

Selection Sort,
Heap Sort &

Fila de Prioridade

Disclaimer

Slides baseados nos slides dos Professores

- Cândida Nunes da Silva
- Carla N. Lintzmayer
- Cid Corvelho
- Lehilton Pedrosa
- Orlando Lee

Selection Sort

Ideia:

- o subvetor $A[1..i-1]$ está ordenado
- $\max(A[1..i-1]) \leq \min(A[i..n])$
- Substituímos a posição $A[i]$ pelo mínimo em $A[i..n]$

Selection Sort (A, n)

Para $i \leftarrow 1$ até $n-1$

$\text{min} \leftarrow i$

para $j \leftarrow i+1$ até n

Se $A[j] < A[\text{min}]$

$\text{min} \leftarrow j$

$A[i] \leftrightarrow A[\text{min}]$

Análise de tempo de Execução

Selection Sort (A, n)

Para $i \leftarrow 1$ até $n-1$

$\text{min} \leftarrow i$

 para $j \leftarrow i+1$ até n

 Se $A[j] < A[\text{min}]$

$\text{min} \leftarrow j$

$A[i] \leftrightarrow A[\text{min}]$

Análise de tempo de Execução

Selection Sort (A, n)

Para $i \leftarrow 1$ até $n-1$

$\Theta(n)$

min $\leftarrow i$

$\Theta(n)$

para $j \leftarrow i+1$ até n

Se $A[j] < A[\text{min}]$

min $\leftarrow j$

$A[i] \leftrightarrow A[\text{min}]$

$\Theta(n)$

$O(n) \cdot \Theta(n) = O(n^2)$

$T(n)$ é $O(n^2)$

Versão Alternativa

Vamos reescrever esse algoritmo para ordenar a partir do final

Ex

A =

2	10	5	4	15	12	20	11	10
---	----	---	---	----	----	----	----	----

Ideia

- $A[i+1..n]$ está ordenado
- $A[1..i] \leq A[i+1]$

Selection-Sort (A, n)

Para $i \leftarrow n$ decrescendo até 2
 $\text{max} \leftarrow \text{Maximum}(A, i)$
 $A[i] \leftrightarrow A[\text{max}]$

Maximum(A, i)

$\text{max} \leftarrow i$

 para $j \leftarrow i-1$ decrescendo até 1

 se $A[j] > A[\text{max}]$

$\text{max} \leftarrow j$

 Devolva max

Análise de Selection-Sort

Assuma que $\text{Maximum}(A, i)$ leva tempo $O(t(i))$

Selection-Sort (A, n)

Para $i \leftarrow n$ decrescendo até 2
 $\text{max} \leftarrow \text{Maximum}(A, i)$
 $A[i] \leftrightarrow A[\text{max}]$

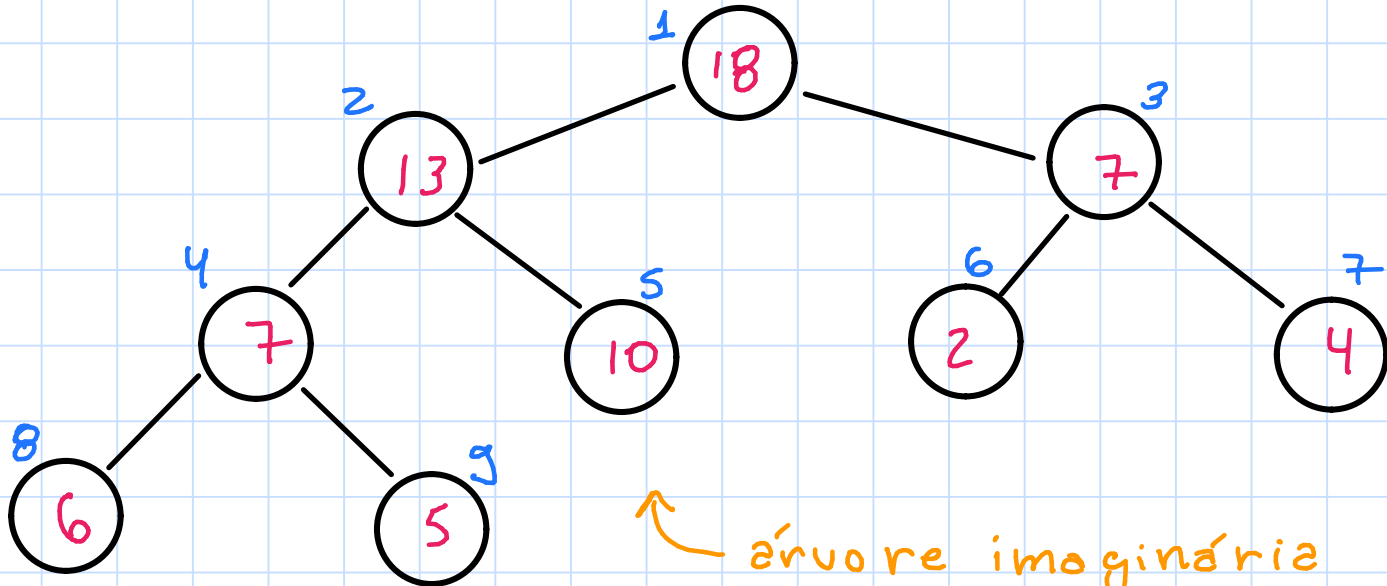
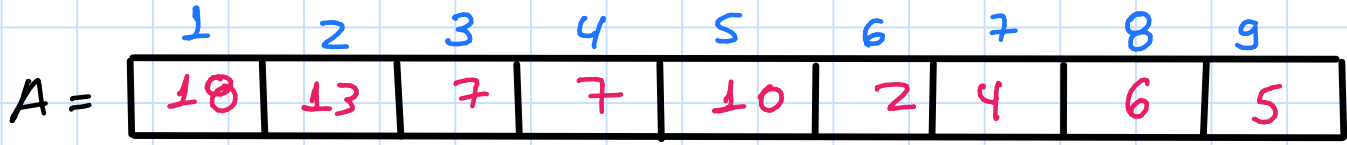
$$T(n) = \sum_{i=2}^n O(t(i)) \leq \sum_{i=2}^n O(n) = O(n^2)$$

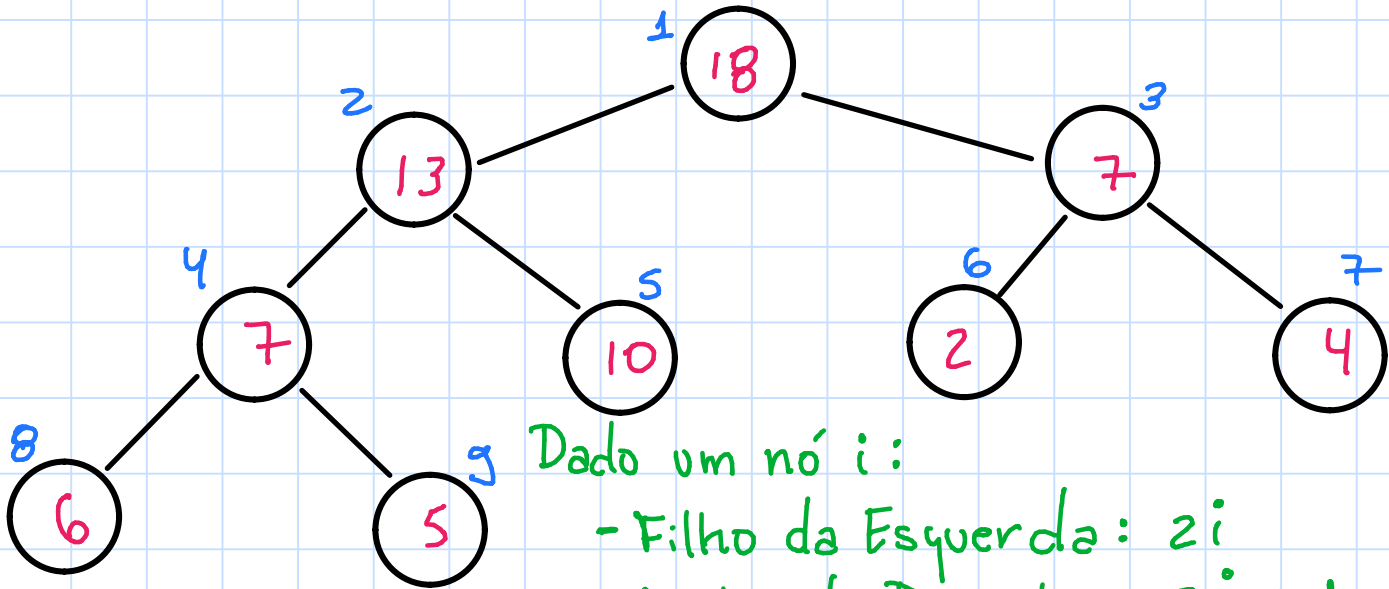
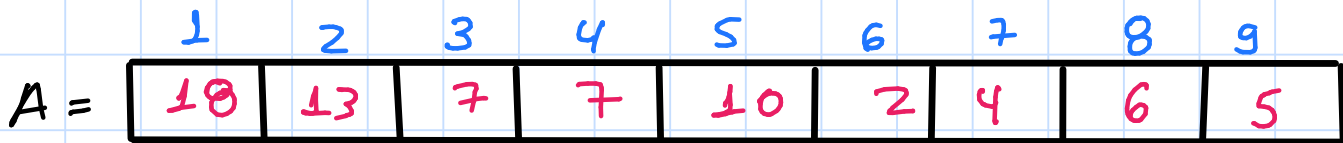
Vamos tentar melhorar Maximum

(Max)-Heap

Heap

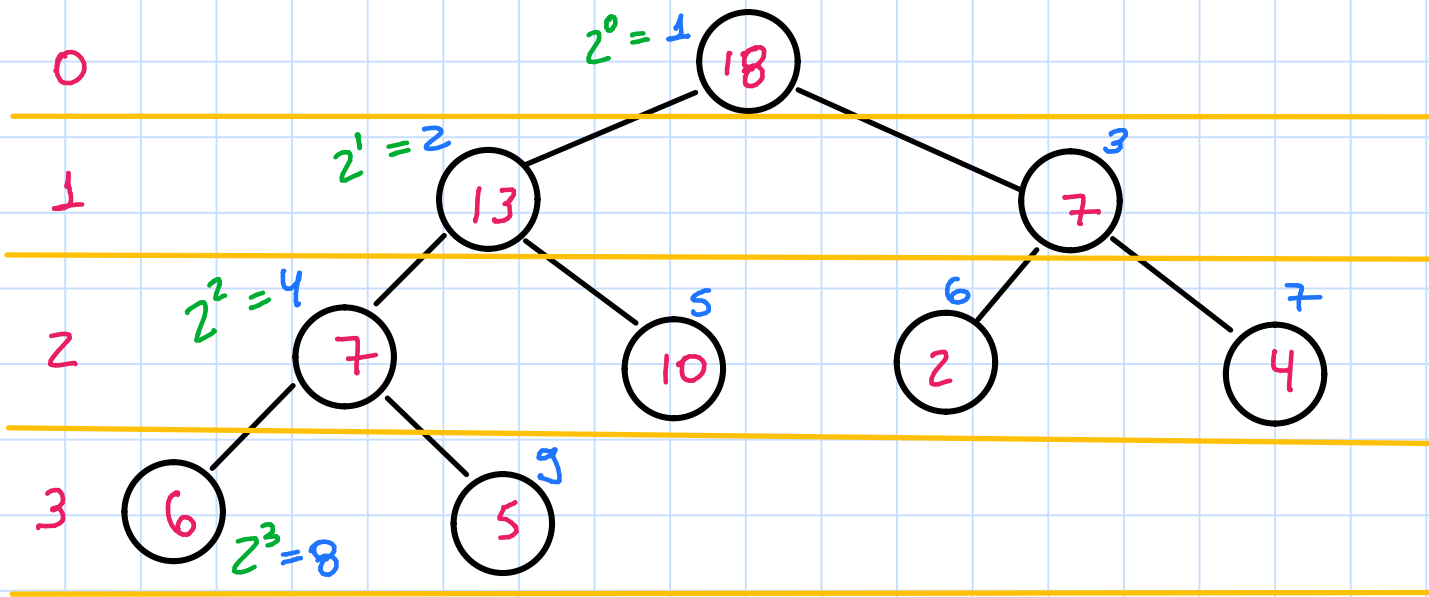
- Um heap é uma árvore binária armazenada em um vetor $A[1..n]$





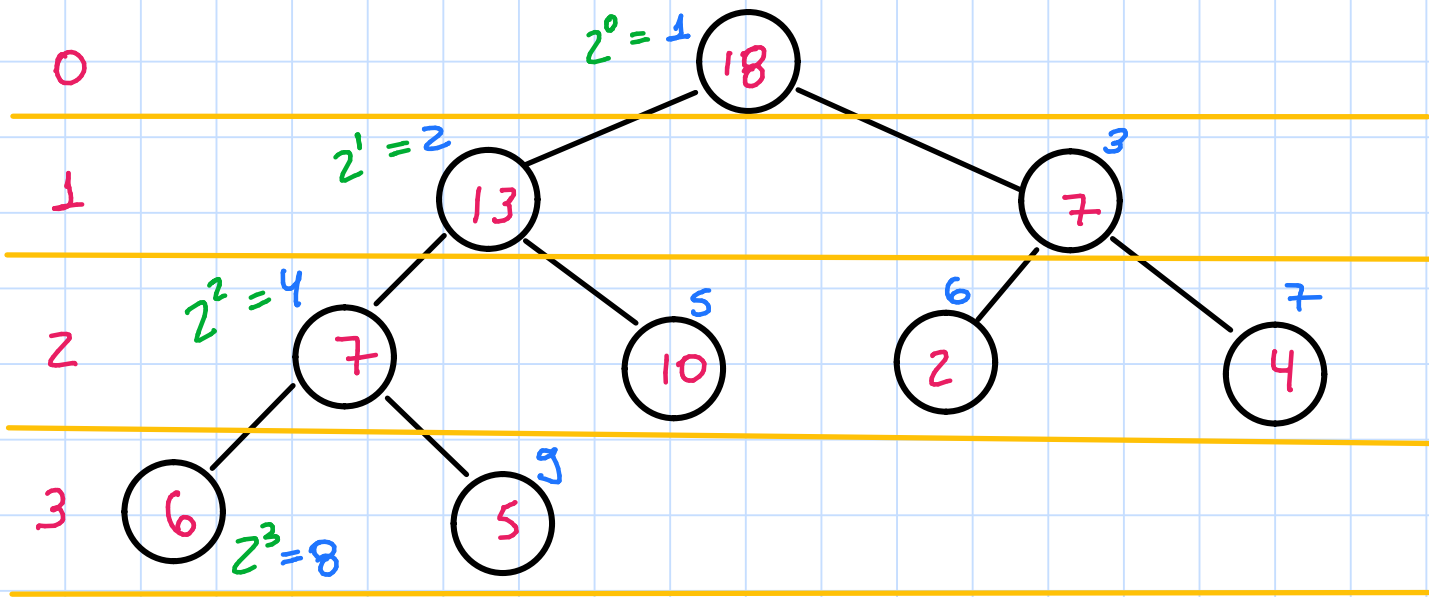
Dado um nó i :

- Filho da Esquerda: $2i$
- Filho da Direita: $2i + 1$
- Pai: $\lfloor i/2 \rfloor$



Um heap é uma árvore completa

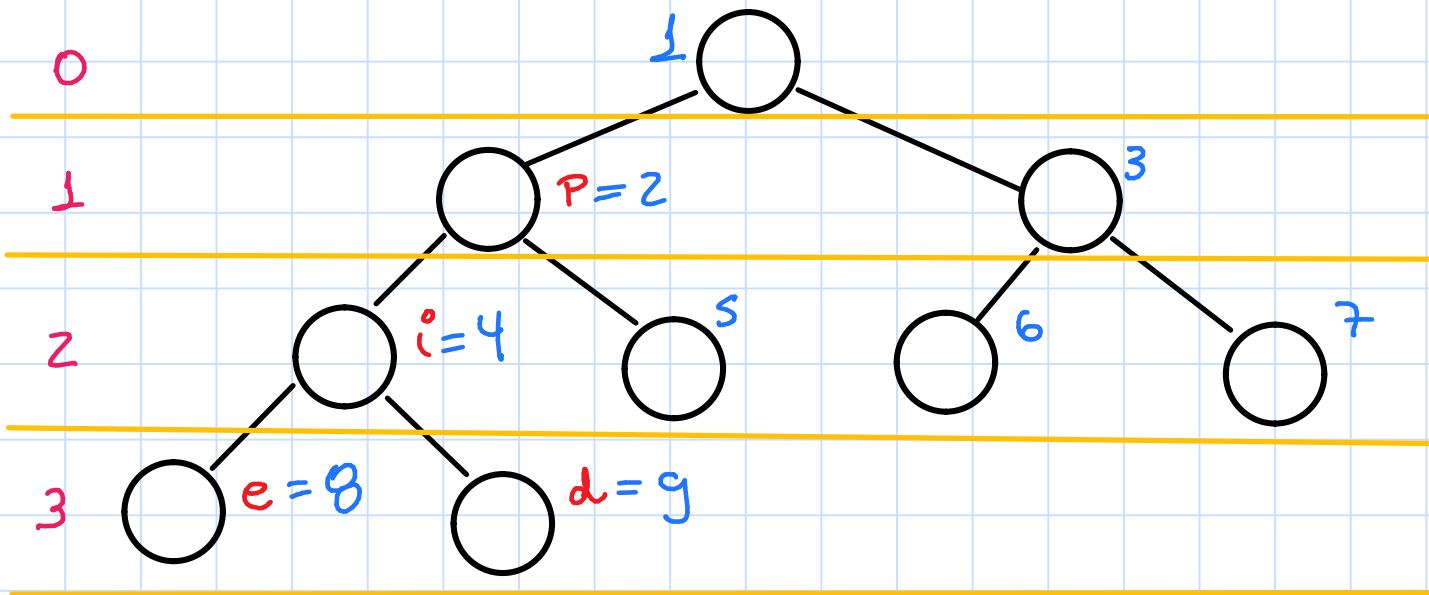
- Cada nível $l=0,1,2,\dots$ tem 2^l nós (com exceção do último)



Seja i um vértice no nível l

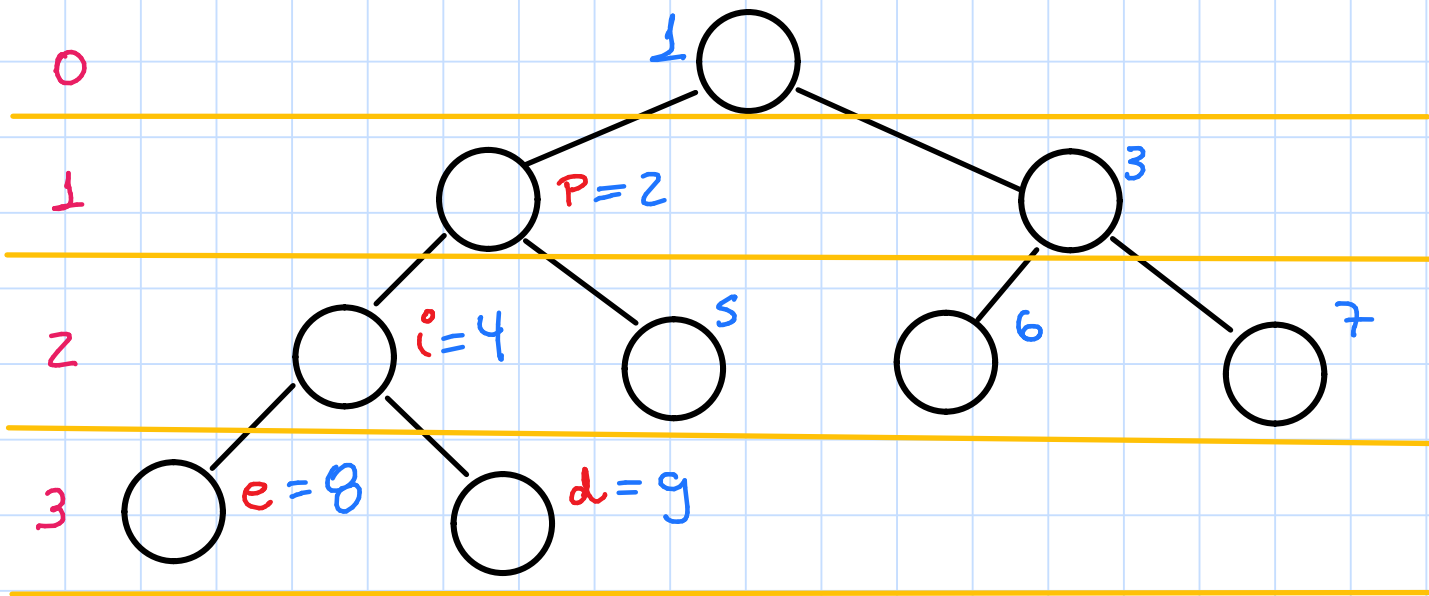
$$i = \sum_{i=0}^{l-1} 2^i + a = 2^l - 1 + a, \text{ onde } 1 \leq a \leq 2^l$$

Seja e , d e p os rótulos, respectivamente, do filho da esquerda, fil. da direita e do pai \Rightarrow



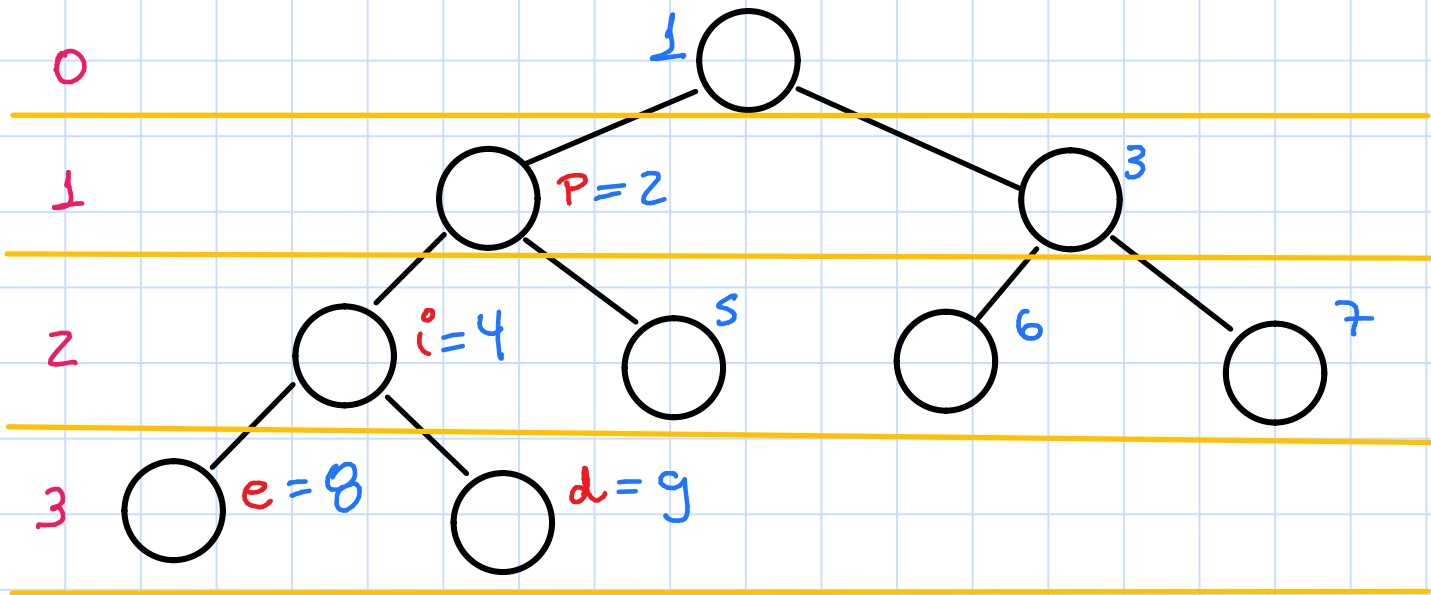
$$i = 2^l + a - 1$$

$$\begin{aligned} \text{filho da esquerda} &= 2^{l+1} + 2(a-1) = 2(2^l + a - 1) \\ &= 2i \end{aligned}$$



$$i = 2^l + a - 1$$

$$\begin{aligned}
 \text{filho da direita} &= 2^{l+1} + 2(a-1) + 1 \\
 &= 2(2^l + a - 1) + 1 \\
 &= 2^i + 1
 \end{aligned}$$



se i é o filho da esquerda

$$i = 2p \iff \frac{i}{2} = p$$

Se i é o filho da direita

ímpar \rightarrow $\underline{i = 2p + 1} \iff \frac{i-1}{2} = p$

$$p = \left\lfloor \frac{i}{2} \right\rfloor$$

O nível de um nó i é $\lfloor \lg i \rfloor$

- Note que o rótulo do primeiro nó em um nível l é

$$i = 2^l + a - 1 = 2^l + 1 - 1 = 2^l$$

- Se i está em um nível l , então

$$2^l \leq i < 2^{l+1}$$

inteiro

$$\begin{aligned} \lg 2^l &\leq \lg i \\ l &\leq \lg i \\ l &\leq \lfloor \lg i \rfloor \end{aligned}$$

$$\begin{aligned} \lg i &< \lg 2^{l+1} \\ \lg i - 1 &< l \end{aligned}$$

AFIRMAÇÃO $\lfloor \lg i \rfloor \leq l \Rightarrow$

$$\lg i - 1 < l$$

se $\lg i$ é inteiro:

$\lg i - 1 < l \Rightarrow \lg i \leq l$, que podemos escrever como

$$\lfloor \lg i \rfloor \leq l$$

se $\lg i$ não é inteiro

$$\lg i - 1 < l \Rightarrow \lceil \lg i - 1 \rceil \leq l \Leftrightarrow \lceil \lg i \rceil - 1 \leq l$$

$$\lfloor \lg i \rfloor + 1 - 1 \leq l \Leftrightarrow \lfloor \lg i \rfloor \leq l$$

O nível de um nó i é $\lfloor \lg i \rfloor$

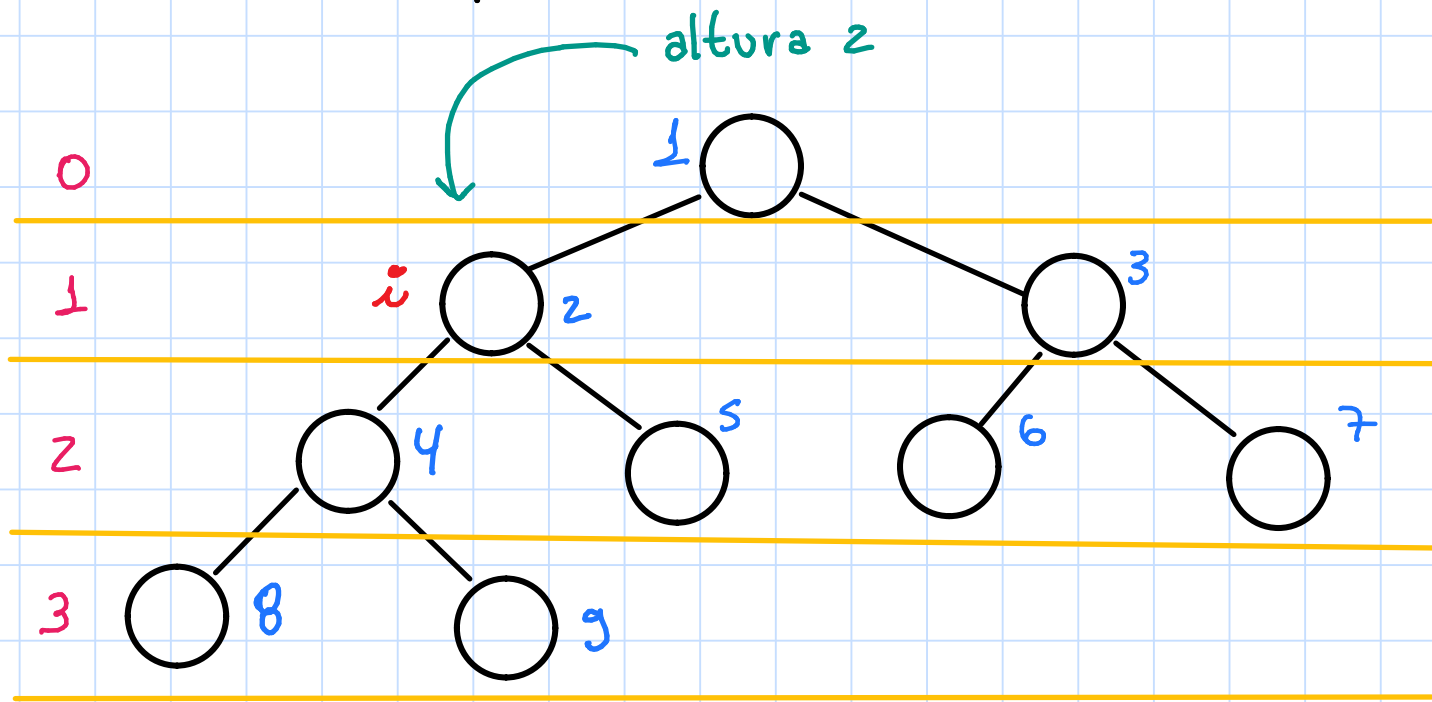
\Rightarrow como consequência, a árvore de uma heap de n elementos tem

níveis

$$\lfloor \lg n \rfloor + 1$$

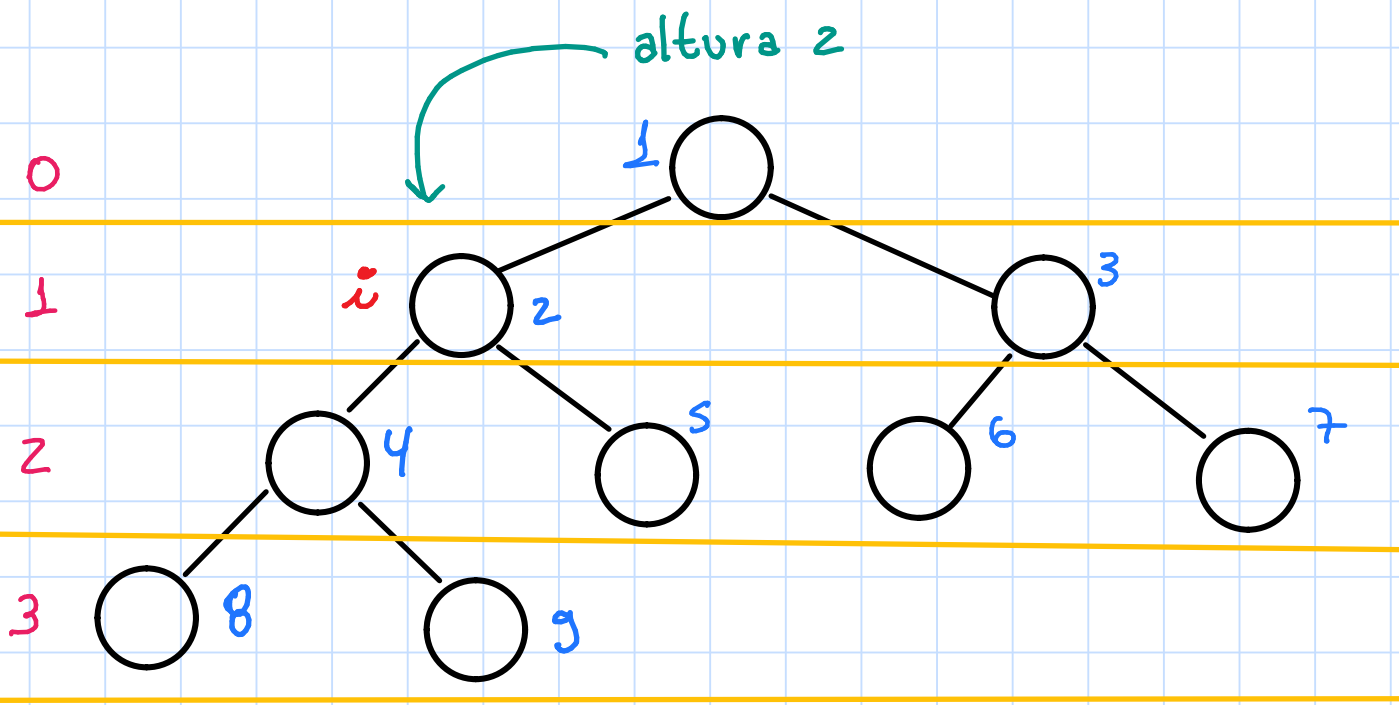
\hookrightarrow Lembre-se que o 1º nível tem rótulo zero

Altura de um nó i é o maior nº de arestas do nó i até uma folha na subárvore enraizada em i



Altura da árvore é a altura do nó raiz

Árvore de uma heap tem $\lfloor \lg n \rfloor + 1$ níveis
 \Rightarrow altura de um heap é $\lfloor \lg n \rfloor = O(\lg n)$



Funções

Pai (i)

Devolva $\lfloor i/2 \rfloor$

Filho - Esquerda (i)

Devolva $2i$

Filho - Direita (i)

Devolva $2i+1$

Heap Máximo e Mínimo

Há dois tipos de Heap

- máximo
- mínimo

← Se falarmos simplesmente Heap, vamos assumir que é Heap Máximo.

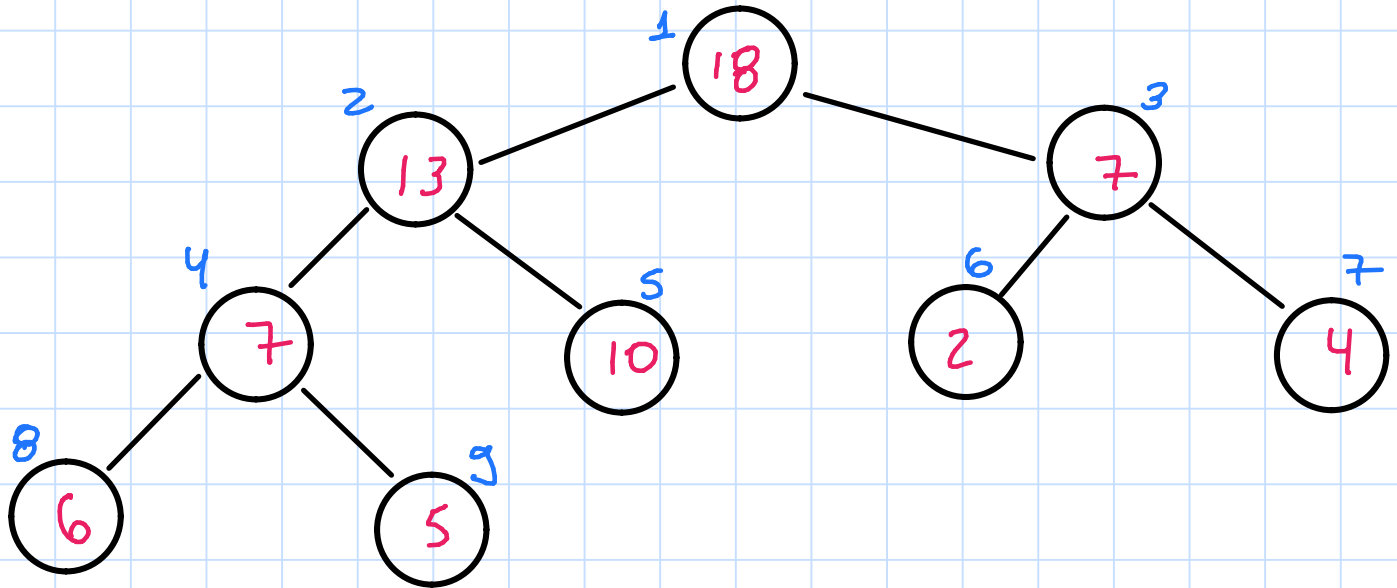
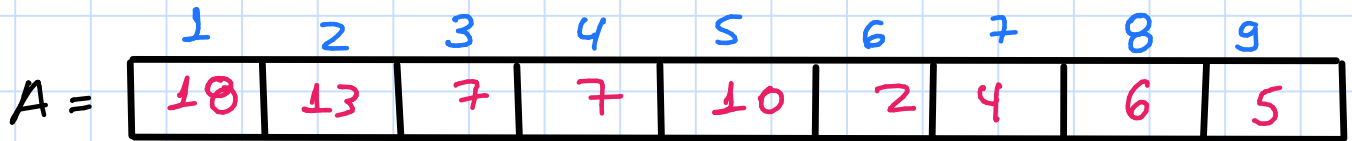
Em um heap mínimo, cada nó é menor que seus filhos, i.e.,

$$A[\text{pai}(i)] \leq A[i] \quad \forall 2 \leq i \leq n$$

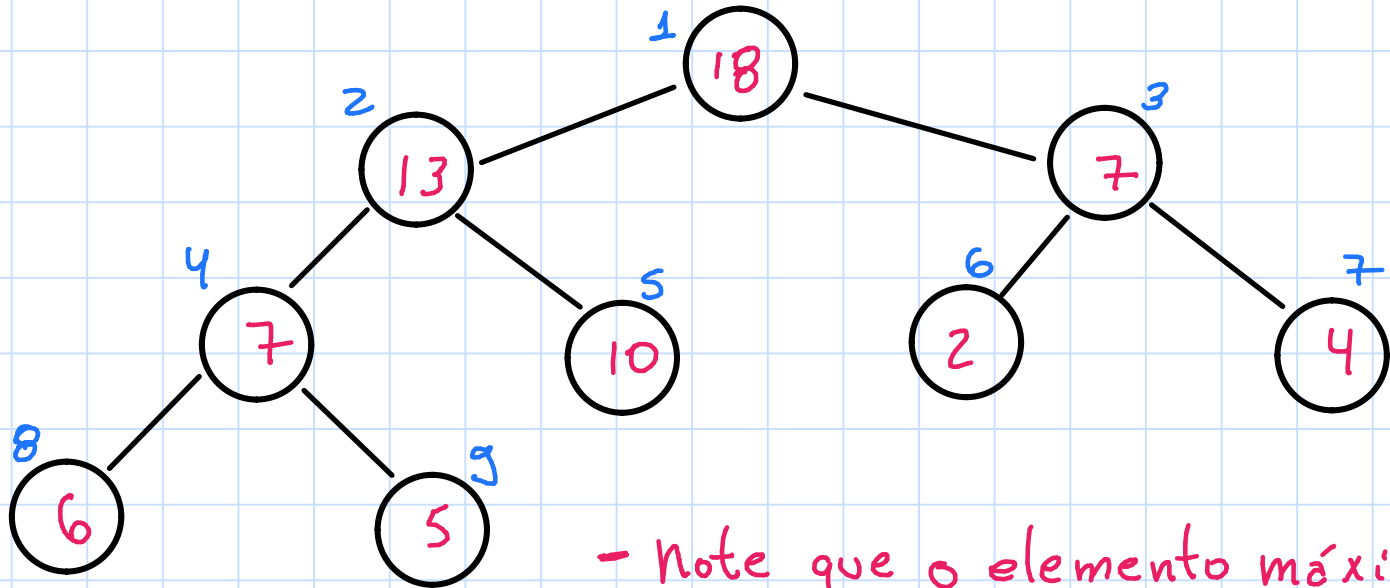
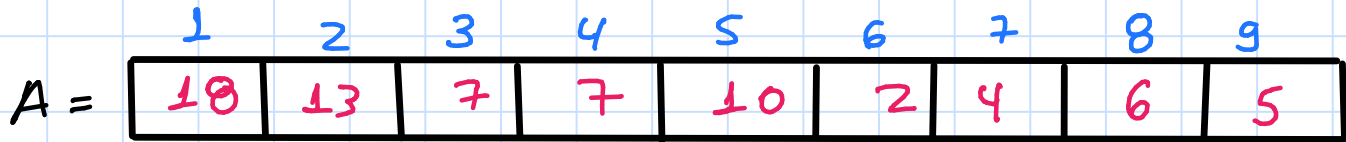
Em um heap máximo, cada nó é maior que seus filhos, i.e.,

$$A[\text{pai}(i)] \geq A[i] \quad \forall 2 \leq i \leq n$$

Exemplo de Heap (Máximo)

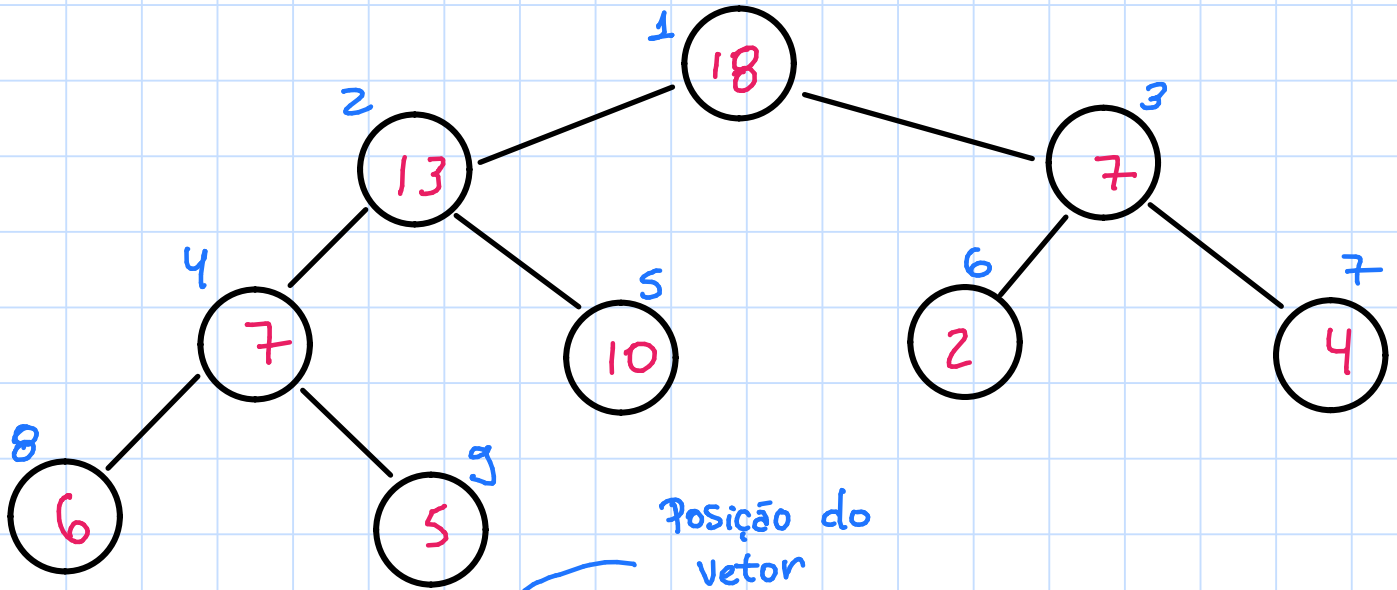
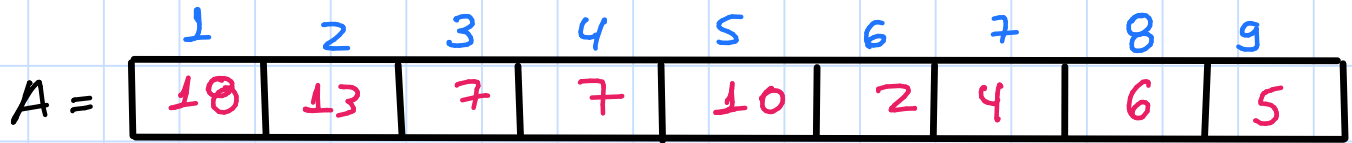


Exemplo de Heap (Máximo)



- Note que o elemento máximo está na raiz
- Cada subárvore tbm é um heap

Operações Heap: Mudança de Valor

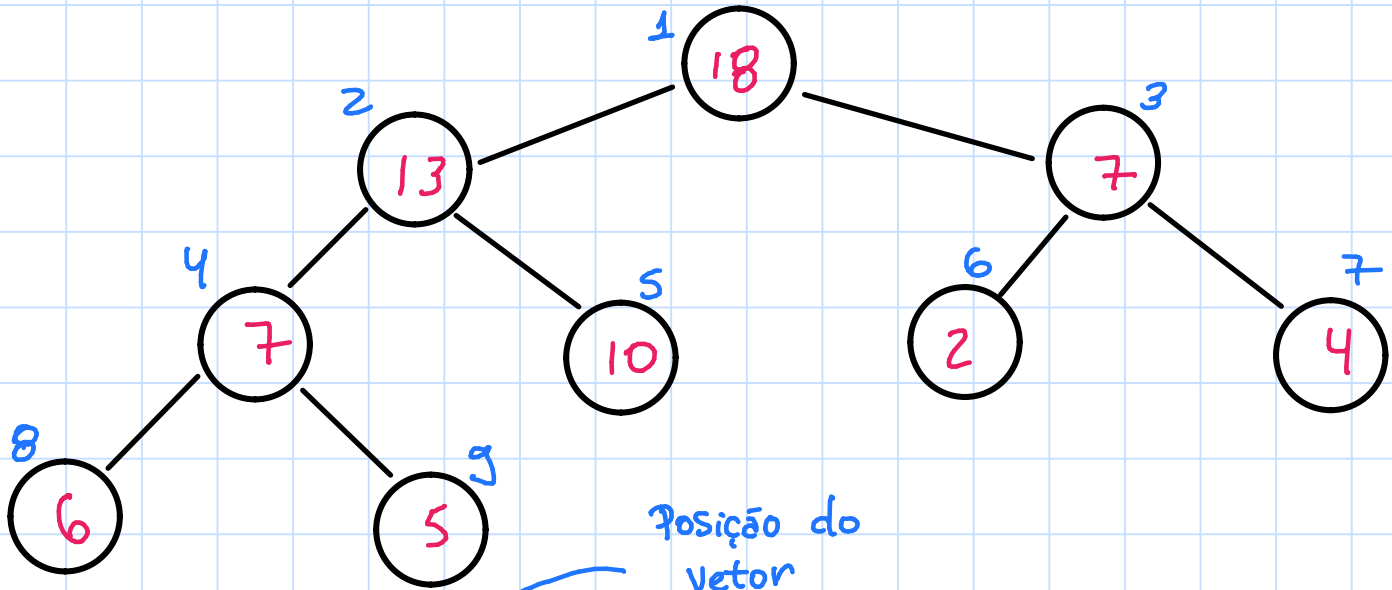
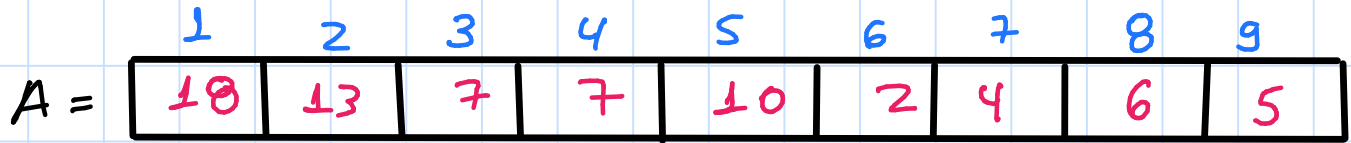


Posição do
vetor

Altera Heap(A, 1, 3)

Novo Valor

Operações Heap: Mudança de Valor



Altera Heap(A, 9, 15)

Posição do vetor

Novo Valor

Corrige HeapDescendo (A, i)

Assumindo que as subárvores do filho da esquerda e do filho da direita são heaps, CorrigeHeapDescendo transforma a subárvore enraizada em i em uma heap

Corrige Heap Descendo (A, n, i)

- 1 $c \leftarrow \text{Filho-Esquerda}(i)$
- 2 $d \leftarrow \text{Filho-Direita}(i)$
- 3 $\text{maior} \leftarrow i$
- 4 se $e \leq n$ e $A[c] > \text{maior}$
- 5 $\text{maior} \leftarrow c$
- 6 se $d \leq n$ e $A[d] > \text{maior}$
- 7 $\text{maior} \leftarrow d$
- 8 se $\text{maior} \neq i$
- 9 $A[i] \leftrightarrow A[\text{maior}]$
- 10 $\text{CorrigeHeapDescendo}(A, n, \text{maior})$

Lema Dado um vetor A e um índice i tal que as subárvores enraizadas no filho da esquerda e direita são heaps, o algoritmo

$\text{CorrigeHeapDescendo}(A, i)$

transforma a subárvore enraizada em i em um heap.

Ideia da demonstração: indução no altura h do nó i

- Se $h=0$, então é folha e o algoritmo está correto

- se $h > 0$, suponha que o algoritmo funciona para árvores com alturas menores que h . Antes da

linha 8 $A[\text{maior}] \geq A[i], A[2i], A[2i+1]$. Após

a linha 9, $A[i] \geq A[2i], A[2i+1]$. Pela H.I. $\text{CorrigeHeap-$

Descendo transforma a subárvore com raiz maior em

Heap

□

Complexidade de CorrigirHeapDescendo

Corrige Heap Descendo (A, n, i)

- 1 $e \leftarrow \text{Filho-Esquerda}(i)$
- 2 $d \leftarrow \text{Filho-Direita}(i)$
- 3 $\text{maior} \leftarrow i$
- 4 se $e \leq n$ e $A[e] > \text{maior}$
- 5 $\text{maior} \leftarrow e$
- 6 se $d \leq n$ e $A[d] > \text{maior}$
- 7 $\text{maior} \leftarrow d$
- 8 se $\text{maior} \neq i$
- 9 $A[i] \leftrightarrow A[\text{maior}]$
- 10 CorrigeHeapDescendo(A, n, maior)

Complexidade de CorrigirHeapDescendo

Corrige Heap Descendo (A, n, i)

```
1  e ← Filho-ESquerda(i)
2  d ← Filho-Direita(i)
3  maior ← i
4  se e ≤ n e A[e] > maior
5     maior ← e
6  se d ≤ n e A[d] > maior
7     maior ← d
8  se maior ≠ i
9     A[i] ↔ A[maior]
10 CorrigirHeapDescendo(A, n, maior)
```

$$\begin{aligned} T(h) &= T(h-1) + \Theta(1) \\ &= O(h) \\ &= O(\lg n) \end{aligned}$$

$\Theta(1)$

$$T(h-1)$$

$$T(n) = T(n-1) + 1$$

$$= T(n-2) + 1 + 1 = T(n-2) + 2$$

$$= T(n-3) + 3$$

\vdots

$$= T(n-i) + i$$

\vdots

$$= T(1) + n - 1$$

$$= 1 + n - 1$$

$$= n$$

$$T(n) = O(n)$$

$$T(0) = T(0-1) + 1$$

Corrige Heap Subindo (A, i)

1 Se $i \geq 2$ e $A[i] > A[\text{pai}(i)]$

2 $A[i] \leftrightarrow A[p]$

3 CorrigelHeap Subindo (A, pai(i))

Complexidade de Corrigir Heap Subindo (A, i)

Corrigir Heap Subindo (A, i)

- 1 Se $i \geq 2$ e $A[i] > A[\text{pai}(i)]$
- 2 $A[i] \leftrightarrow A[p]$
- 3 Corrigir Heap Subindo ($A, \text{pai}(i)$)

Complexidade de Corrigir Heap Subindo (A, i)

Corrigir Heap Subindo (A, i)

- 1 Se $i \geq 2$ e $A[i] > A[\text{pai}(i)]$
 - 2 $A[i] \leftrightarrow A[p]$
 - 3 Corrigir Heap Subindo (A, $\text{pai}(i)$)
- $\left. \begin{array}{l} \text{1} \\ \text{2} \end{array} \right\} \Theta(1)$
 $\left. \begin{array}{l} \text{3} \end{array} \right\} T(h-1)$

Seja h a altura do nó i

$$T(h) = T(h-1) + \Theta(1)$$

$$= O(\lg n)$$

Alterar Heap (A, n, i, k)

$t \leftarrow A[i]$

$A[i] \leftarrow k$

Se $A[i] > t$

CorrigirHeapSubindo (A, i)

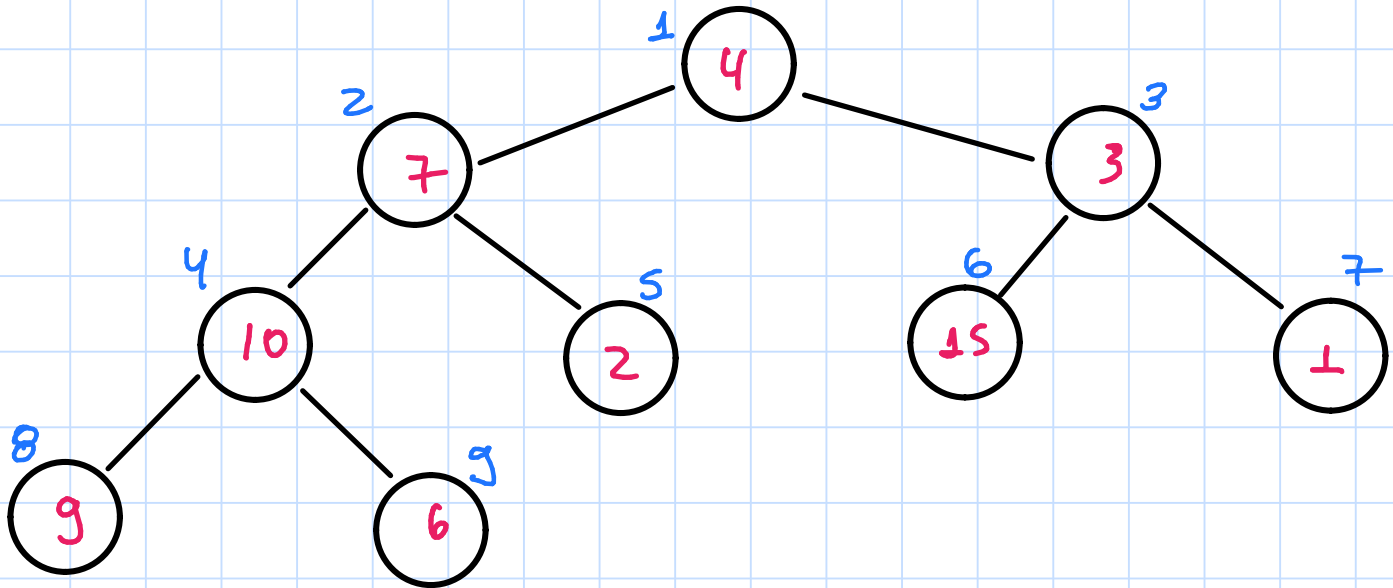
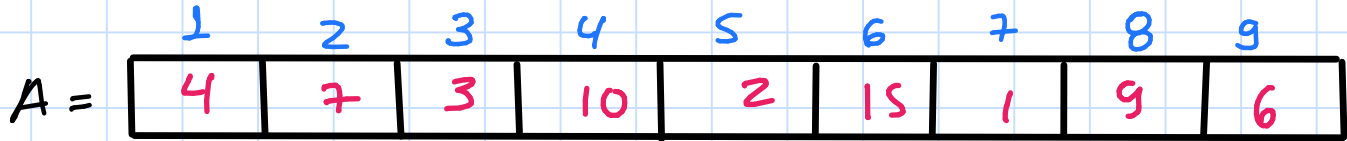
Se $A[i] < t$

CorrigirHeapDescendo (A, n, i)

- Correção é uma consequência direta da correção de CorrigirHeapSubindo e CorrigirHeapDescendo

- Tempo de Execução: $O(\lg n)$

Construção da Heap



ConstroiHeap(A, n) # elementos no vetor

para $i = \lfloor n/2 \rfloor$ decrescendo até 1

CorrigeHeapDescendo(A, n, i)

Por que começamos em $\lfloor n/2 \rfloor$?

$$\underline{2i \leq n} \iff \underline{i \leq \frac{n}{2}}$$

↑
Filho esquerda
existe

→ Como i é inteiro $i \leq \lfloor \frac{n}{2} \rfloor$

Complexidade de ConstróiHeap

ConstróiHeap(A, n)

1 para $i = \lfloor n/2 \rfloor$ decrescendo até 1 $\Theta(n)$

2 CorrigeHeapDescendo(A, n, i) $\Theta(n) \cdot \Theta(\lg n)$

Uma análise rápida nos mostra que

$$T(n) = O(n \lg n)$$

Uma análise mais minuciosa nos mostra que é

$$T(n) = \Theta(n)$$

\Rightarrow claramente é $\Omega(n)$ (linha 1)

Lema Em uma heap com n elementos, existem no máximo

$\lceil n/2^{h+1} \rceil$
nós de altura h .

Demonstração

Se $h=0$, então existem $\lceil n/2 \rceil = \lceil n/2^{0+1} \rceil = \lceil n/2^{h+1} \rceil$ elementos que não são folhas

$$2i = n$$

$$i = \frac{n}{2} = \lfloor \frac{n}{2} \rfloor$$

$$n = \lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil$$

$$2i+1 = n$$

$$i = \frac{n-1}{2} = \lfloor \frac{n}{2} \rfloor$$

Se $h > 0$, então por hipótese de indução existem no máximo

$$\lceil n/2^h \rceil$$

Se $h > 0$, então por hipótese de indução existem no máximo

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

nós de altura $h-1$. Note que o número de nós com altura h é a metade do número de nós com altura $h-1$. Portanto, o número máximo de nós com altura h é

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

□

Assim, temos que o tempo de execução de constrói Heap

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

O leço da linha 1 este iterando sobre os vértices que não são folha (altura entre 1 e $\lfloor \lg n \rfloor$)

$$\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil C \cdot h = \sum_{h=1}^{\lfloor \lg n \rfloor} \left\lceil \frac{n/2^h}{2} \right\rceil C \cdot h$$

$$\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h} \cdot C \cdot h$$

$$\left\lceil \frac{n/a}{2} \right\rceil \leq \frac{n}{a}$$

caso n/a par

$$\frac{n/a}{2} \leq \frac{n}{a}$$

caso n/a ímpar

$$\frac{n/a + 1}{2} \leq \frac{n}{a}$$

$$n/a + 1 \leq 2n/a$$

$$1 \leq n/a$$

$$a \leq n$$

$$= Cn \sum_{h=1}^{\lfloor \lg n \rfloor} h/2^h$$

$$\leq 4Cn = O(n) \leq 4$$

Próx. Slide

Vamos provar que $\sum_{i=0}^m \frac{1}{2^i} \leq 4$, para qualquer $m \geq 0$

$$\Rightarrow \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{1}{2^h} \leq 4$$

Vamos provar que $\sum_{i=0}^m \frac{i}{2^i} \leq 4$, para qualquer $m \geq 0$

$$\text{Seja } S = \sum_{i=0}^m \frac{i}{2^i} = \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \dots + \frac{m}{2^m}$$

$$\text{Portanto } \frac{S}{2} = \frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \dots + \frac{m}{2^{m+1}}$$

Note que

$$\begin{aligned} \frac{S}{2} &= S - \frac{S}{2} = \left(\frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \dots + \frac{m}{2^m} \right) - \left(\frac{1}{2^2} + \frac{2}{2^3} + \dots + \frac{m}{2^{m+1}} \right) \\ &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^m} - \frac{m}{2^{m+1}} \end{aligned}$$

$$\frac{S}{2} = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^m} - \frac{m}{2^{m+1}}$$

$$S = 2 \frac{S}{2} = 2 \left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^m} \right) - \frac{2m}{2^{m+1}}$$

$$\leq 2 \left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^m} \right)$$

$$= 2 \sum_{i=1}^m \left(\frac{1}{2} \right)^i = 2 \left[\frac{\left(\frac{1}{2} \right)^{m+1} - 1}{\frac{1}{2} - 1} \right]$$

$$= 2 \left[\frac{\left(\frac{1}{2} \right)^{m+1} - 1}{-\frac{1}{2}} \right] = -4 \left[\left(\frac{1}{2} \right)^{m+1} - 1 \right]$$

$$= 4 - 4 \left(\frac{1}{2} \right)^{m+1} \leq 4$$

Correção

ConstroiHeap(A, n)

- 1 para $i = \lfloor n/2 \rfloor$ decrescendo até 1
- 2 CorrigeHeapDescendo(A, n, i)

Tco ConstroiHeap(A, n) transforma um vetor de números A em um Heap.

$P(t) =$ "Antes da t -ésima iteração começar, vale que

- $i = \lfloor n/2 \rfloor + 1 - t$, e
- a árvore enraizada em $A[i]$, para $i+1 \leq j \leq n$ é uma heap

Heap - Sort

Selection-Sort (A, n)

Para $i \leftarrow n$ decrescendo até 2
 $\max \leftarrow \text{Maximum}(A, i)$
 $A[i] \leftrightarrow A[\max]$

Maximum(A, i)

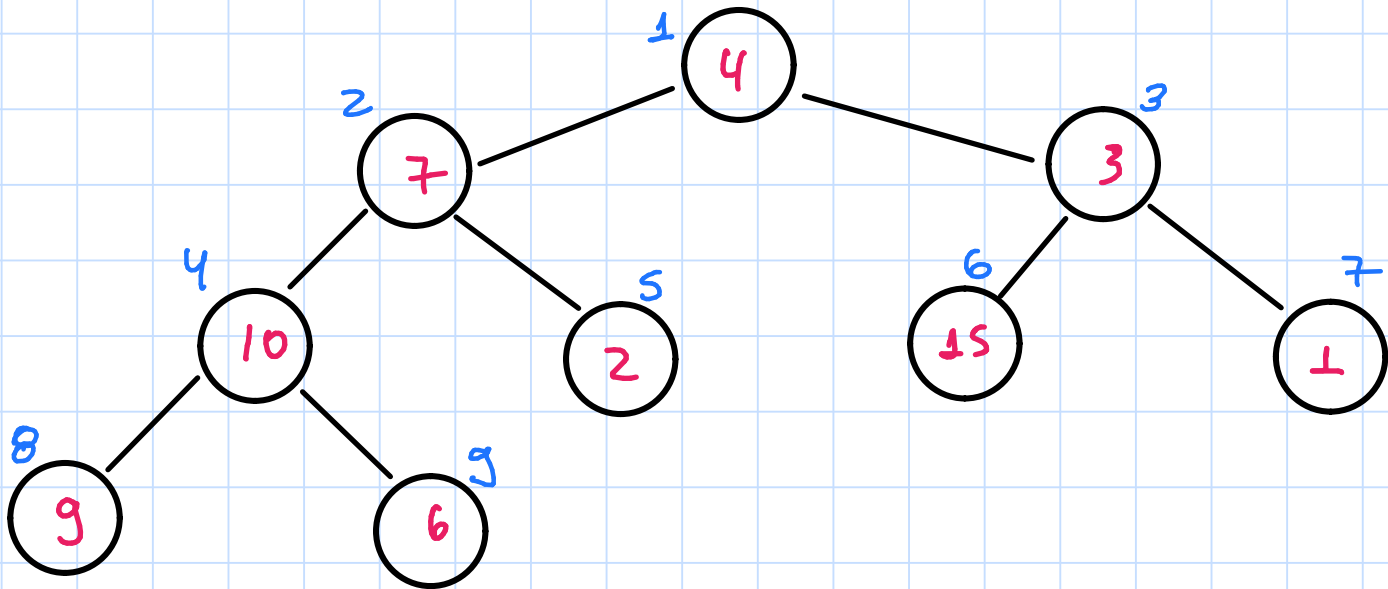
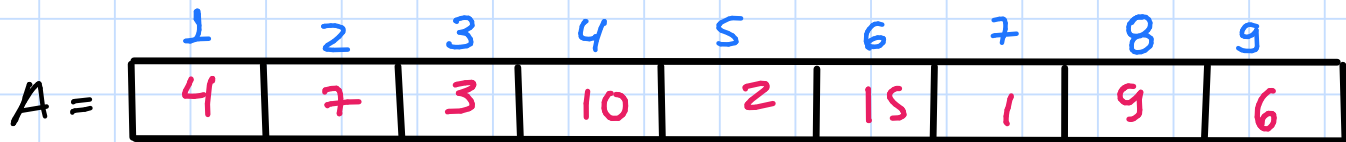
$\max \leftarrow i$

para $j \leftarrow i-1$ decrescendo até 1
se $A[j] > A[\max]$

$\max \leftarrow j$

Devolva \max

Exemplo



HeapSort (A, n)

- 1 ConstroiHeap (A, n)
- 2 Para $i \leftarrow n$ decrescendo até 2
- 3 $A[1] \leftrightarrow A[i]$
- 4 CorrigeHeapDescendo ($A, i-1, 1$)

Tempo de Execução

HeapSort (A, n)

- 1 ConstróiHeap (A, n)
- 2 Para $i \leftarrow n$ decrescendo até 2
- 3 $A[1] \leftrightarrow A[i]$
- 4 CorrigeHeapDecrescendo ($A, i-1, 1$)

Tempo de Execução

HeapSort (A, n)

- 1 ConstroiHeap (A, n) $O(n)$
- 2 Para $i \leftarrow n$ decrescendo até 2 $O(n)$
- 3 $A[1] \leftrightarrow A[i]$ $O(n)$
- 4 CorrigeHeapDescendo ($A, i-1, 1$) $O(n) \cdot O(\lg n)$

$$T(n) = O(n \lg n)$$

Correção

HeapSort (A, n)

- 1 ConstroiHeap(A, n)
- 2 Para $i \leftarrow n$ decrescendo até 2
- 3 $A[1] \leftrightarrow A[i]$
- 4 CorrigeHeapDescendo($A, i-1, 1$)

Teo O algoritmo HeapSort(A, n) ordena um vetor A de n elementos

$P(t) =$ " Antes da t -ésima iteração começar,

- $i = n + 1 - t$
- $A[1.. \underbrace{n+1-t}_i]$ é um heap
- $A[\underbrace{n+2-t}_{i+1}.. n]$ está ordenado
- $A[1.. \underbrace{n+1-t}_i] \leq \min(A[\underbrace{n+2-t}_{i+1}.. n])$
- Contém uma permutação dos elementos iniciais "

Base ($t=1$)

- $i = n + 1 - t = n$
- $A[1.. \underbrace{n+1-t}_i] = A[1.. n]$ é um heap
- $A[\underbrace{n+2-t}_{i+1}.. n] = A[n+1.. n] = \emptyset$ está ordenado
- $A[1.. \underbrace{n+1-t}_i] = A[1.. n] \leq \min(A[\underbrace{i+1}_{i+1}.. n] = A[n+1.. n])$

Passo: $P(t)$ Vale, iteração t ocorre, vamos provar $P(t+1)$
 $P(t) =$ Antes da t -ésima iteração começar,

- ① $i = n + 1 - t$
- ② $A[1.. \underbrace{n+1-t}_i]$ é um heap
- ③ $A[\underbrace{n+2-t}_{i+1}.. n]$ está ordenado
- ④ $A[1.. \underbrace{n+1-t}_i] \leq \min(A[\underbrace{n+2-t}_{i+1}.. n])$

Seja $x = A[i]$. Por ②, $x \geq A[1.. n+1-t]$ e, por ④, $x \leq \min(A[n+2-t.. n])$. A linha 3 executa e coloca x em $A[i] = A[n+1-t]$. Portanto $A[n+1-t.. n]$ está ordenado e $A[1.. n-t] \leq \min(A[n+1-t.. n])$ (por 4 e 2).

Após a linha 4 temos que $A[1.. i-1] = A[1.. n-t]$ é uma heap. O laço finaliza e i é decrementado, passando a valer $i = n - t$.

□

Teo O algoritmo $\text{HeapSort}(A, n)$ ordena um vetor A de n elementos

Demonstração

- Pela correção de ConstructHeap , após a exe. da linha 1, A é um Heap.
- Pela invariante $P(k)$, o laço termina após n iterações
- Por $P(n)$, temos que

$$A[2..n] \text{ está ordenado e} \\ A[1..1] = A[1] \leq \min(A[2..n])$$

- Portanto A está ordenado □

Ordenação

Tempo de Execução

Insertion Sort

$$O(n^2)$$

Merge Sort

$$O(n \lg n)$$

Selection Sort

$$O(n^2)$$

Heap Sort

$$O(n \lg n)$$

Quick Sort

$$O(n^2)$$

Qualquer algoritmo baseado em comparação requer $\Omega(n \lg n)$ no pior caso.

Exemplo de algoritmo que não é baseado em Comp.

Counting-Sort (A, n) \triangleright Sabemos que $1 \leq A[i] \leq C$
 $\neq i$

Seja $B[1..C]$ um vetor

Para $i = 1..C$

$$B[i] = 0$$

Para $i = 1$ até n

$$B[A[i]] \leftarrow B[A[i]] + 1$$

$$k \leftarrow 0$$

Para $j \leftarrow 1$ até C

para $i \leftarrow 1$ até $B[j]$

$$A[k+i] \leftarrow j$$

$$k \leftarrow k + B[j]$$

Exemplo de algoritmo que não é baseado em Comp.

Counting-Sort (A, n) \triangleright Sabemos que $1 \leq A[i] \leq C$
 $\neq i$

Seja $B[1..C]$ um vetor

Para $i = 1..C$
 $B[i] = 0$ $\left. \vphantom{\text{Para } i = 1..C} \right\} \Theta(C)$

Para $i = 1$ até n
 $B[A[i]] \leftarrow B[A[i]] + 1$ $\left. \vphantom{\text{Para } i = 1 \text{ até } n} \right\} \Theta(n)$

$k \leftarrow 0$

Para $j \leftarrow 1$ até C $\left. \vphantom{\text{Para } j \leftarrow 1 \text{ até } C} \right\} \Theta(C)$

para $i \leftarrow 1$ até $B[j]$
 $A[k+i] \leftarrow j$
 $k \leftarrow k + B[j]$ $\left. \vphantom{\text{para } i \leftarrow 1 \text{ até } B[j]} \right\} \Theta(n)$

$T(n) = \Theta(n)$

Fila de Prioridades

Uma **Fila de Prioridade** é um tipo abstrato de dados que consiste de uma coleção S de itens com prioridades associadas e permite as operações

- **Maximum(S)** devolve o elemento de maior prioridade
- **Extract-Max(S)** remove o elemento de maior prioridade
- **Increase-key(S, x, p)** altera a prioridade de x para p
- **Insert(S, x, p)** insere um elemento x com prioridade p

Fila de Prioridade Pode ser implementada como

- Vetor
- Heap
- outros

Inserte Heap(A, n, p)

$A[n+1] \leftarrow p$

Corrige Heap Subindo($A, n+1$)

Devolve $A, n+1$

Remove Heap(A, n, i)

$A[i] \leftarrow A[n]$

Corrige Heap Descendo($A, n-1, i$)

Devolve $A, n-1$

