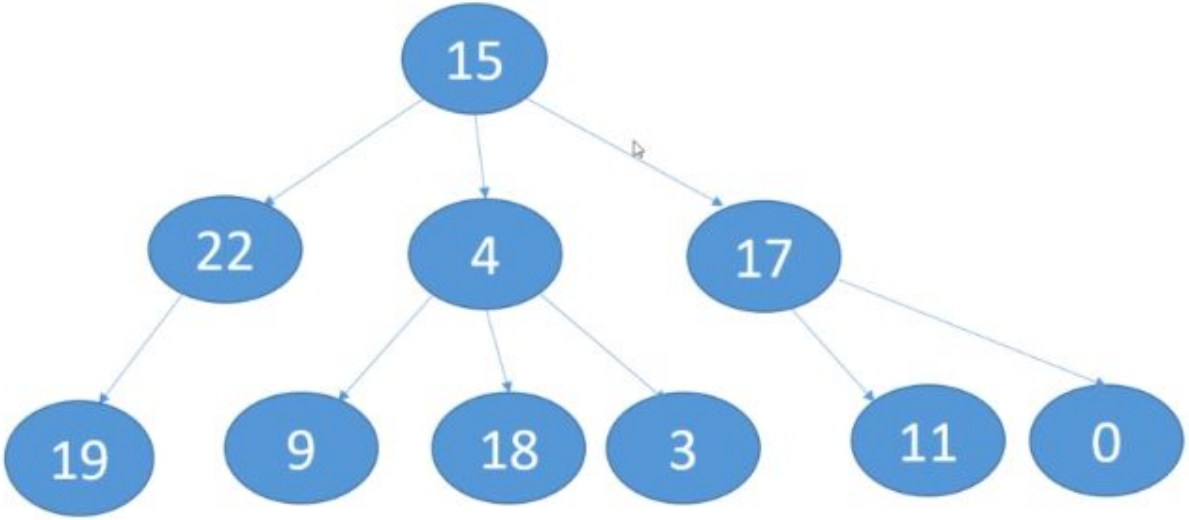
Heaps

Agenda

- Define tree
- Parts of a tree
- Types of trees
- Define heap
- Define heapify
- Adding/Deleting from heap

Tree

- Hierarchical data structure.
- It has nodes.



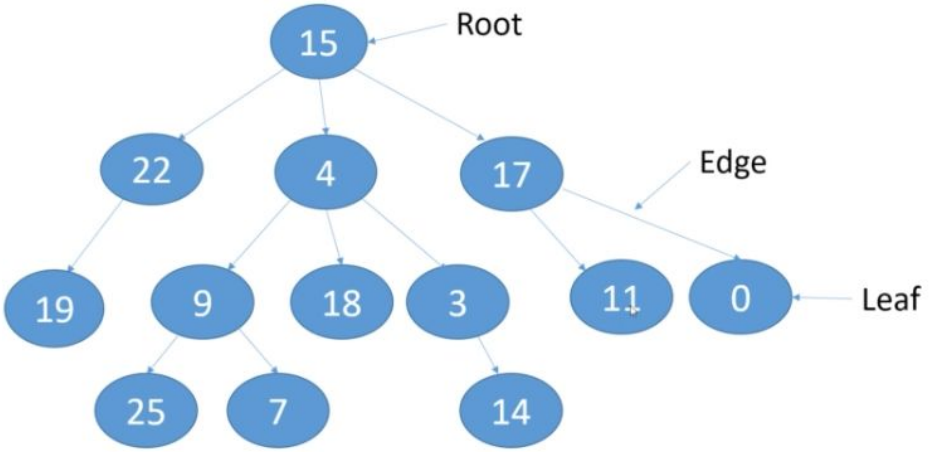
Parts of a trees

- Nodes have **children.**

- Each node can have only have one **parent.**

- Special node called **root**

(only one root). Do not have a parent.



- A **leaf** node has no children.

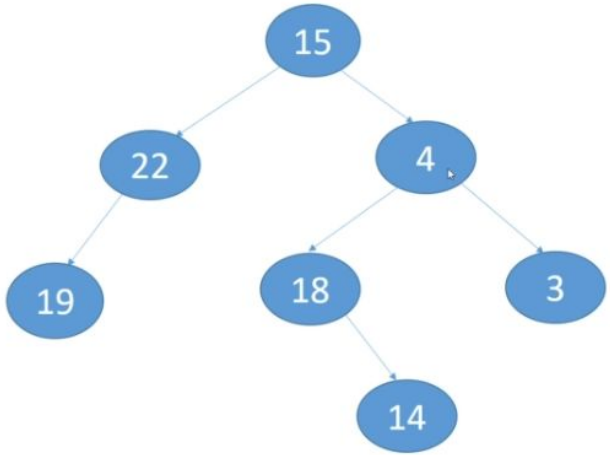
- An **edge** connects two nodes to show that there is a relationship between them.

- A **path** is the sequence of nodes along the edges of a tree. A path does not cross a node more than one.

Binary Tree

Every node has 0, 1 or 2 children

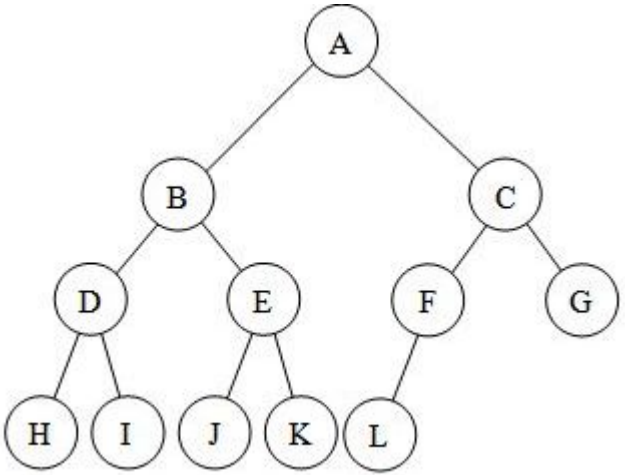
Children are referred as left child and right child



Complete Binary Tree

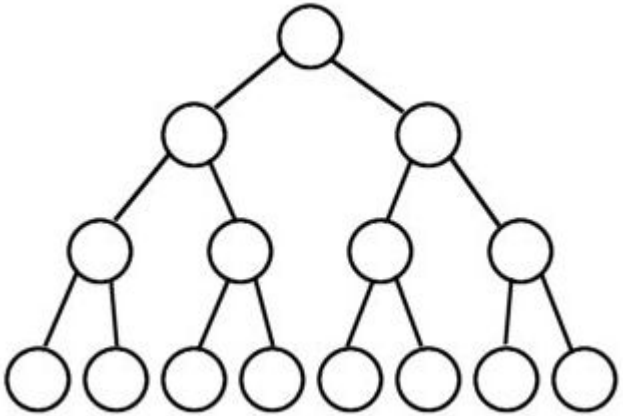
If every level is complete (has two children) except the last level.

On the last level children are all to the left as much as possible.

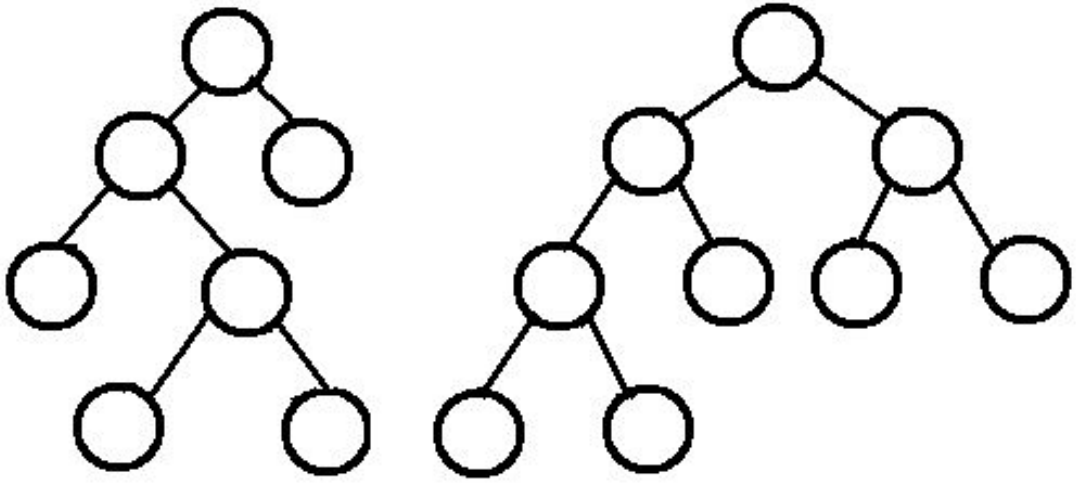


Full Binary Tree

Every node except the leaves has 2 children.



Full and Complete Binary Trees

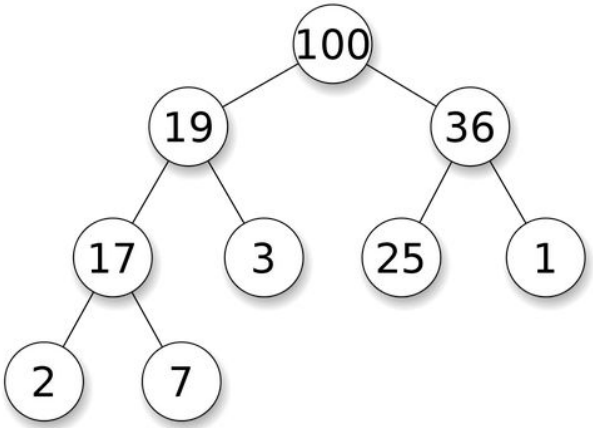


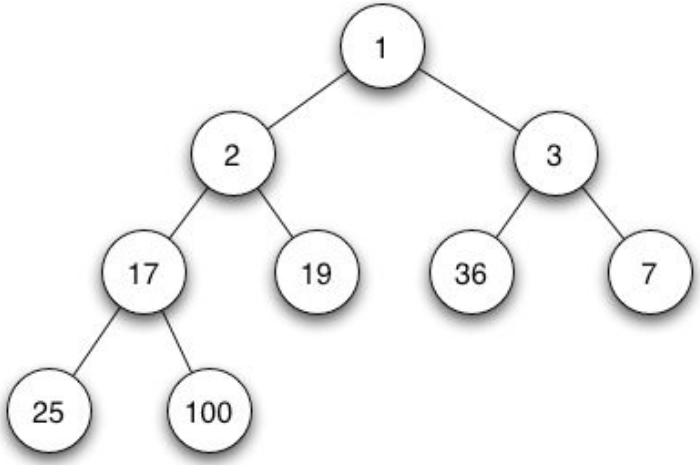
full tree

complete tree

Heaps

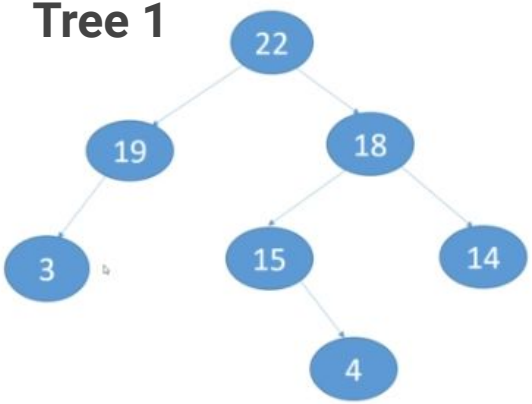
- Complete binary tree (every level full except the last one)
- The tree must satisfy the heap property:
  - **Max heap:** Every parent is greater than or equal to its children
  - **Min heap:** Every parent is less than or equal to its children





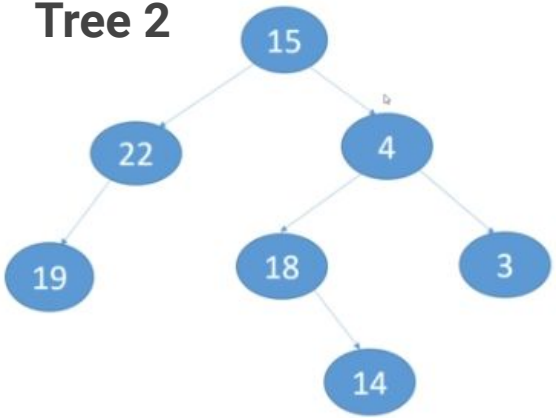
Are these heaps?

Tree 1



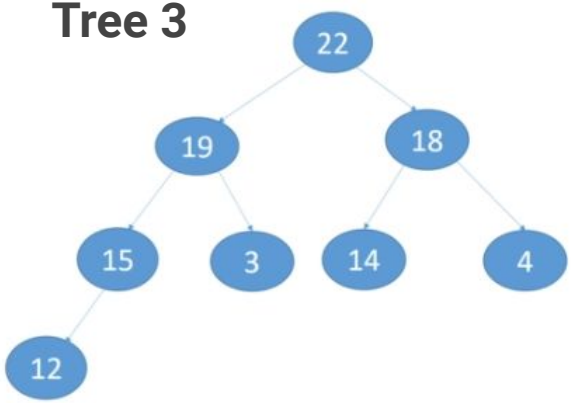
No!
It looks like a max heap, but it is not a complete binary tree.

Tree 2



No!
It is not a complete binary tree and it does not meet the parent-child value relationship.

Tree 3



Yes!
It is a complete binary tree and it is max heap.

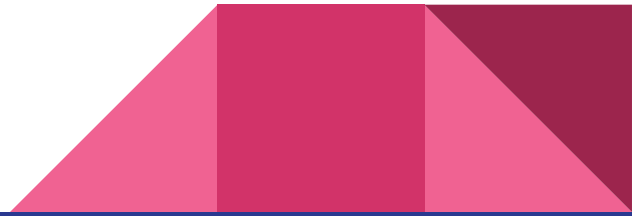
Heaps - Rules

Children are added at each level from left to right

Because of the property of the heap:

The maximum or minimum value will always be at the root of the tree. So, what is the time complexity of getting the min (min heap) or max (max heap) values in a from a heap?

O(1) => constant time (advantage of heaps)

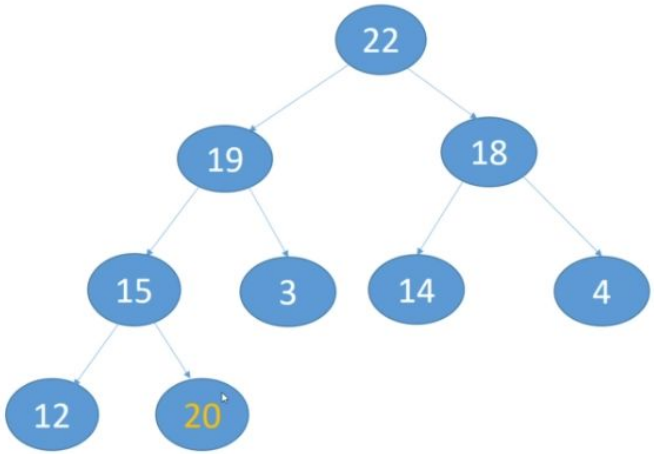


What happen when we delete or insert values? How do we recover the properties of the heap?

We have to **heapify** to recover the properties of the heap.

Heapify: process of converting a binary tree into a heap this often has to be done after an insertion or deletion

*** We do not care about the order between siblings. The order between parent and children is important ***



Time for some exercises!!!!!

Go to our **apcsa_materials repo** and look for:

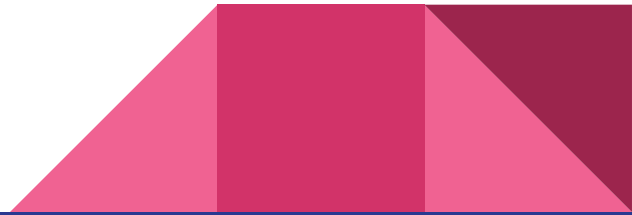
classwork/heaps/heaps_exercise.md

How can we store heaps?

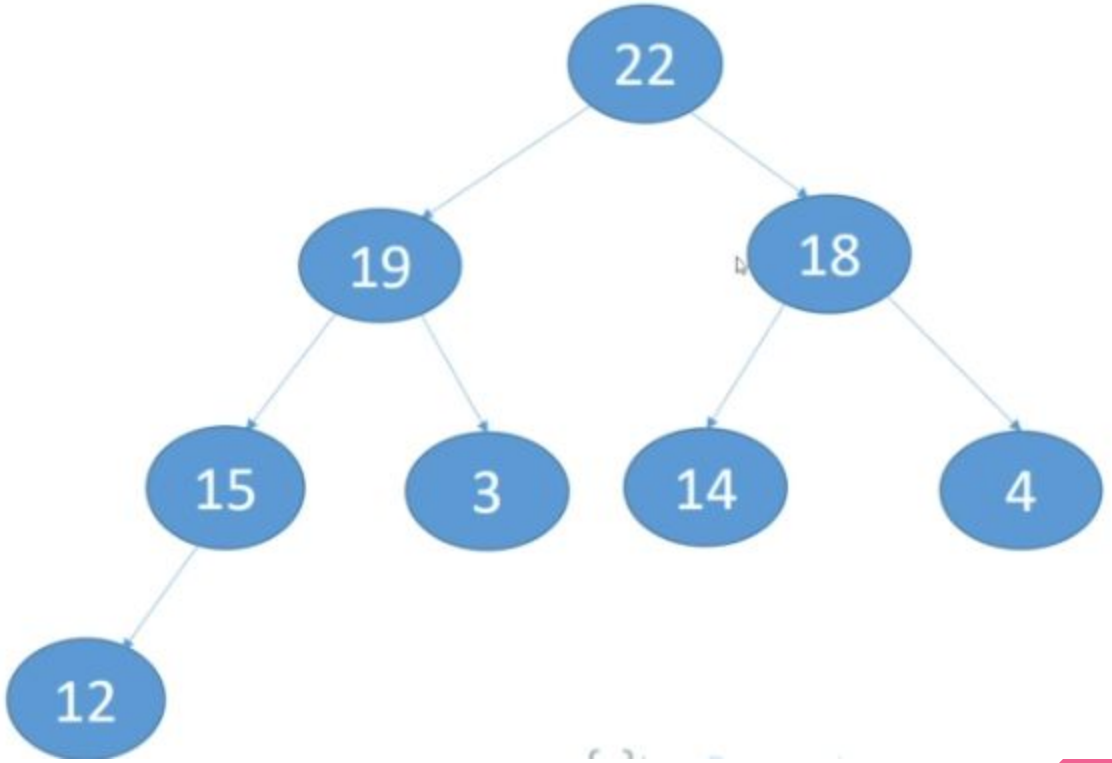
Storing heaps as arrays???

YES!!! We can store heaps as arrays

- The root will go at array[0]
- Traverse each level in the tree from left to right and so,
  - the left child of the root would go into array[1]
  - the right child of the root would go into array[2]
- Then the left child of the left child of the root would go into array[3] and the right child of the left child of the root would go into array[4] and so.



0	1	2	3	4	5	6	7
22	19	18	15	3	14	4	12



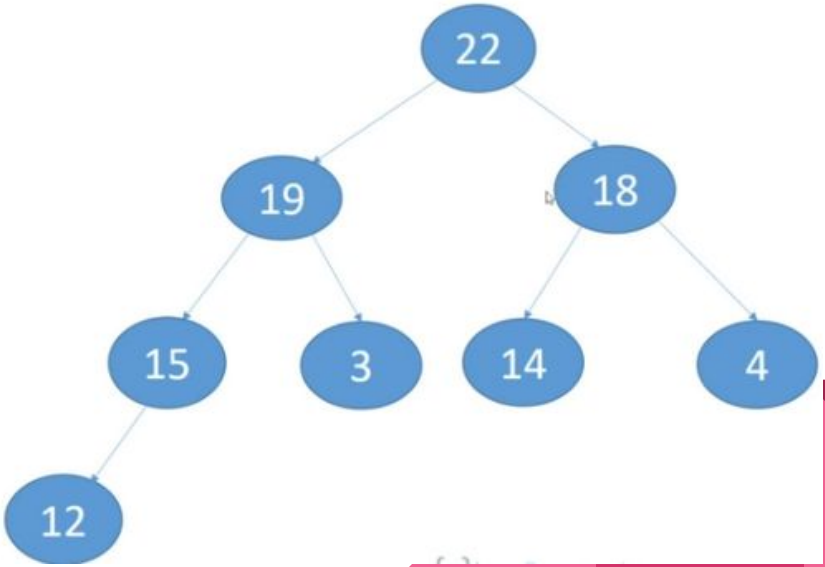
0	1	2	3	4	5	6	7
22	19	18	15	3	14	4	12

For the node at array[i]:

left child = 2i + 1

right child = 2i + 2

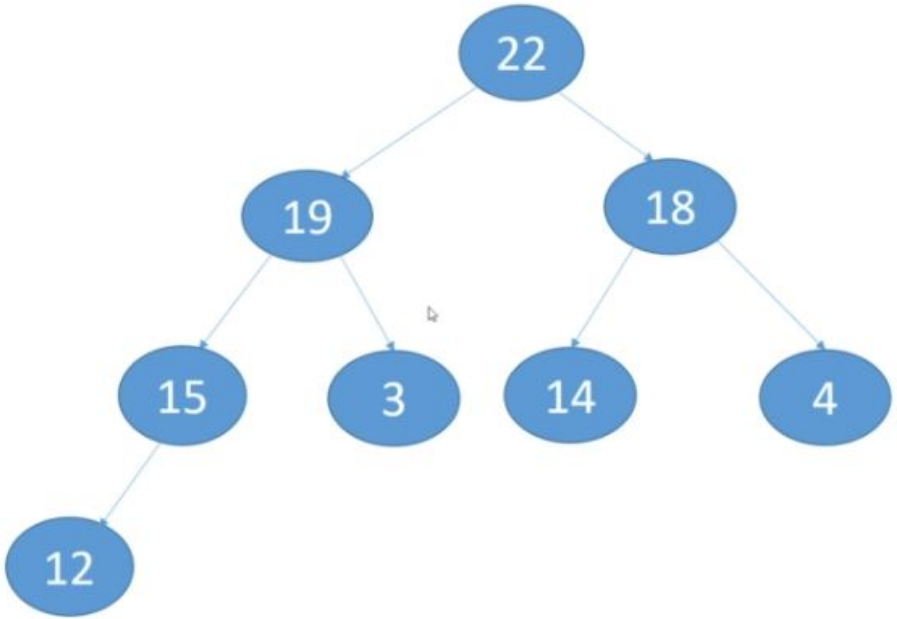
parent = (i - 1) / 2

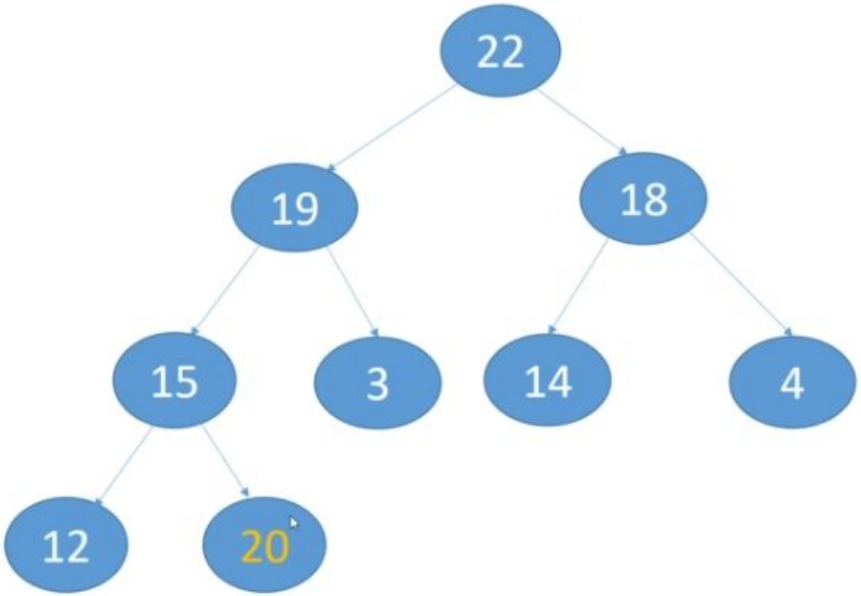


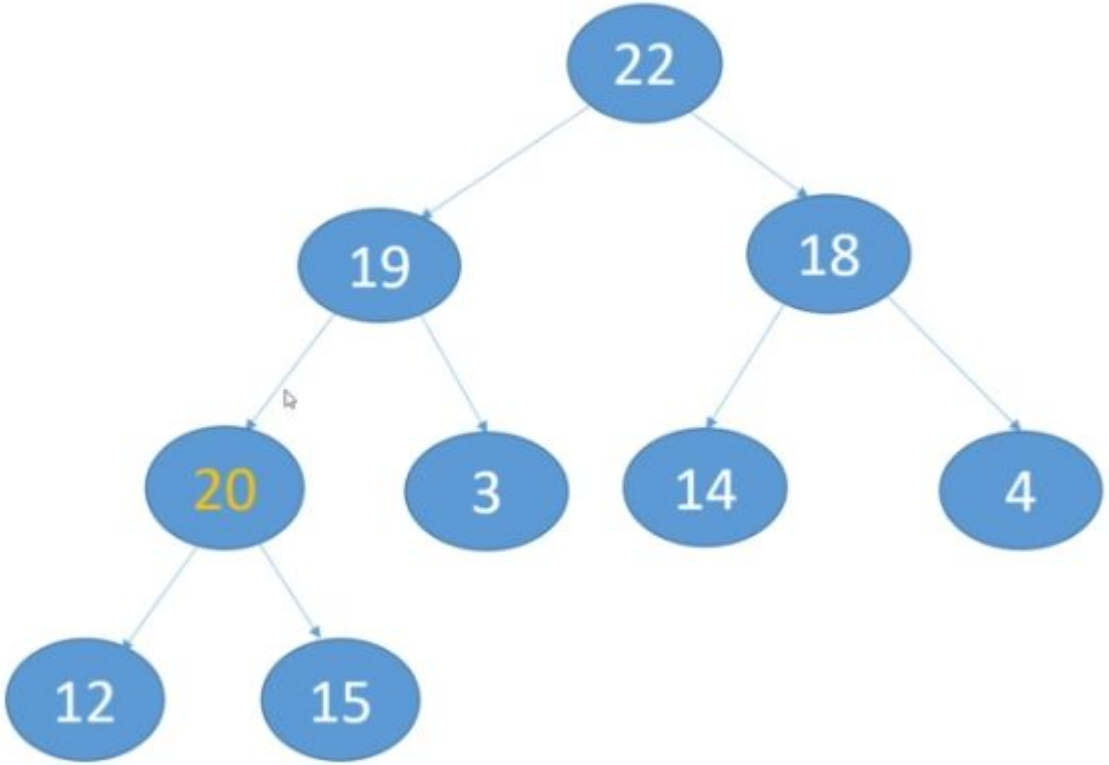
Insert into Heap

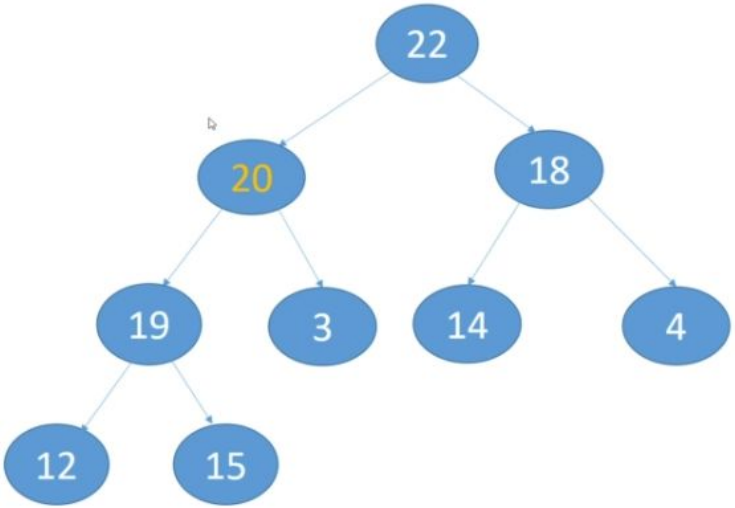
- Always add new items to the end of the array
- Then we have to fix the heap (heapify)
- We compare the new item against its parent
- If the item is greater than its parent, we swap it with its parent
- Repeat process until heap meets property

Example: Add 20 to the heap




22
19
18
15
3
14
4
12
20

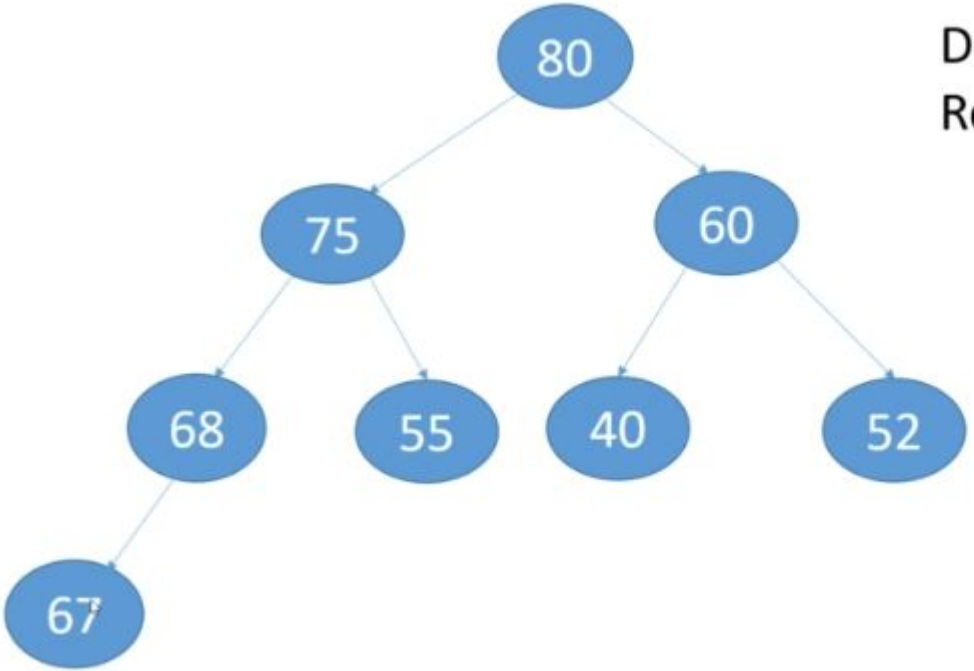

22
19
18
20
3
14
4
12
15



Storing the heap as an array, when we're comparing, all we have to do is **calculate where the node's parent is**, **compare it with the new value** in the new node. If it is **less than the parent** value, we just **swap** them (max heap), this swapping works because the characteristic of the heap.

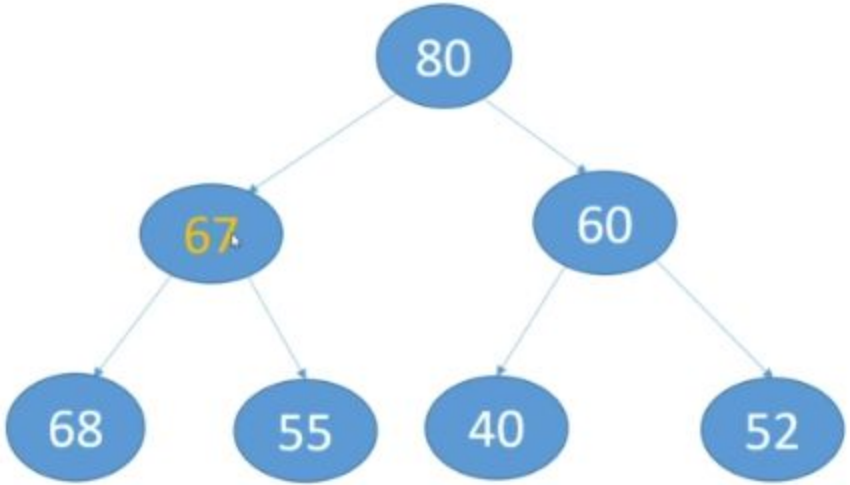
Heaps - Delete

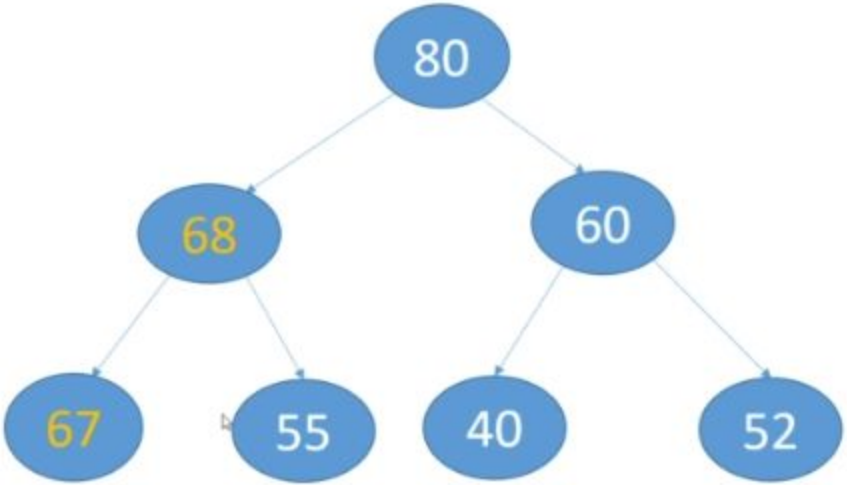
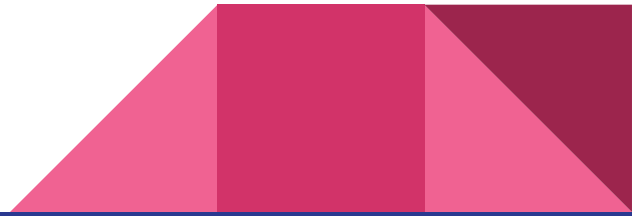
- Must choose a replacement value
- Will take the rightmost value, so that the tree remains complete
- Then we must heapify the heap
- When replacement value is greater than parent: fix heap above. Otherwise, fix heap below.



Delete 75

Replacement value: 67


80
67
60
68
55
40
52


80
68
60
67
55
40
52

Coding Time!!!!!