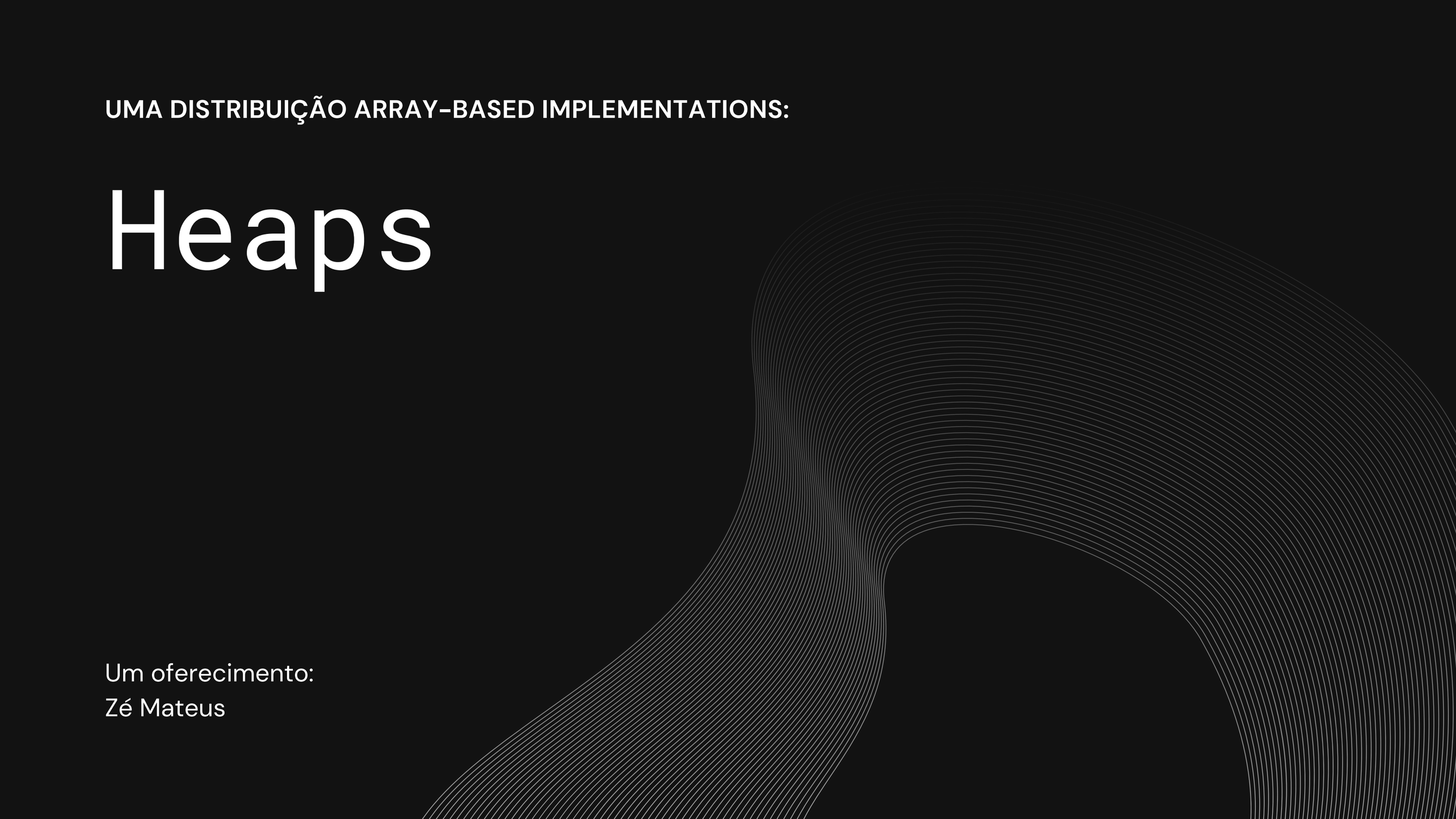
UMA DISTRIBUIÇÃO ARRAY-BASED IMPLEMENTATIONS:

Heaps



Um oferecimento:
Zé Mateus

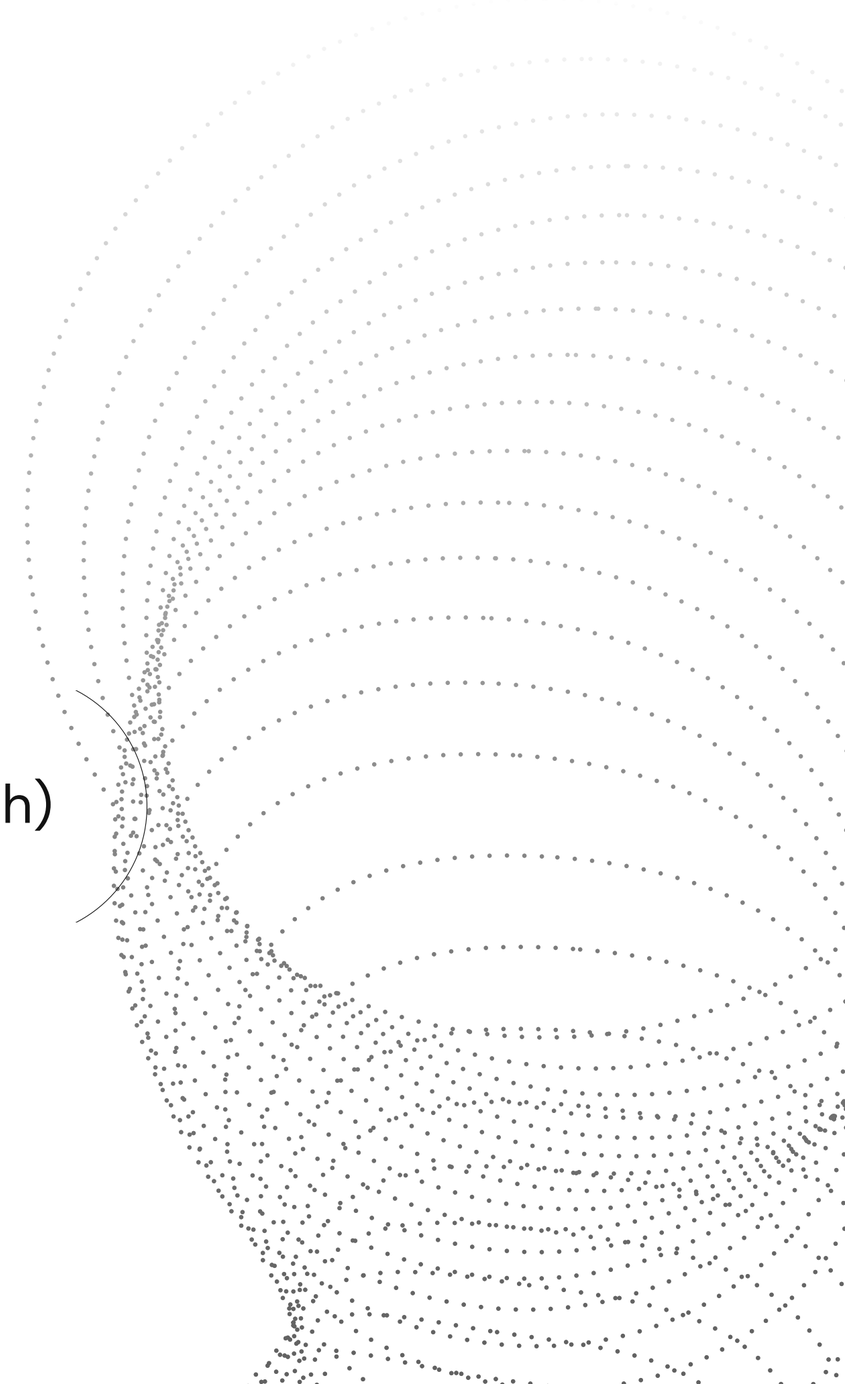
Priority queue

ABSTRACT DATA TYPE (ADT)

E find max(heap h)

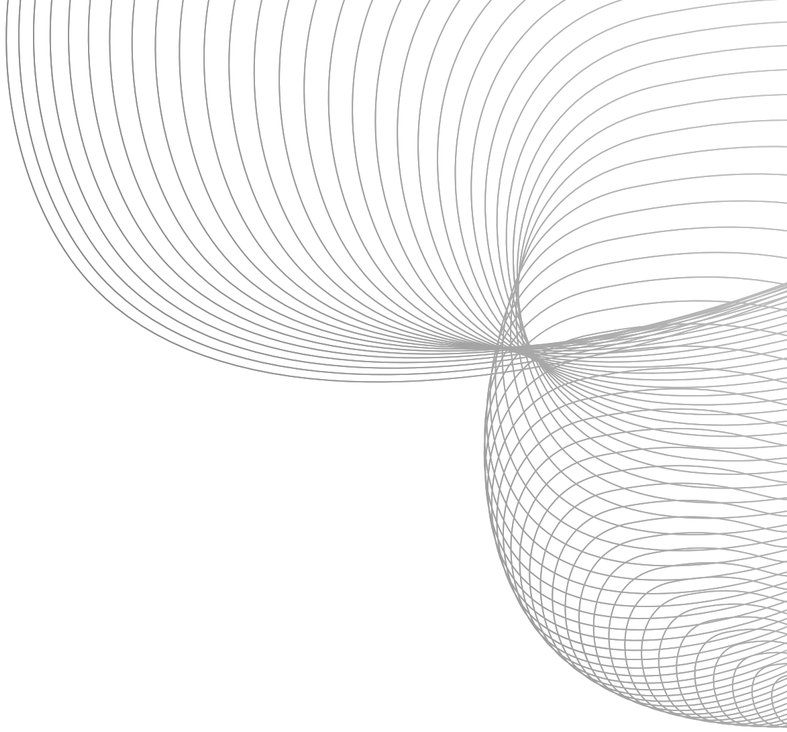
E remove max(heap h)

void add(heap h, Key K, Element E)



Binary Tree

2 MAIN CONDITIONS



SHAPE PROPERTY

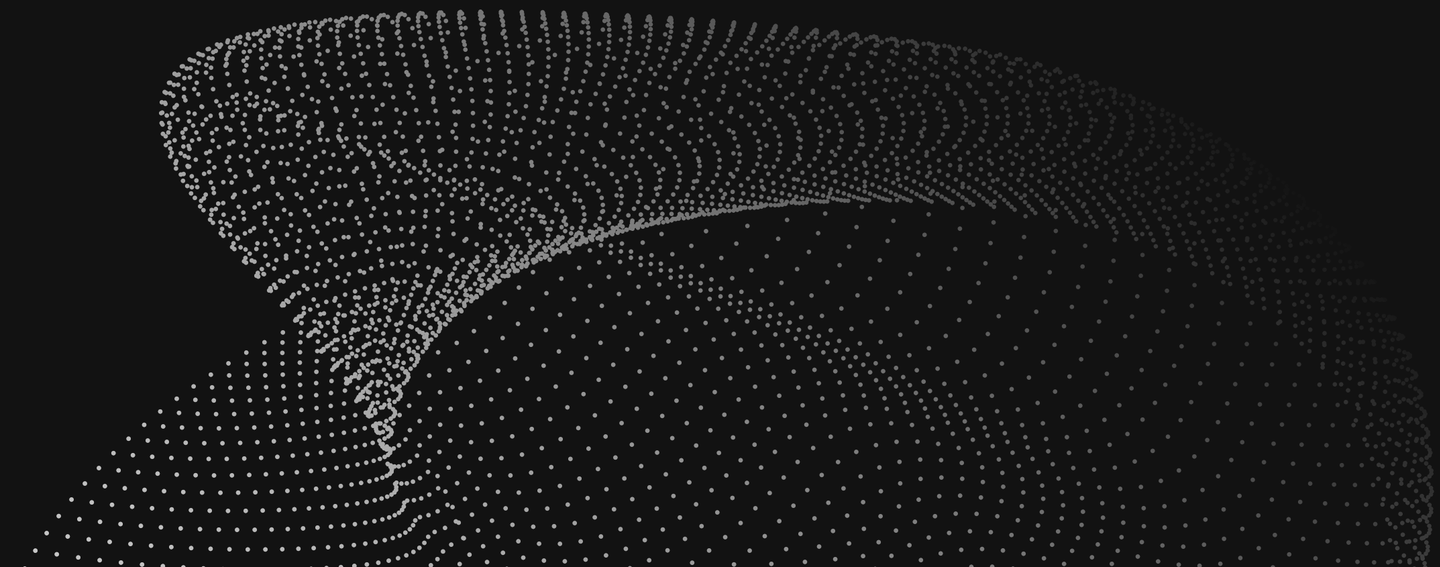
Complete Binary Tree

PARENTAL DOMINANCE

Each node key is ≥/≤ than the childrens key, depending if it is a max-heap or a min-heap

2 tipos de heap

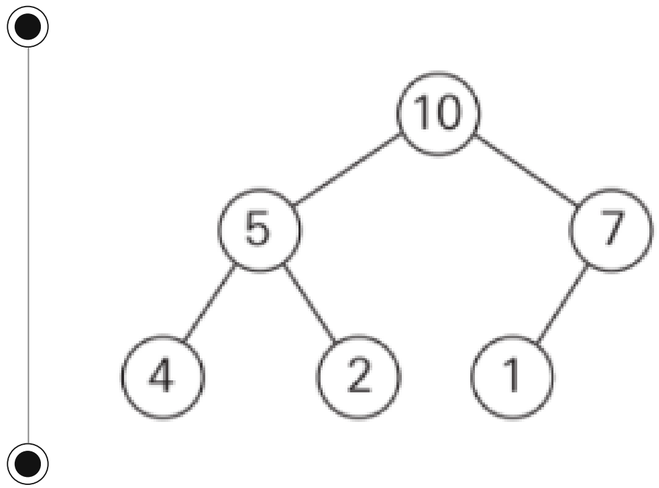
MAX-HEAP E MIN-HEAP

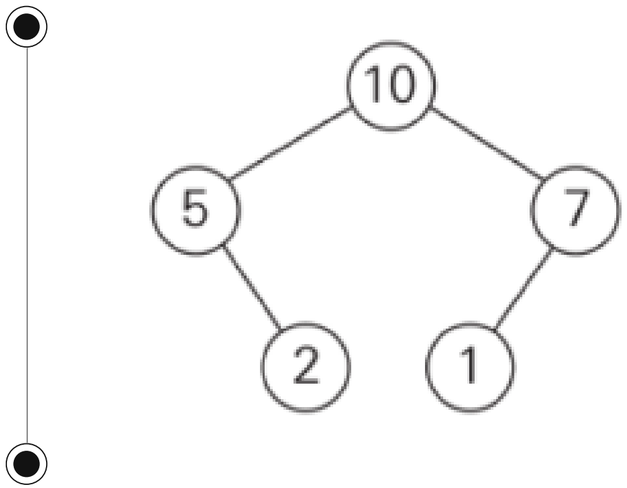


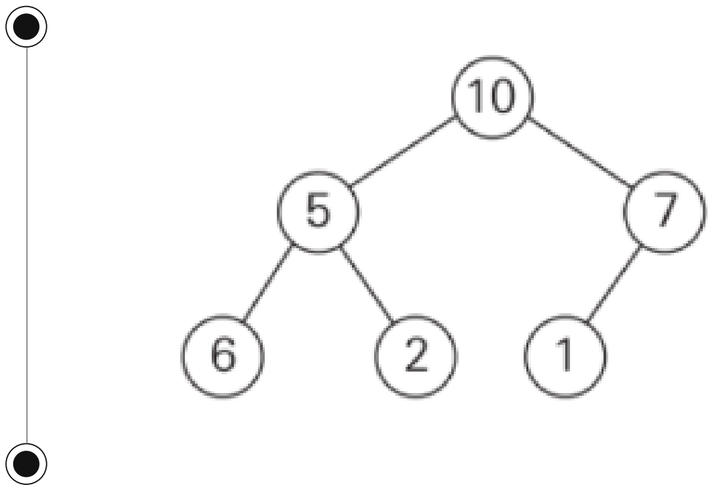
Max-heap: ≥

Min-heap: ≤

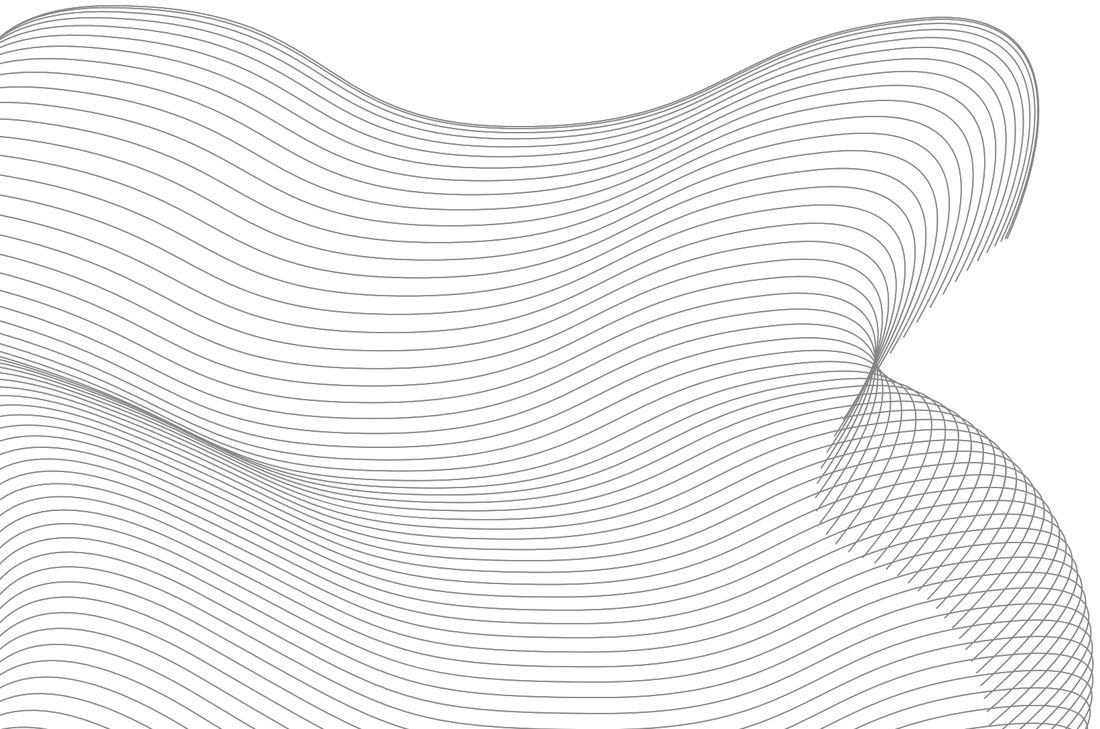
Agora vocês:



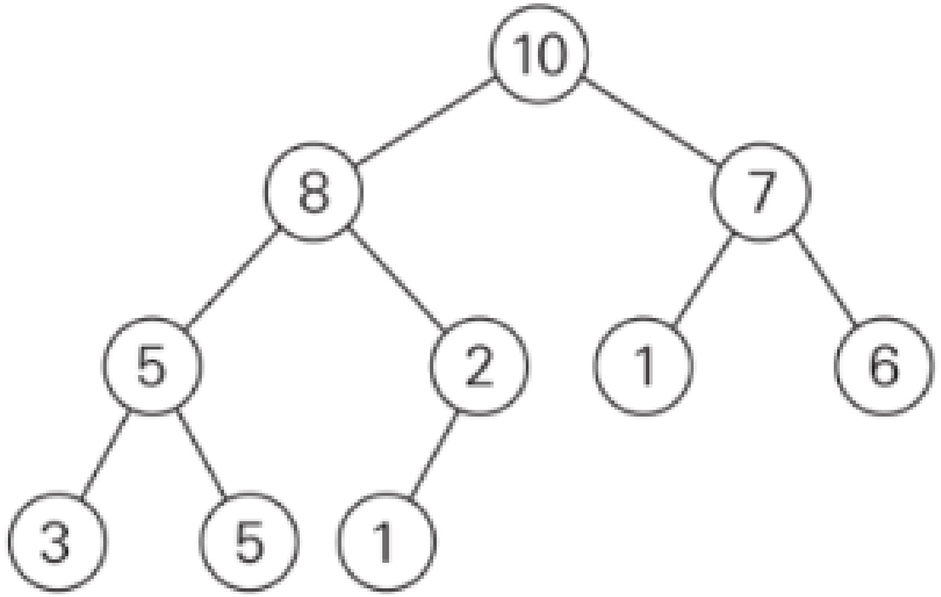


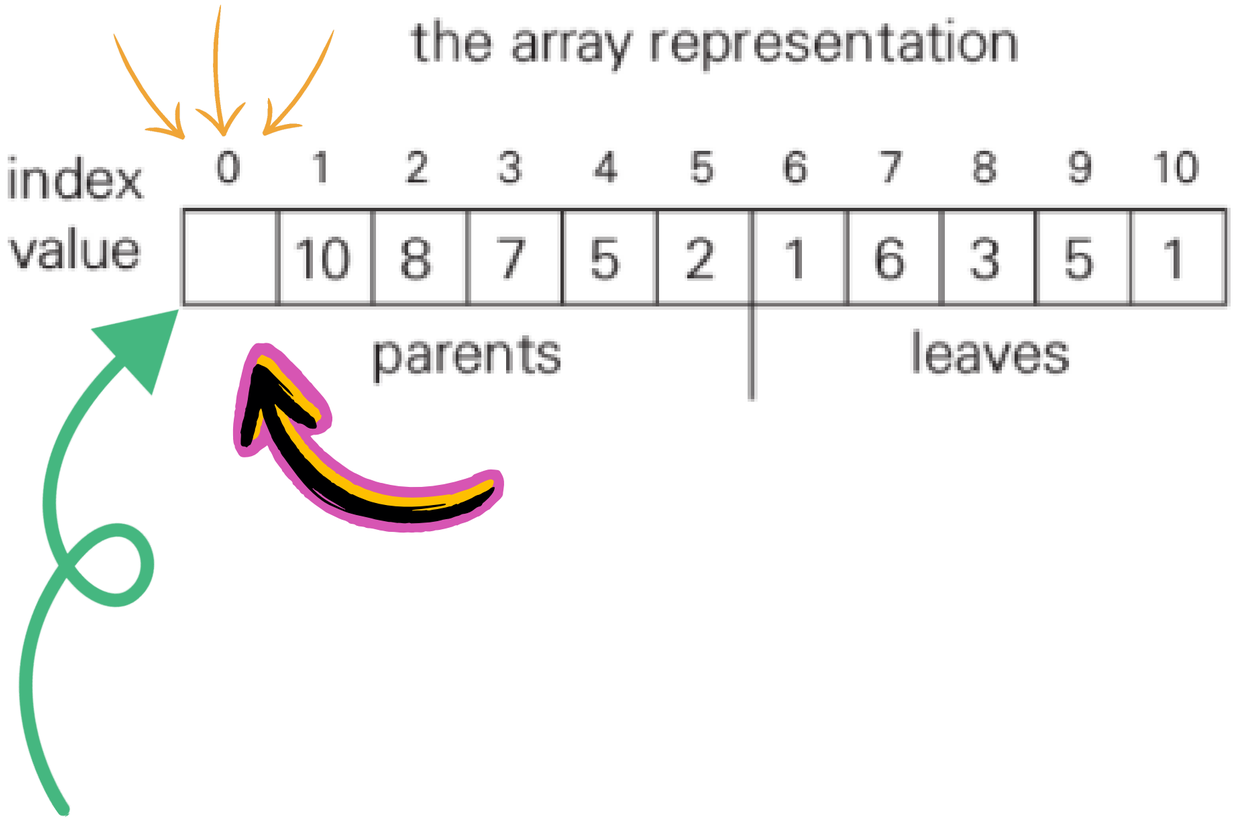


Checklist



- Left-most tree: is a heap
- Middle tree: is not a heap (shape property not met)
- Right-most tree: is not a heap (parental dominance not met)


10
8
7
5
2
1
6
3
5
1


the array representation
index
value
0 1 2 3 4 5 6 7 8 9 10
10 8 7 5 2 1 6 3 5 1
parents
leaves

Propriedades interessantíssimas

INTERNAL NODES

first floor(n/2) positions

LEAVES

last ceiling(n/2) positions

Root position: index 1

Children of 1 ≤ i ≤ floor(n/2): 2i and 2i + 1 (if value ≤ n)

Parent of 2 ≤ i ≤ n: floor(i/2)

Criando Heaps

Top-Down ou Bottom-up

Top-Down

Elements not known beforehand

Elements inserted one-by-one

Heapify after each insertion

Worst temporal efficiency of each insertion in O(log n)

Worst temporal efficiency of heap creation in O(n log n)

Criando Heaps

Top-Down ou Bottom-up

Bottom-up

Elements known beforehand

All elements inserted at once

Heapify from the last to the first internal node

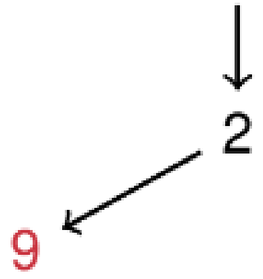
Worst temporal efficiency of heap creation in O(n)

Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10

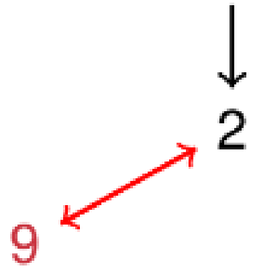
Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10



Heaps: top-down creation (tree view)

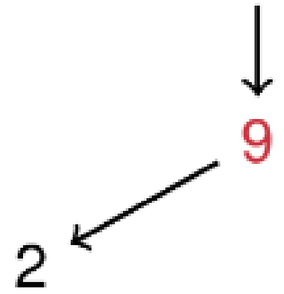
Inserting: 2, 9, 7, 6, 5, 8, 10



Heapify process

Heaps: top-down creation (tree view)

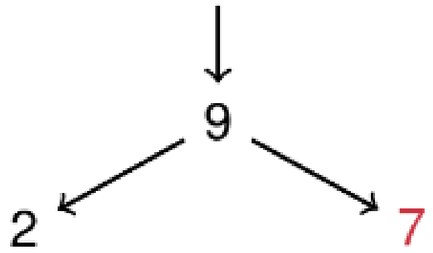
Inserting: 2, 9, 7, 6, 5, 8, 10



Heapify process

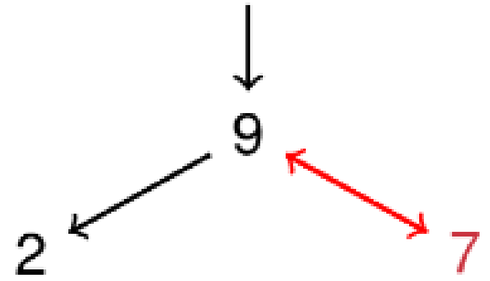
Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10



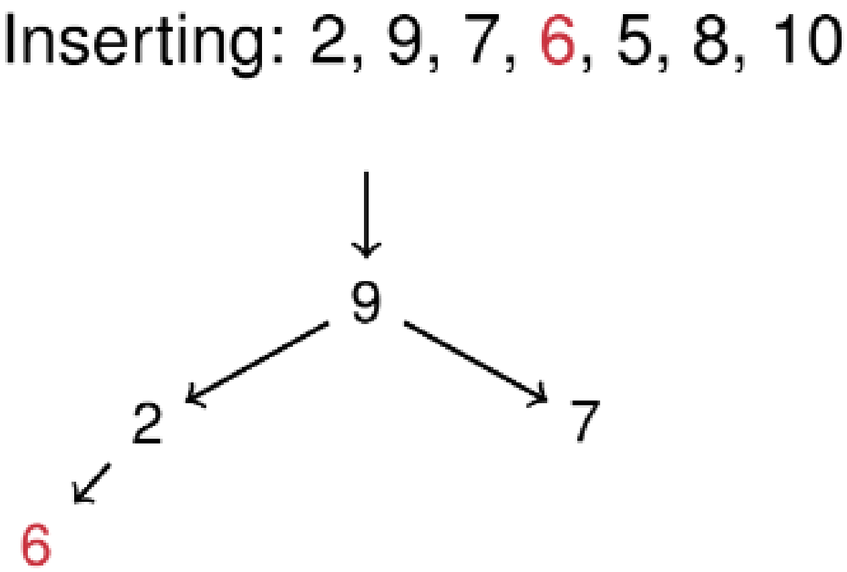
Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10



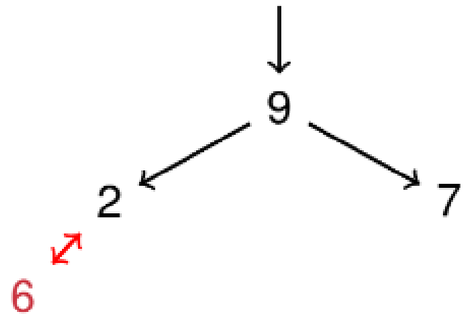
Heapify process

Heaps: top-down creation (tree view)



Heaps: top-down creation (tree view)

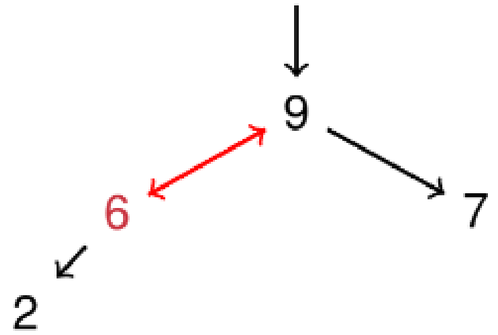
Inserting: 2, 9, 7, 6, 5, 8, 10



Heapify process

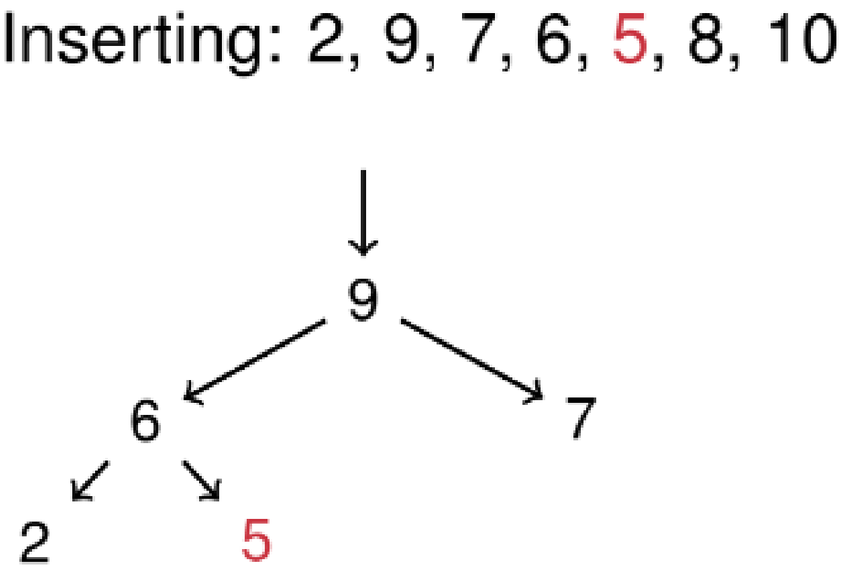
Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10



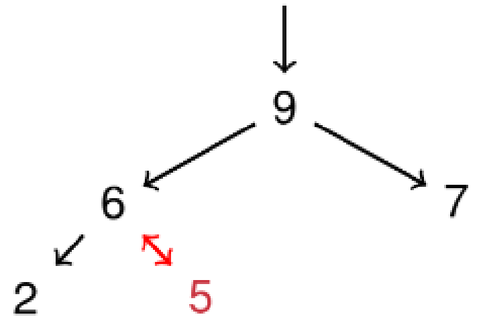
Heapify process

Heaps: top-down creation (tree view)



Heaps: top-down creation (tree view)

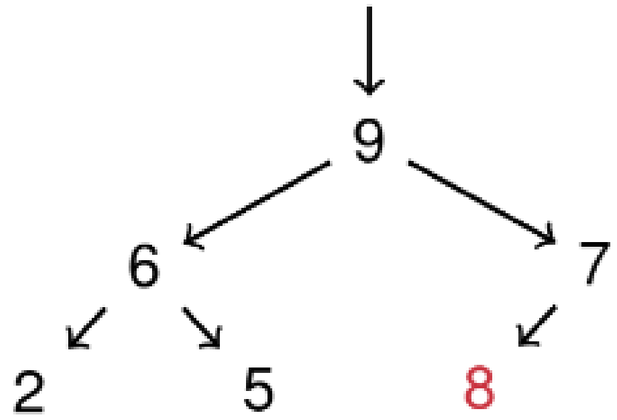
Inserting: 2, 9, 7, 6, 5, 8, 10



Heapify process

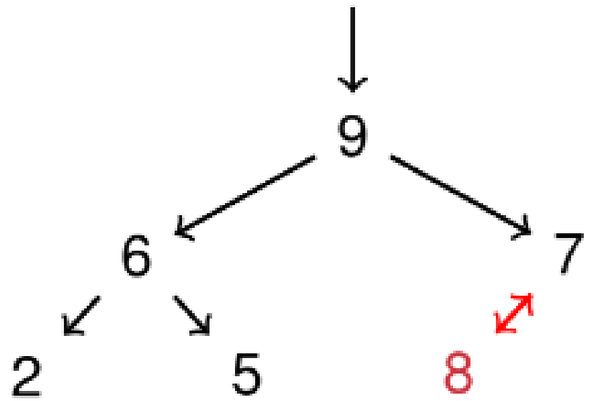
Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10



Heaps: top-down creation (tree view)

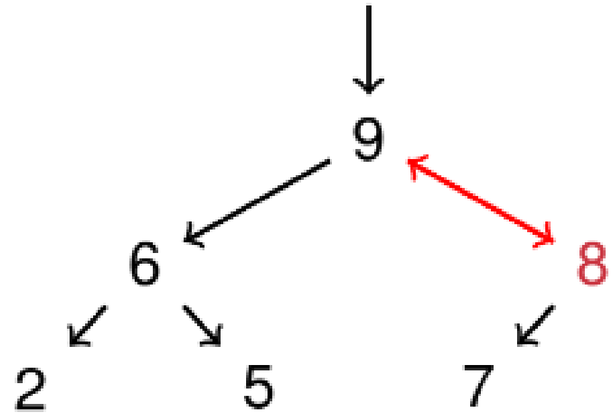
Inserting: 2, 9, 7, 6, 5, 8, 10



Heapify process

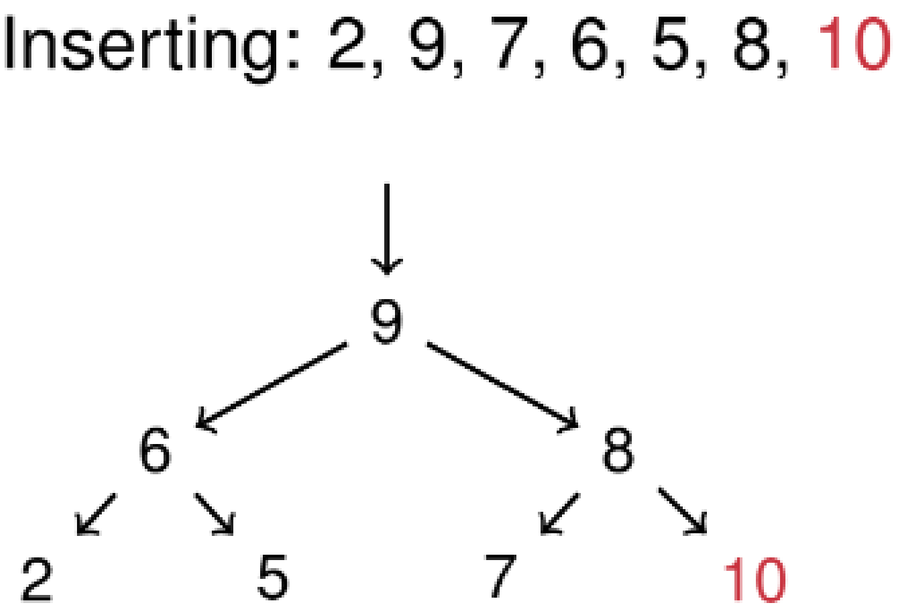
Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10



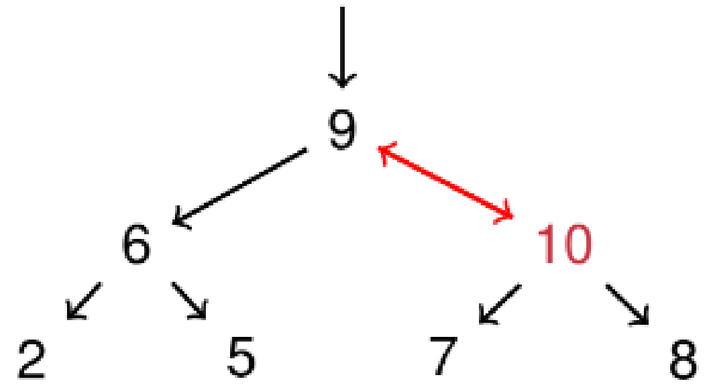
Heapify process

Heaps: top-down creation (tree view)



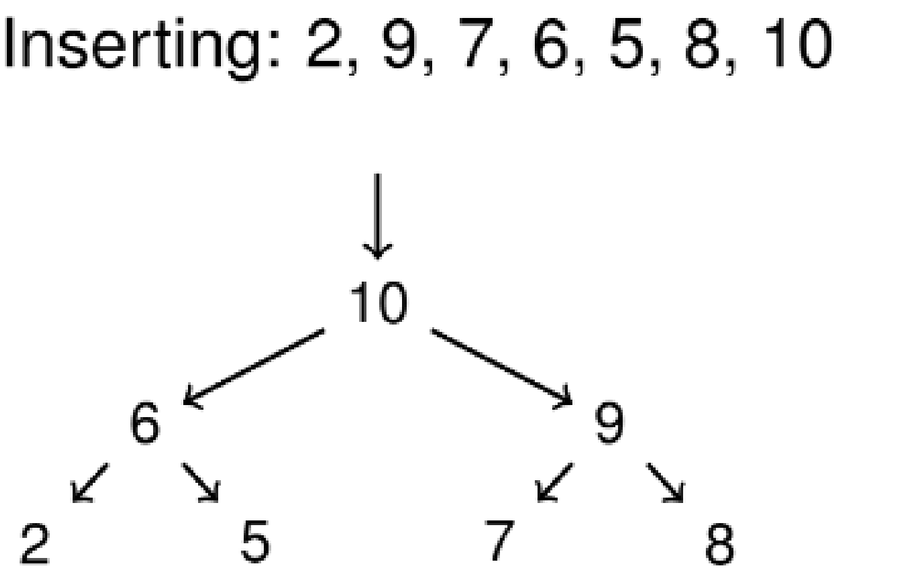
Heaps: top-down creation (tree view)

Inserting: 2, 9, 7, 6, 5, 8, 10



Heapify process

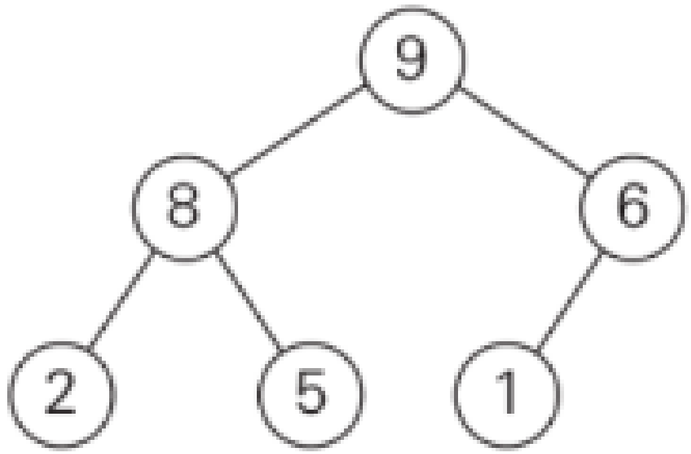
Heaps: top-down creation (tree view)

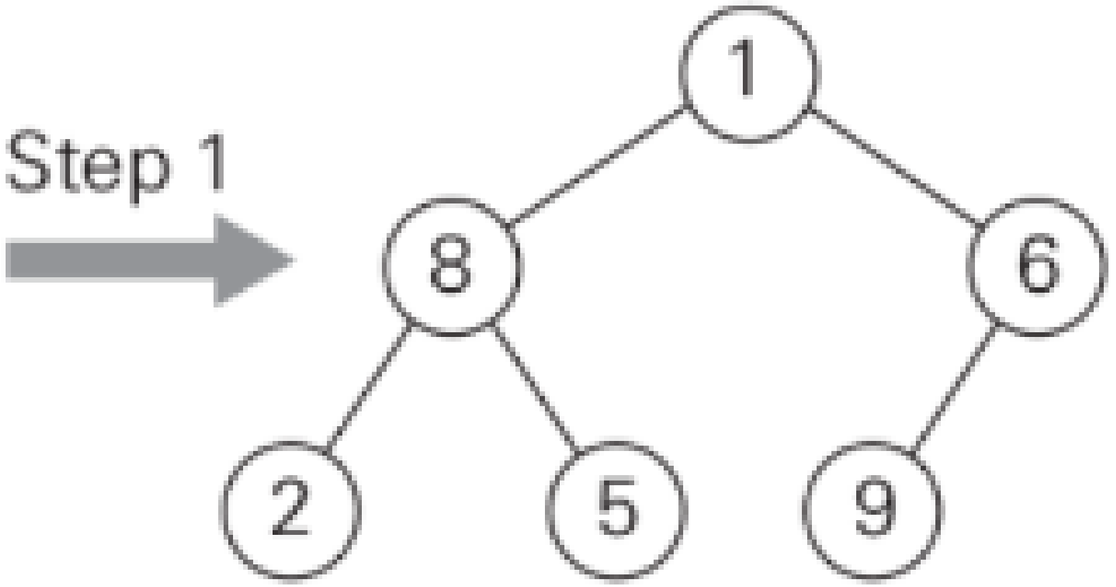


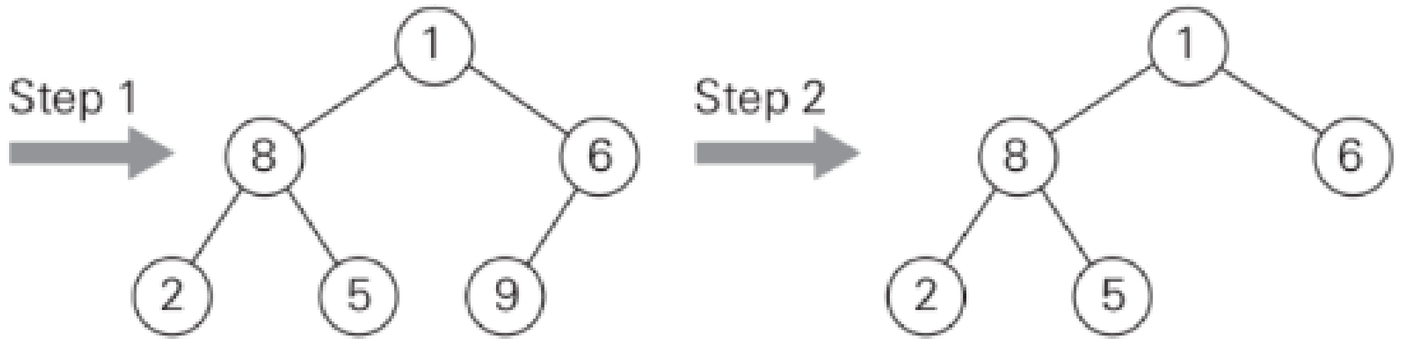
O tal do pop

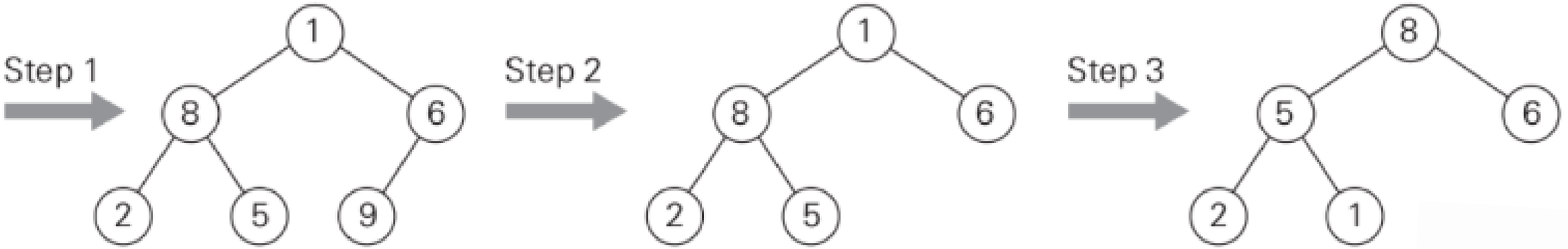
THE LARGEST ELEMENT IS THE ROOT (MAX-HEAP)

1 Exchange the root's key with the last key K.
2 Decrease the heap's size by 1.
3 Heapify the tree by sifting K down (see bottom-up algorithm).




Step 1
1
8
6
2
5
9


Step 1
1
8
6
2
5
9
Step 2
1
8
6
2
5


Step 1
1
8
6
2
5
9
Step 2
1
8
6
2
5
Step 3
8
5
6
2
1

FIm

PERGUNTAS?