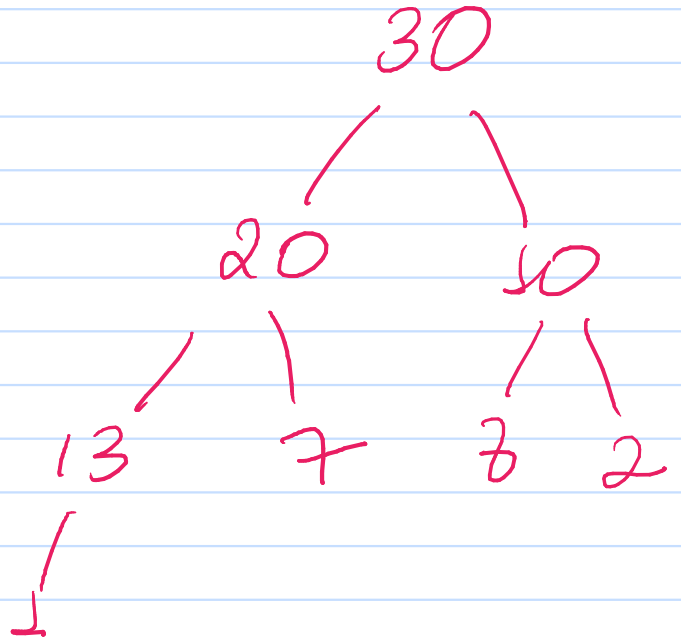
# Heaps (de máximo)

↳ Estrutura implementada com vetores que podem ser pensados como árvore binária em que todo elemento é maior que seus filhos.

| 30 | 20 | 10 | 13 | 7 | 8 | 2 | ↓ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |




↳ elemento e:

- A[L i/2 J] é sua mãe (A[L i/2 J] ≥ A[i])
- A[2i] e A[2i+1] são seus filhos

# Operações em heaps (complexidade)

↳ Consulta ao maior elemento:

Retorna A[1]

$O(1)$

↳ Alteração de valor / prioridade:

- Aplicar a rotina "Subir"

ou

- Aplicar a rotina "Descer"

para manter o heap

Descer (i, n)

$j \leftarrow 2i$

Se $j \leq n$

Se $j < n$

Se $A[j+1] > A[j]$

$j \leftarrow j+1$

Se $A[i] < A[j]$

$A[i] \leftrightarrow A[j]$

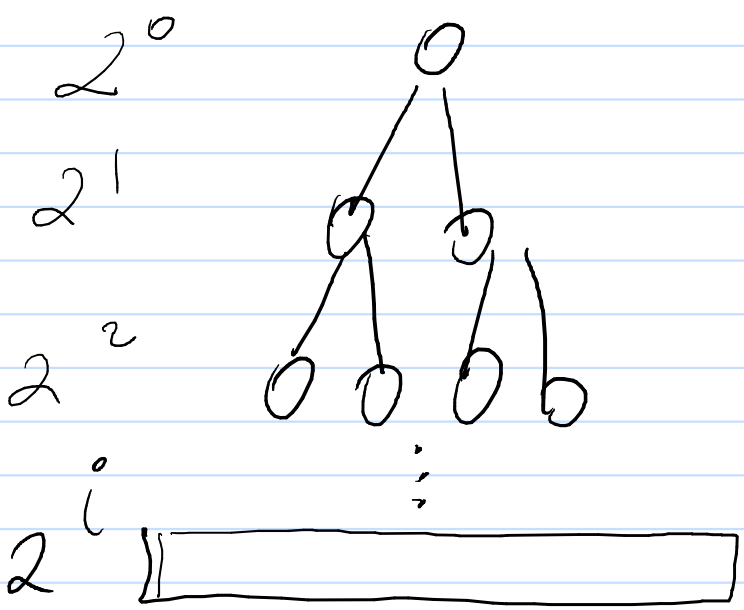
Descer (j, n)

$O(a)$

↳ altura da árvore

# Altura de uma árvore binária completa

Qtd de elem. no nível $i$




$\left\lceil \frac{n}{2^i} \right\rceil$

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

$\left\lceil \frac{n}{2^2} \right\rceil$

$\left\lceil \frac{n}{2} \right\rceil$

$2^0 + 2^1 + 2^2 + \dots + 2^i = n$ ⇨ total de elem. na árvore

$2^{i+1} - 1 = n$

$\Rightarrow 2^{i+1} = n + 1$

$\Rightarrow 2 \cdot 2^i = n - 1$

$\Rightarrow 2^i = \frac{n+1}{2} = \left\lceil \frac{n}{2} \right\rceil$

⇨ Qtd de elementos no último nível da árvore (metade do valor total)

$\log 2^i = \log \frac{n+1}{2}$

$\Downarrow$

$i = \log n + 1 - \log 2$

$\Downarrow$

$i = (\log n + 1) - 1$

$\Downarrow$

$i = O(\log n)$

$\downarrow$

Altura de árvore binária completa

↳ Inserção de um novo elemento $O(\log n)$

- Coloca na última posição do vetor.

- Conserta o heap, através da rotina "subir"

↳ Remoção do maior $O(\log n)$

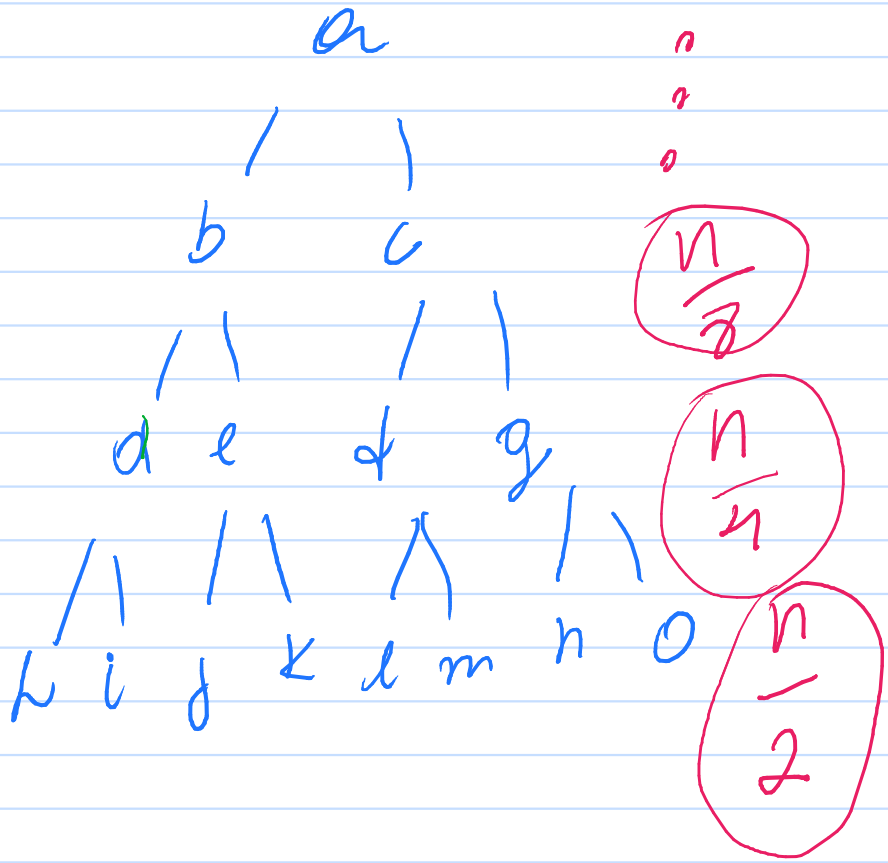
- Substitui o maior pelo último elemento

- Conserta, utilizando a rotina "Descer".

## Construção de um heap

- Aplica descer p/ cada elemento:

```
Stua-heap (A[1..n])
Para i ← ⌊n/2⌋ até 1
    Descer (i, n)
```




Complexidade:

$$T(n) \leq \sum_{h=2}^{\lfloor \log n \rfloor} \frac{n}{2^h} \cdot O(h) = O\left(n \underbrace{\sum_{h=1}^{\infty} \frac{h}{2^h}}_{2}\right) = O(n)$$

# Heap sort

```
HeapSort (A[1..n])
    Construi-heap (A[1..n])          O(n log n)
    m ← n
    Enquanto m > 1                   } O(n log n)
        A[1] <-> A[m]
        m ← m - 1
        Descer (1, m)
```

$O(n \log n)$

Invariante: $A[1..m]$ é heap de máximo contendo os $m$ menores elementos de $A$ e

$A[m+1..n]$ contém os $n-m$ maiores elementos de $A$, ordenados.