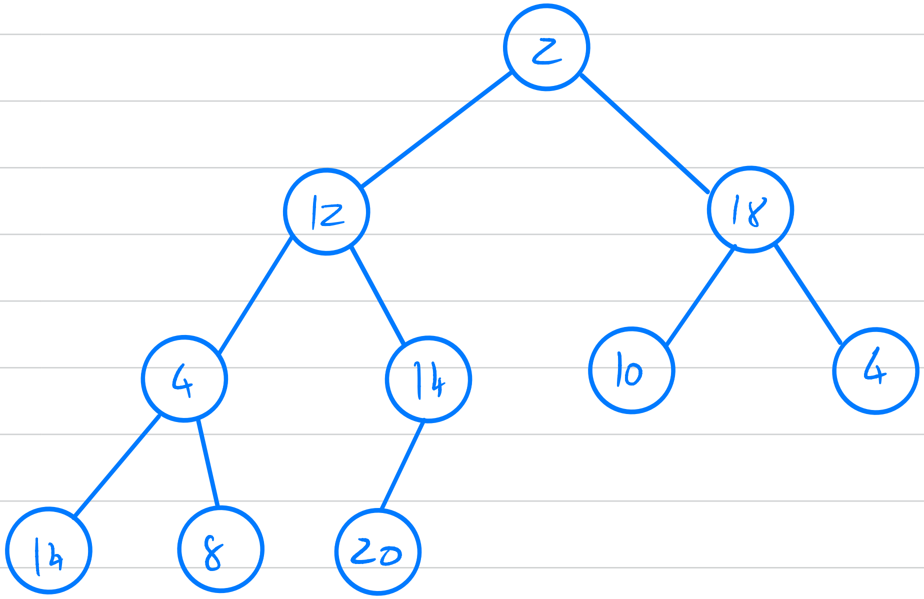
2	12	18	4	14	10	4	14	8	20
1	2	3	4	5	6	7	8	9	10



Construção de Heaps

Exemplo:

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

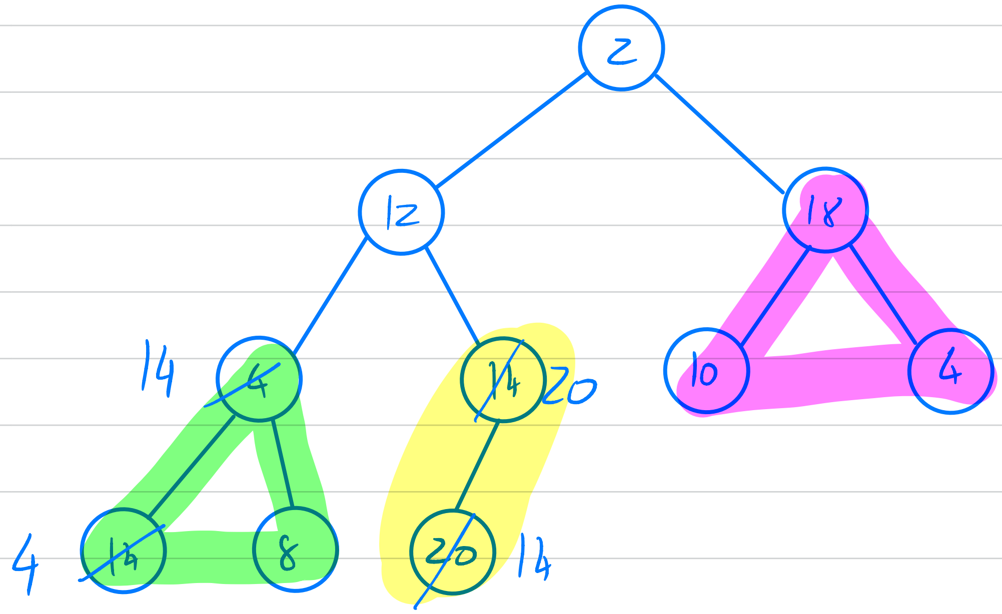


- Construímos um heap aplicando a operação de Max Heapify de baixo para cima da direita para a esquerda.

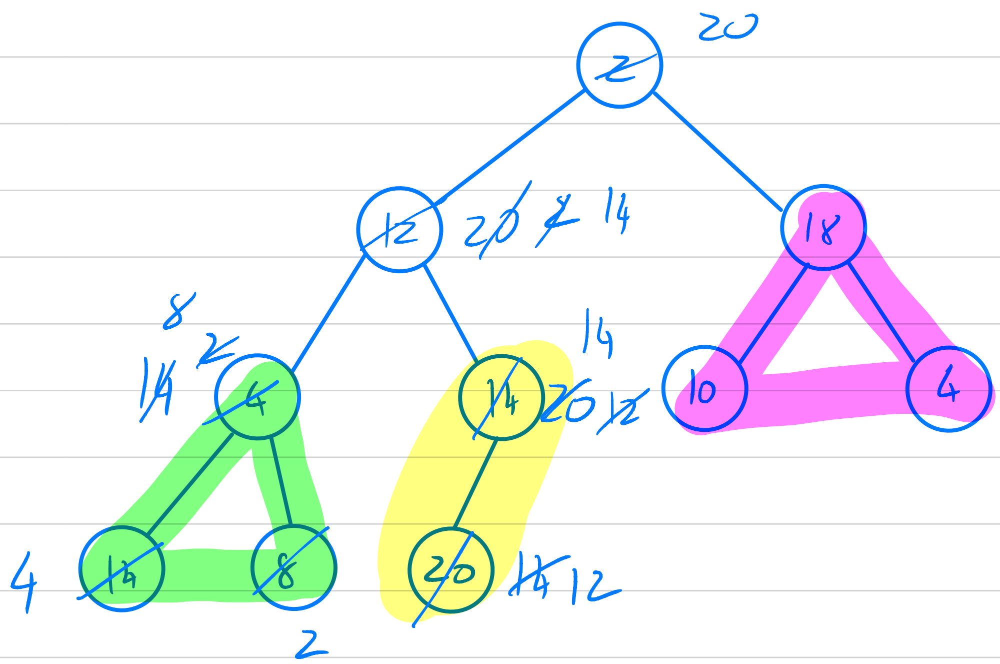
Construção de Heaps

Exemplo:

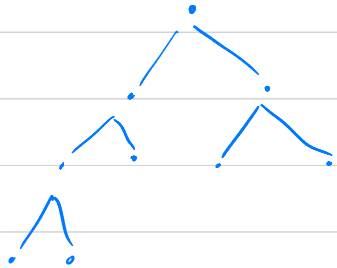
2	12	18	4	14	10	4	14	8	20
1	2	3	4	5	6	7	8	9	10



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



Propriedades de Heaps



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

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

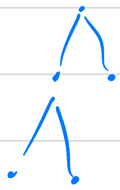
Prova

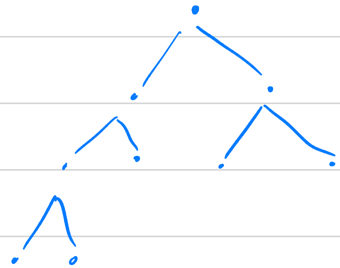
- 2 casos: $i \leq \lfloor n/2 \rfloor \Rightarrow i$ tem filhos
  $i > \lfloor n/2 \rfloor \Rightarrow i$ não tem filhos

(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





- 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

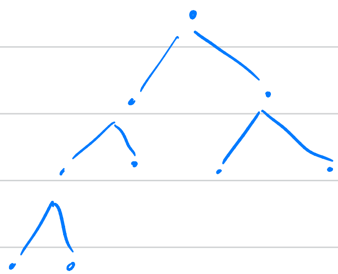
(b) $i > \lfloor n/2 \rfloor \Rightarrow i$ não tem filhos

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

(B.2) i é ímpar: $\exists m.\ i = 2 * m + 1$

Propriedades de Heaps

- Índice do primeiro nó com filhos num heap com $\underline{n}$ 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: $\exists m.\ n = 2 * m$

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

$i > m$

$left(i) = 2 * i > 2 * m = n$

(B.2) i é ímpar: $\exists m.\ i = 2 * m + 1$

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

$\Rightarrow i > m$

$left(i) = 2 * i \Rightarrow left(i) > 2 * m$

$> 2 * m + 1$

porque left(i) é par

Construção de Heaps - Build Max Heap

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

Complexidade [1ª Aproximação]

$O(n \lg n)$

Complexidade [limite apertado]

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \underbrace{trees(n, h)}_{\text{nº de árvores de altura } h} \times O(h)$$

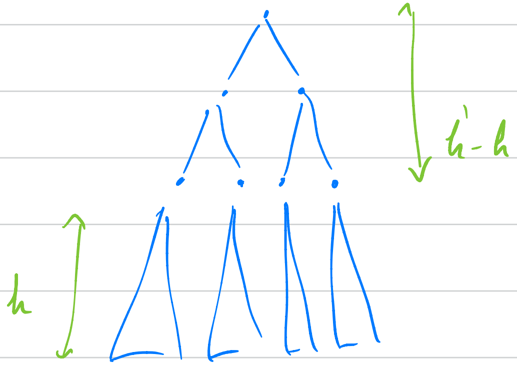
$O(h)$ ⇒ complexidade do Max Heapify numa árvore de altura h

↳ nº de árvores de altura h

Propriedades de Heaps

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

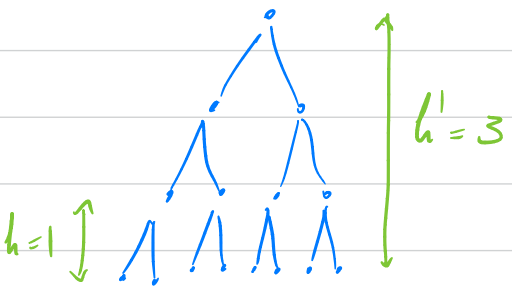
Prova:



nº total de elementos:

Nº de árvores de altura h?

$$2^{\lfloor \log_2 n \rfloor - h} < 2^{(\log_2 n) - h}$$

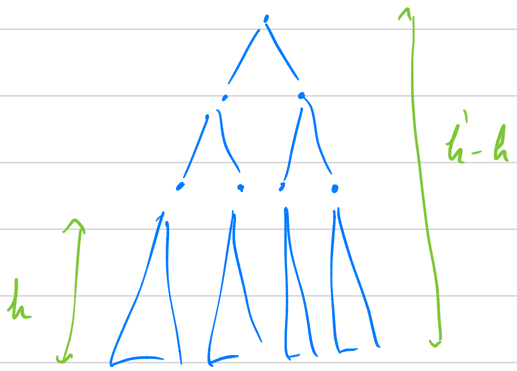


Nº de árvores de altura h:

Propriedades de Heaps

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

Prova:



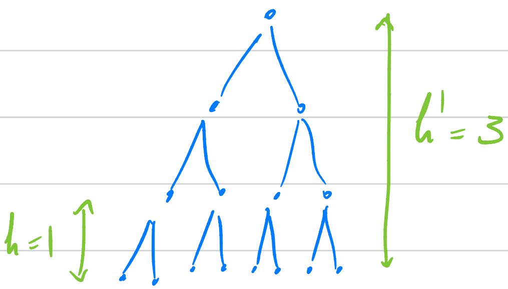
nº total de elementos: $2^{h'+1}-1$

Nº de árvores de altura h? $2^{h'-h}$

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

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

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



Nº de árvores de altura h: 4

Construção de Heaps - Build Max Heap

BuildMaxHeap(A)
    k := ⌊A.length/2⌋
    for i := k to 1
        MaxHeapify(A, i)

Complexidade [limite apertado]

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

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

Construção de Heaps - Build Max Heap

```
BuildMaxHeap(A)
    k := ⌊A.length/2⌋
    for i := k to 1
        MaxHeapify(A, i)
```

Complexidade [limite apertado]

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

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

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

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

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

$$= O(2n)$$

$$= O(n)$$

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

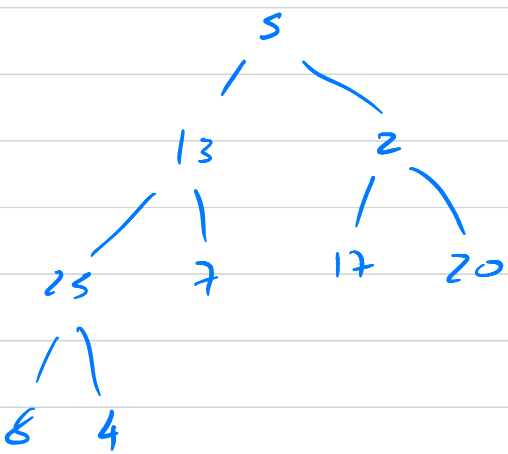
$$\sum_{i=0}^{\infty} i\, a^{i-1} = \frac{+1}{(1-a)^2}$$

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

Heap Sort - Exemplo

< 5, 13, 2, 25, 7, 17, 20, 8, 4 >

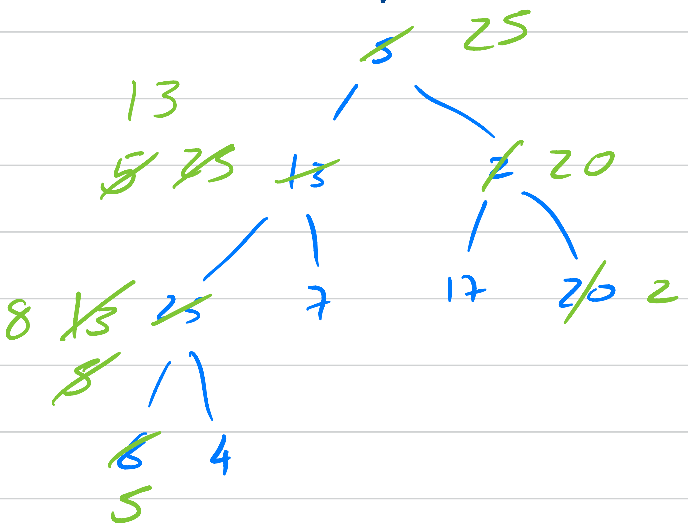
1 Build Max Heap



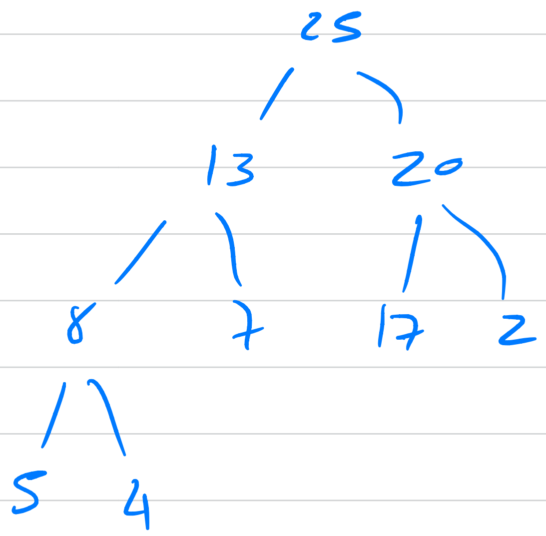
Heap Sort - Exemplo

< 5, 13, 2, 25, 7, 17, 20, 8, 4 >

① Build Max Heap



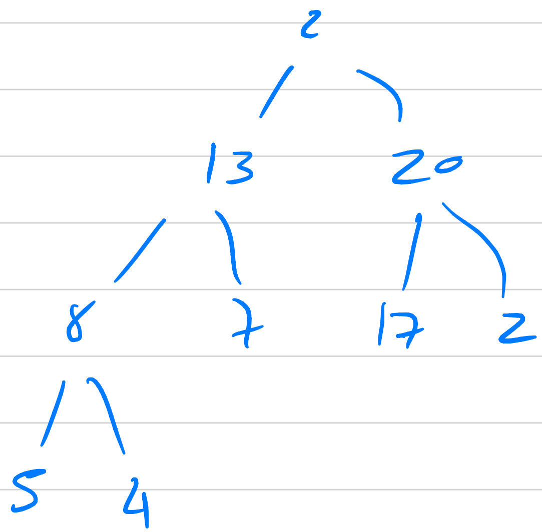
< 25, 13, 20, 8, 7, 17, 2, 5, 4 >



Heap Sort - Exemplo

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

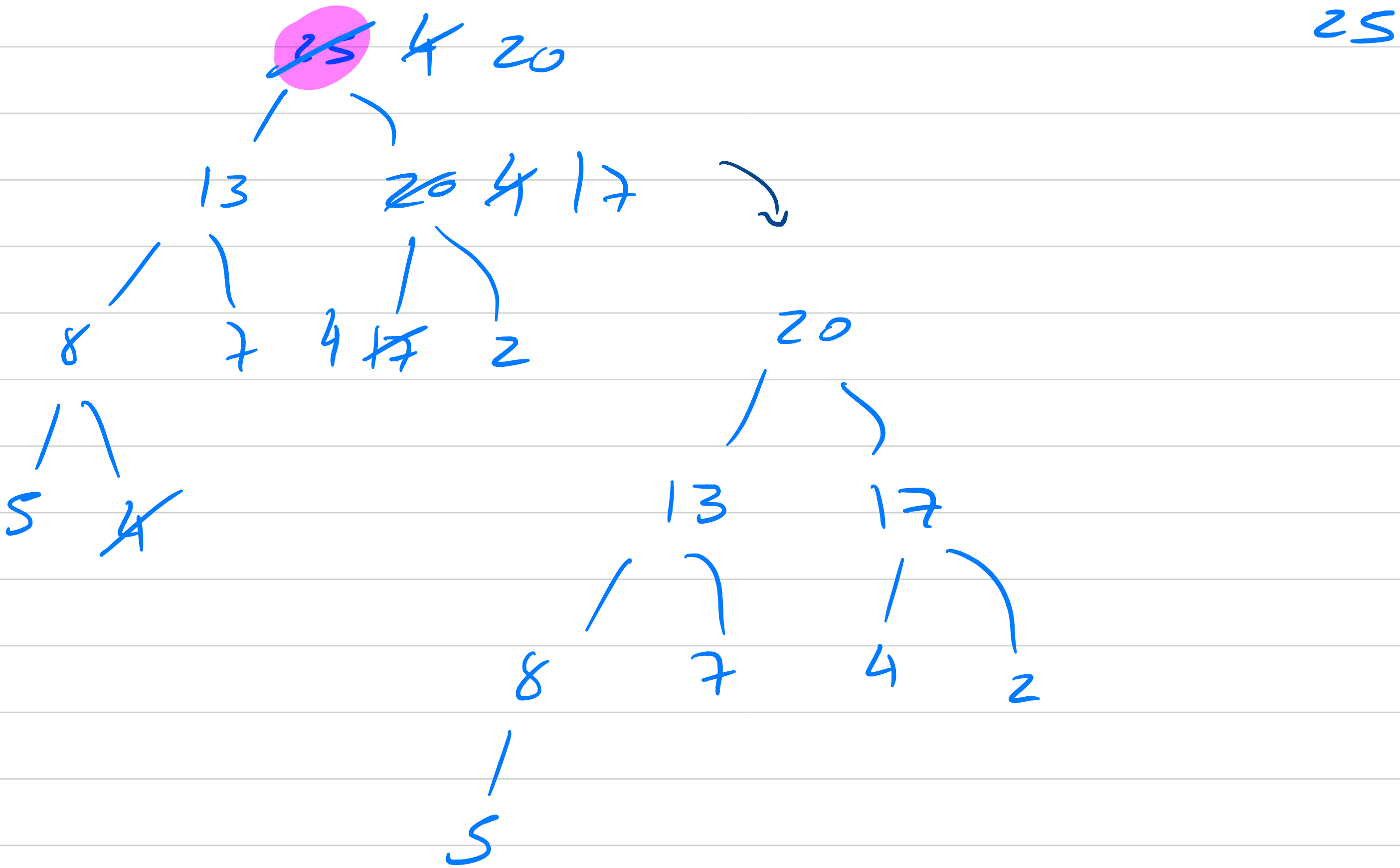
2) Sequência de MaxHeapify's



Heap Sort - Exemplo

< 5, 13, 2, 25, 7, 17, 20, 8, 4 >

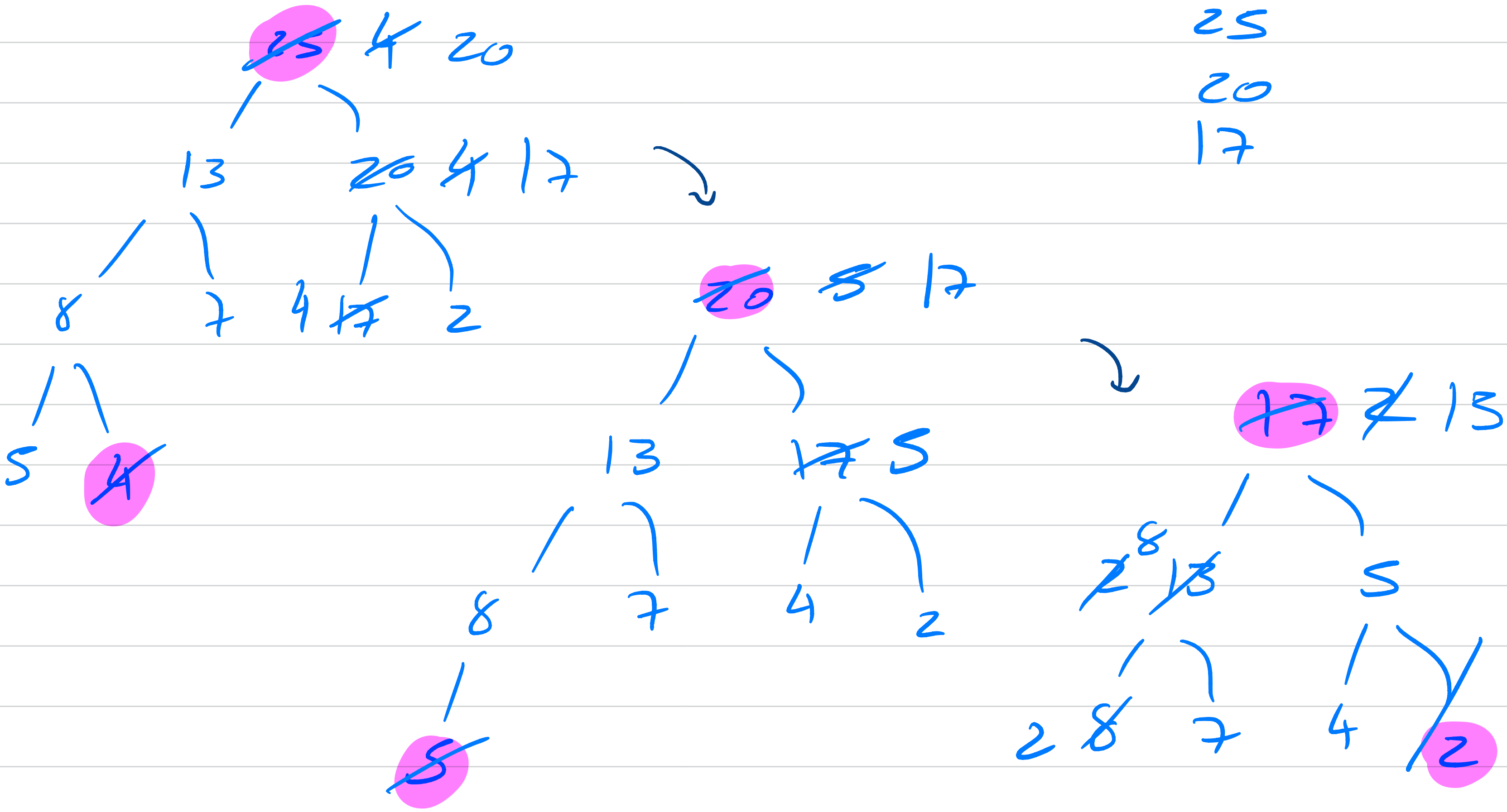
(2) Sequência de Max Heapify's



Heap Sort - Exemplo

< 5, 13, 2, 25, 7, 17, 20, 8, 4 >

(2) Sequência de Max Heapify's



25

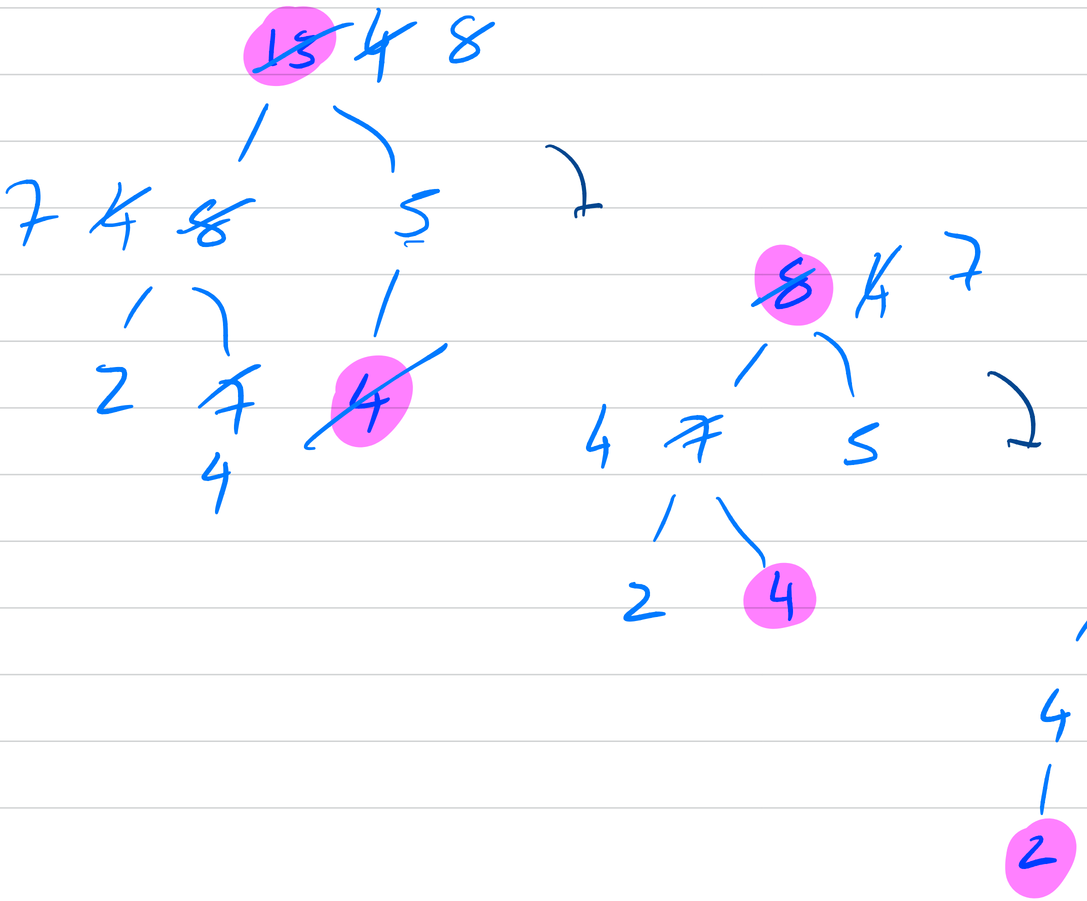
20

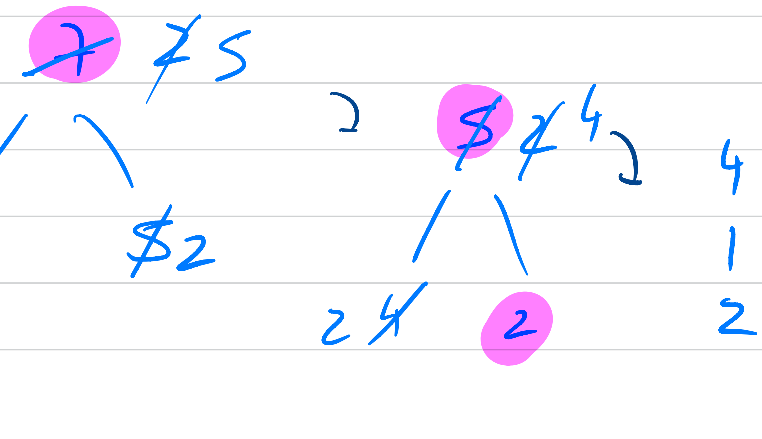
17

Heap Sort - Exemplo

$< 5, 13, 2, 25, 7, 17, 20, 8, 4 >$

(2) Sequência de MaxHeapify's





25
20
17
13
8
7
5
4
2

Heap Sort

Complexidade

HeapSort (A)

for i=1 to A.length

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 $\underline{i}$ para k

  Supondo que: $\underline{\underline{A[i] \leq k}}$
- $Insert\,key(A, k)$: introduz a chave $\underline{k}$ 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
```

$O(\log n)$

Filas de Prioridade

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

  Supondo que: $\underline{A[i] \leq k}$

```
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 $\underline{k}$ no heap

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

$$\underline{\underline{O(\lg n)}}$$