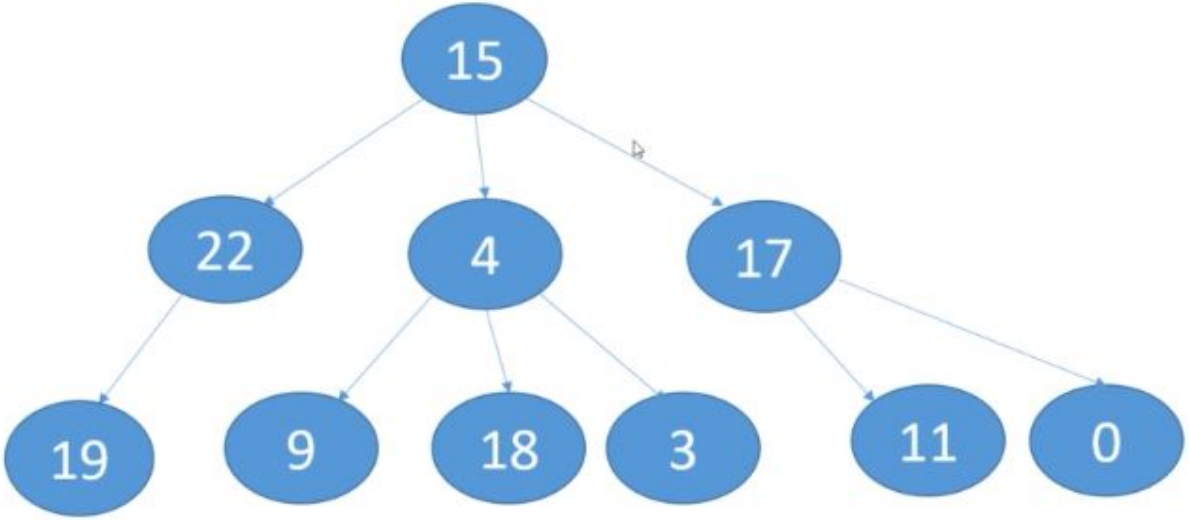
Heaps

Tree

- Hierarchical data structure.
- It has nodes.



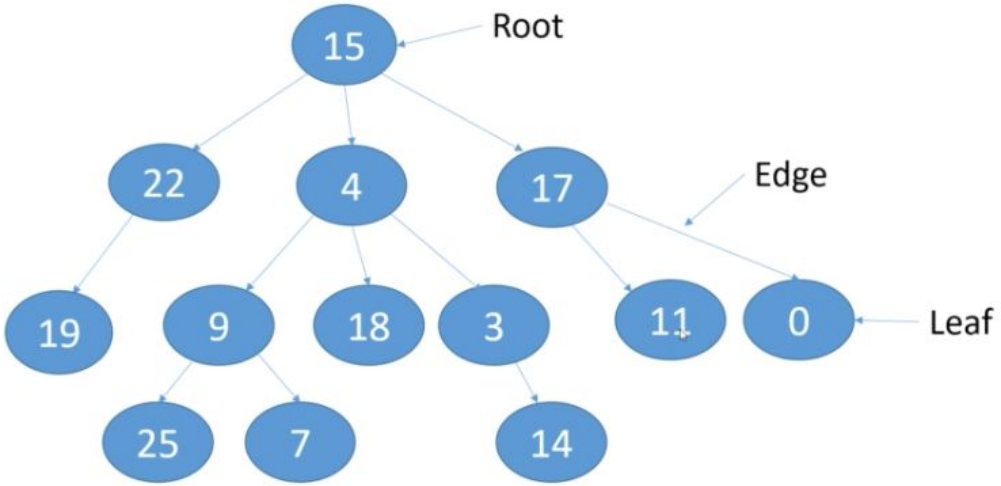
Trees

- Nodes have **children.**

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

- Special node called **root** (only one root). Do not have a parent.

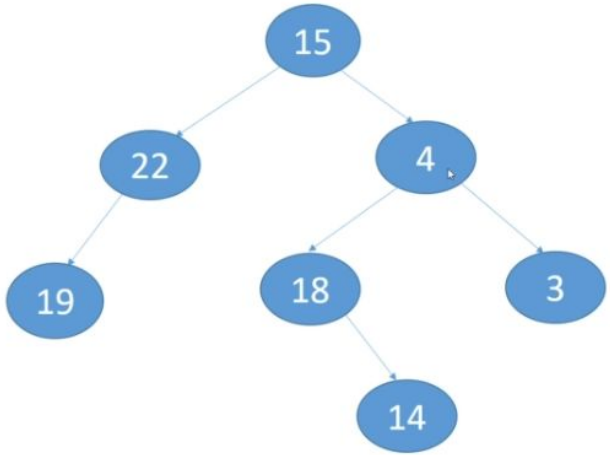
- A **leaf** node has no children.



Binary Tree

Every node has 0, 1 or 2 children

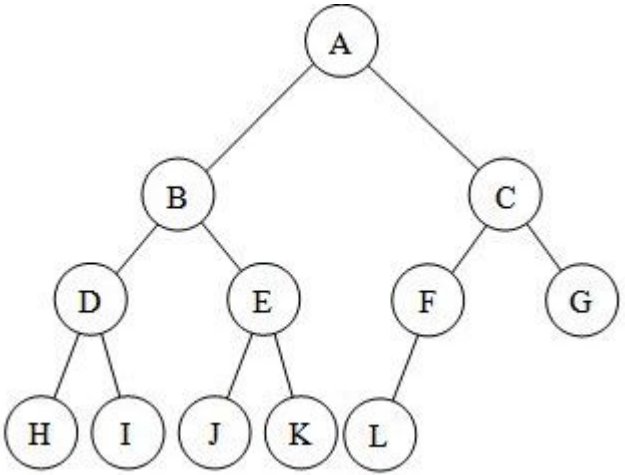
Children are referred as left child and right child



Complete Binary Tree

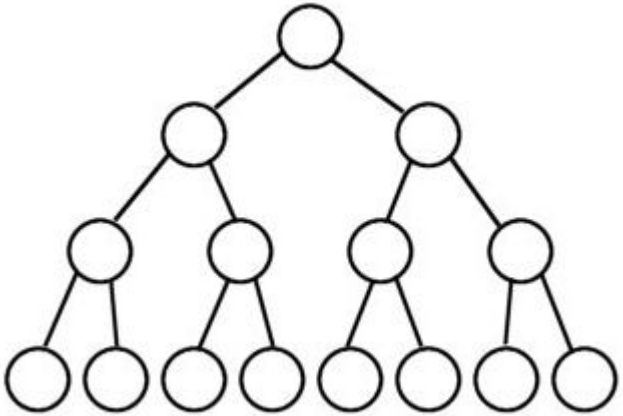
If every level is complete (has two children) except the last level.

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

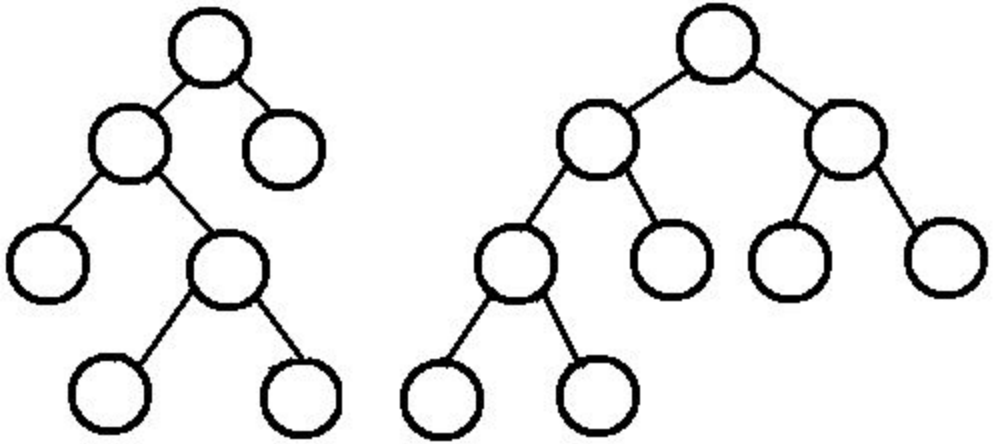


Full Binary Tree

Every node except the leaves has 2 children.



Full and Complete Binary Trees

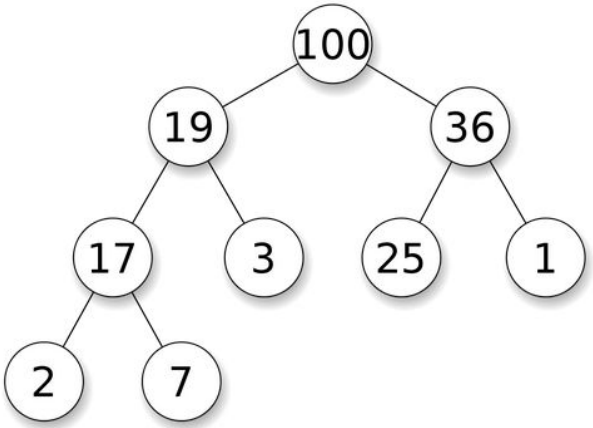


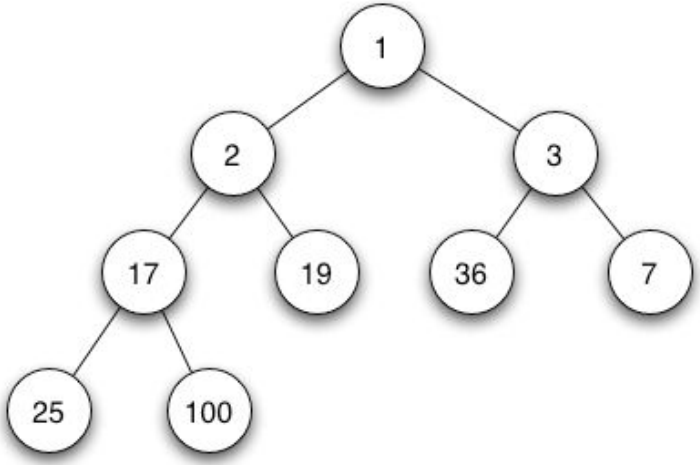
full tree

complete tree

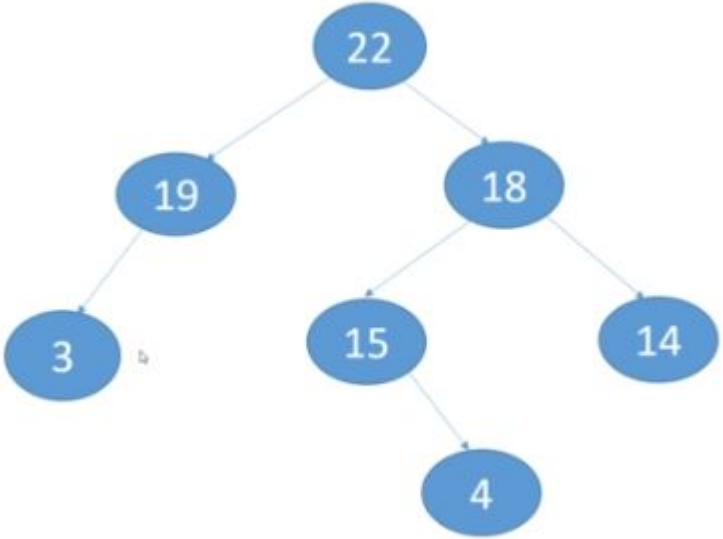
Heaps

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

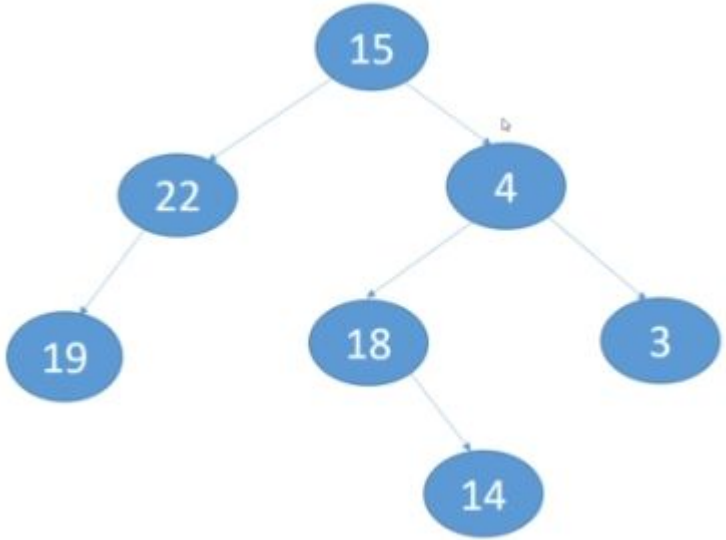




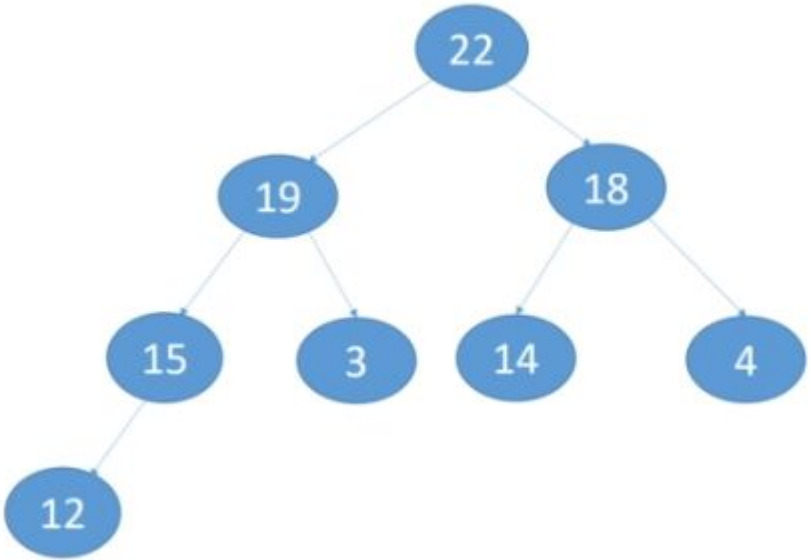
Is this a heap?



Is this a heap?



Is this a heap?



Heaps

Children are added at each level from left to right

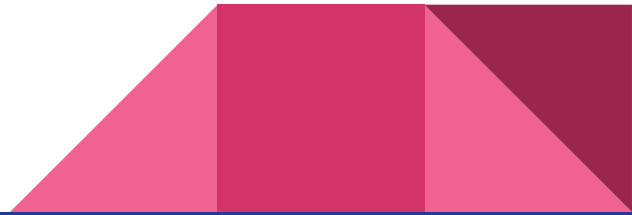
Because of the property of the heap:

The maximum or minimum value will always be at the root of the tree - Get min (min heap) and max (max heap) at constant time (advantage of using a heap)

What happen when we delete or insert values we need to heapify to recover the property of heaps

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

No required ordering between siblings (the order between parent and children is important but not for siblings)

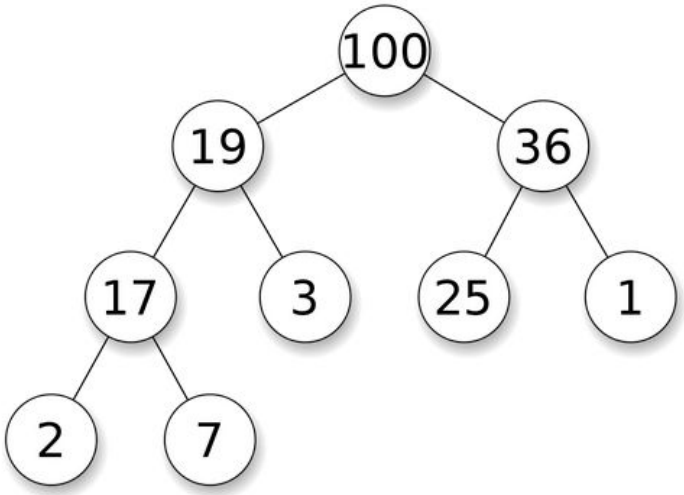


Exercise: Adding - Removing

Let's **add** a value to the heap:

HINT: If you add a new node it should be added as the left child of the 3. And depending on the value of the element it may not go there.

1. How would you add the value 200 to this heap?
2. How would you add the value 20 to the new heap?



Now let's **remove**:

HINT: Replace the root node with the lowest, right-most item

on the heap.

1. How would you remove the 200?
2. What position ultimately must have no value?
3. What would a valid heap look like afterwards?
4. How would you then remove the next largest?

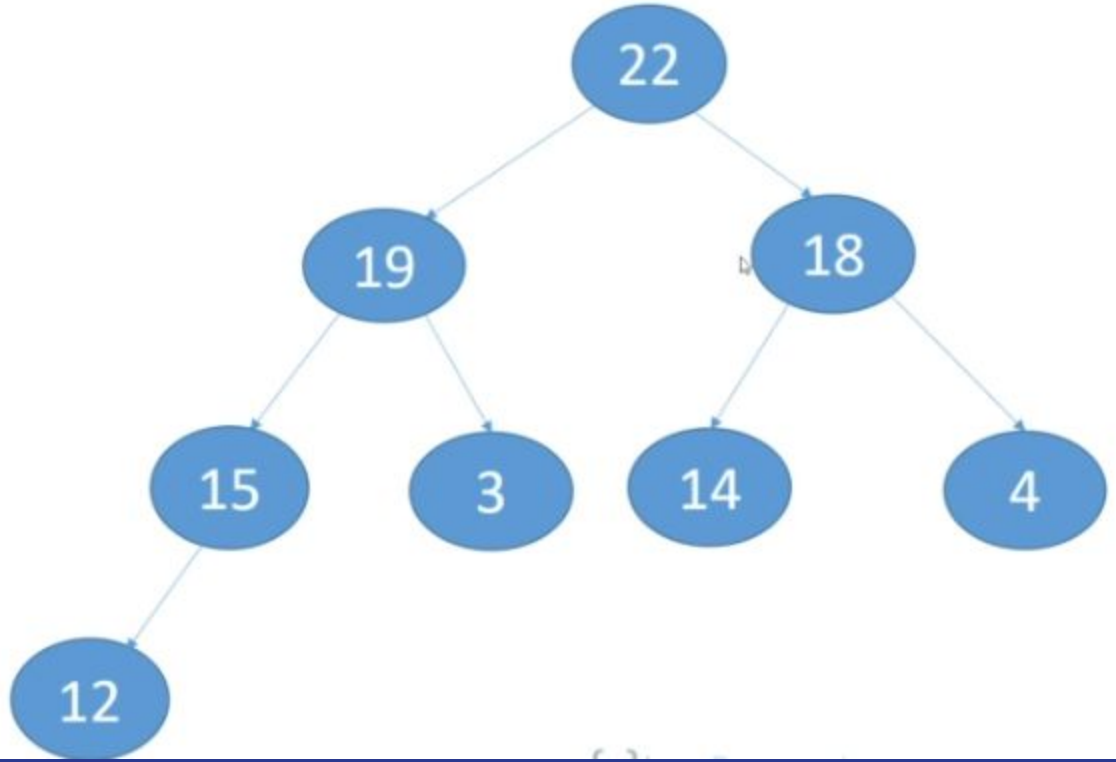
How can we store heaps?

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



For the node at array[i]:

left child = 2i + 1

right child = 2i + 2

parent =(i - 1) / 2

Insert into Heap

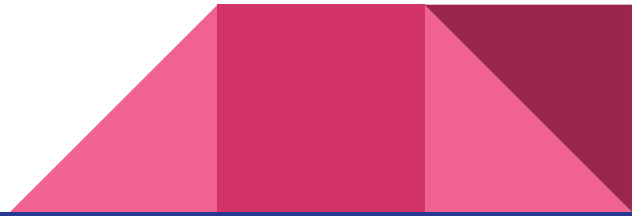
Always add new items to the end of the array

Then we have to fix the heap (heapify)

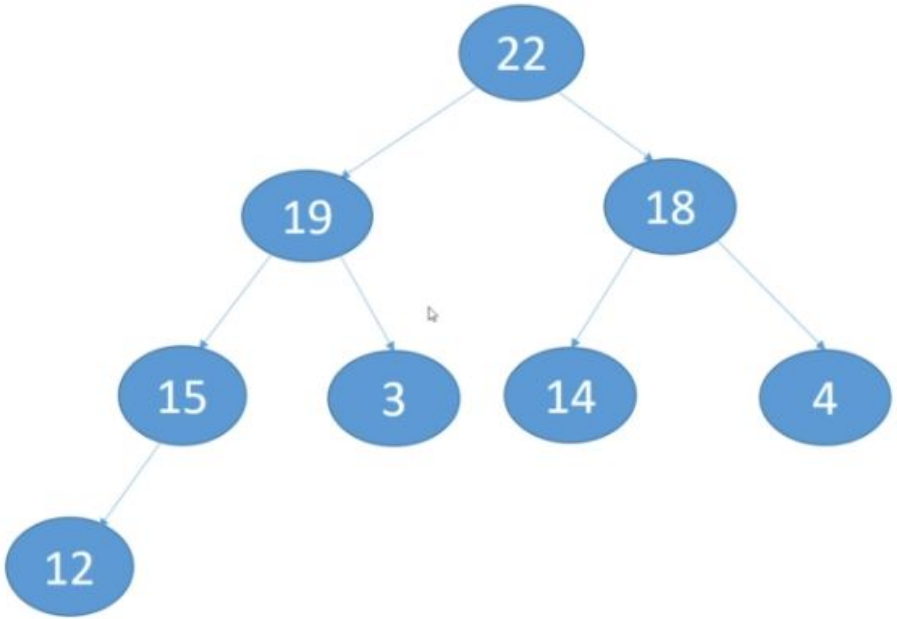
We compare the new item against its parent

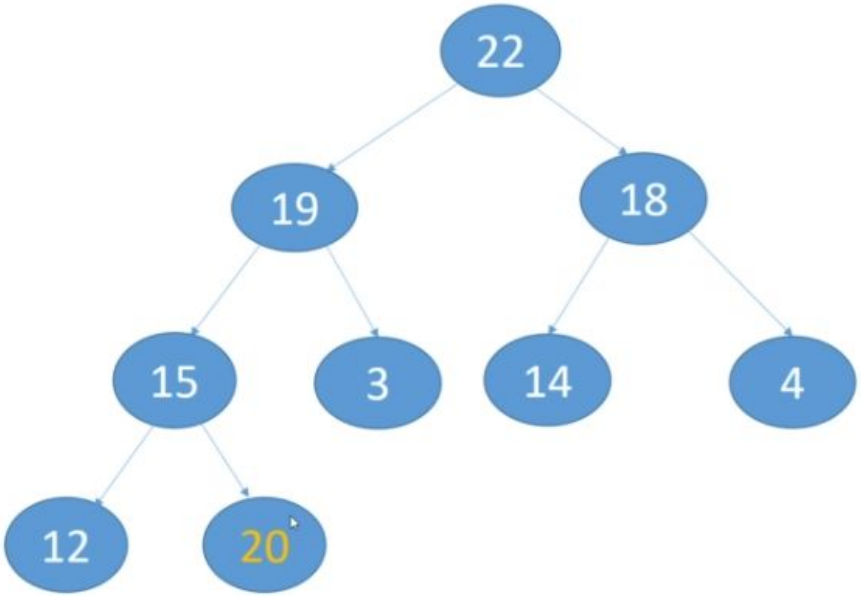
If the item is greater than its parent, we swap it with its parent

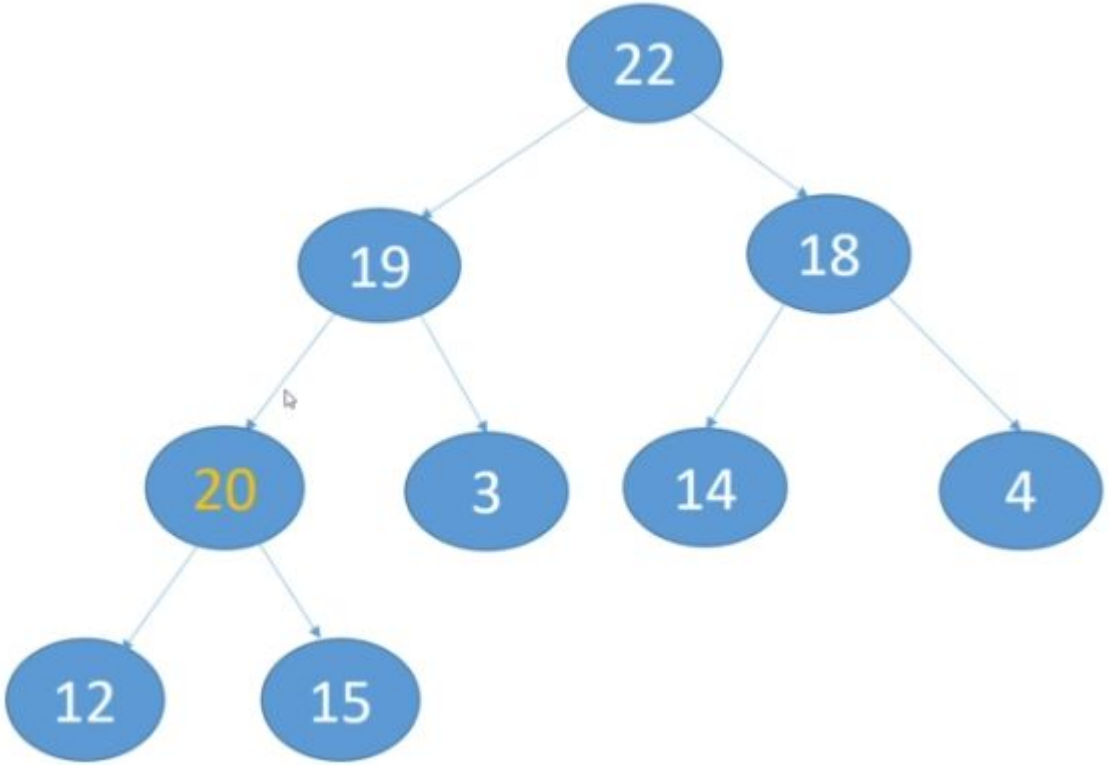
Repeat process until heap meets property

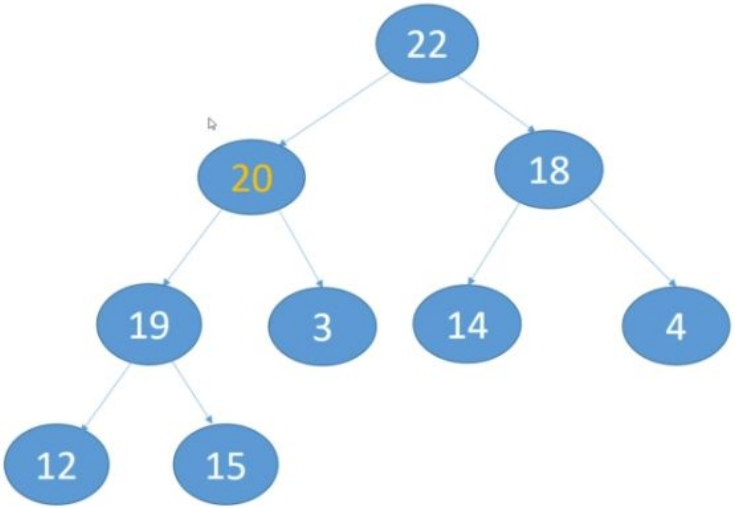


Example: Add 20 to the heap




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.

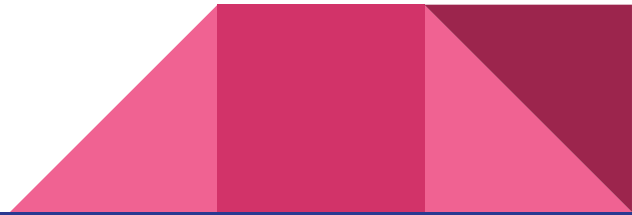
Heaps - Delete

Must choose a replacement value

Will take the rightmost value, so that the tree remains complete

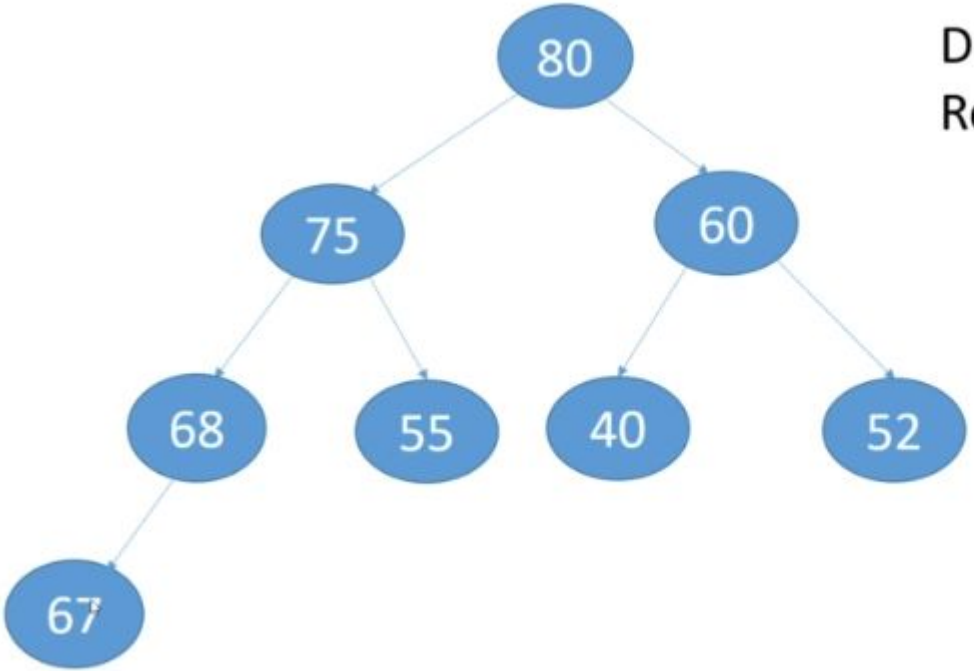
Then we must heapify the heap

When replacement value is greater than parent: fix heap above. Otherwise, fix heap below.



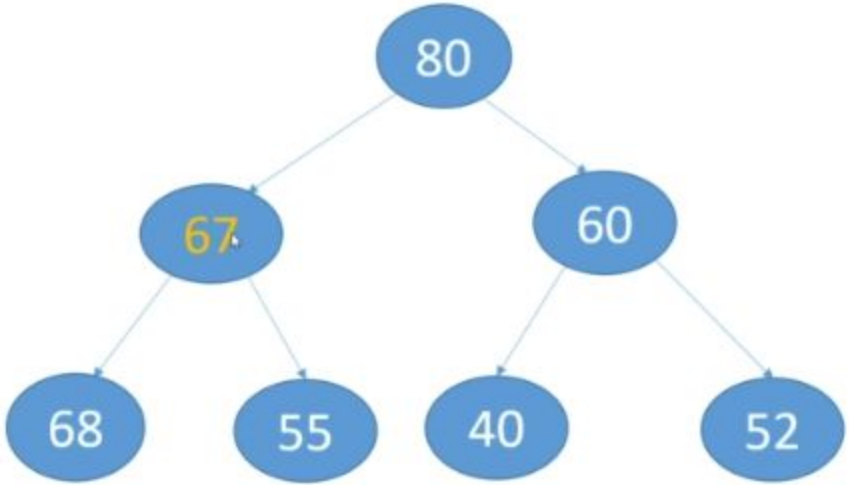
Heaps - Delete

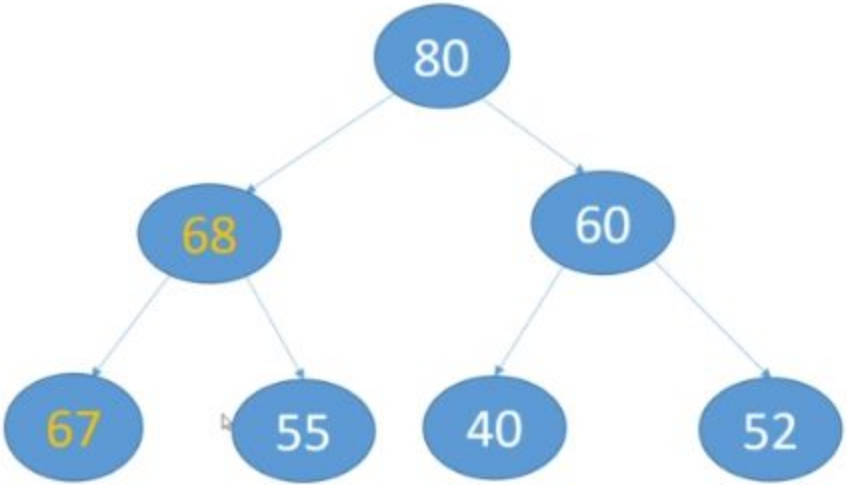
- Fix heap above – same as insert. Swap replacement value with parent
- Fix heap below – swap the replacement value with the larger of its two children
- Rinse and repeat in both cases until the replacement value is in its correct position



Delete 75

Replacement value: 67


80
67
60
68
55
40
52


80
68
60
67
55
40
52