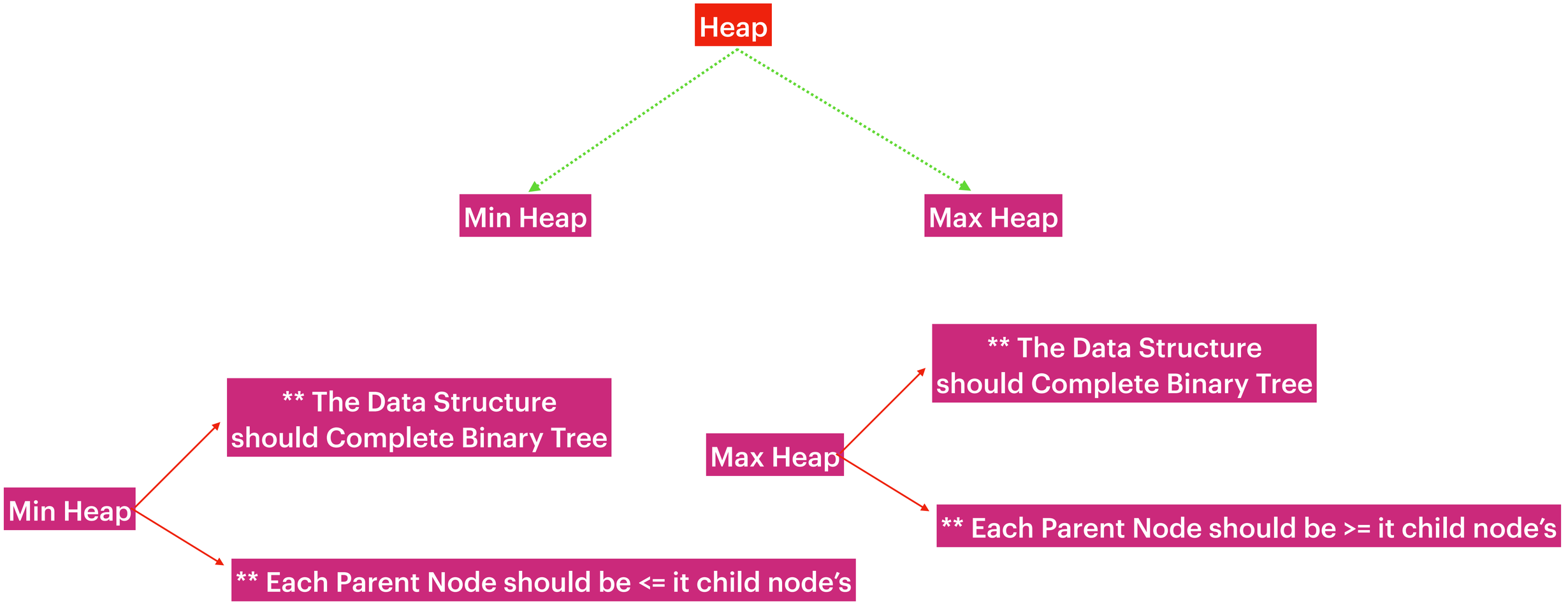
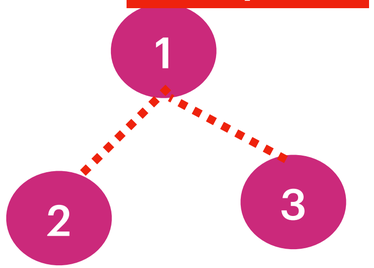
Let's Understand Heap Before moving on to weighted Graphs



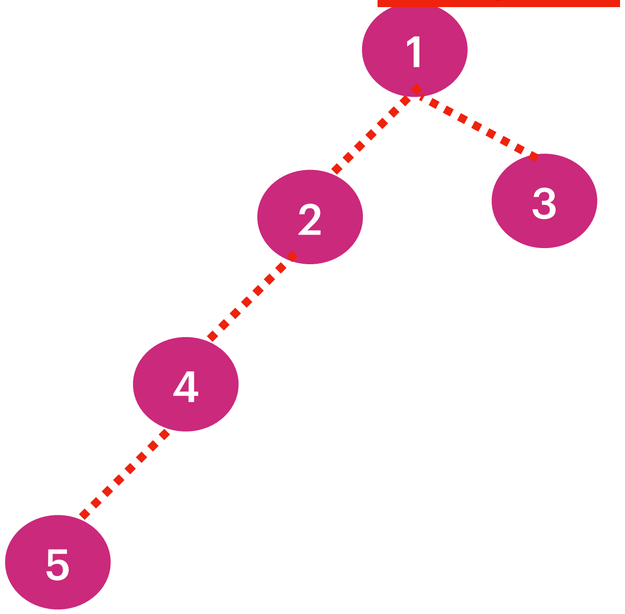
What is Binary Tree ?

Each Parent node at max have two Childs

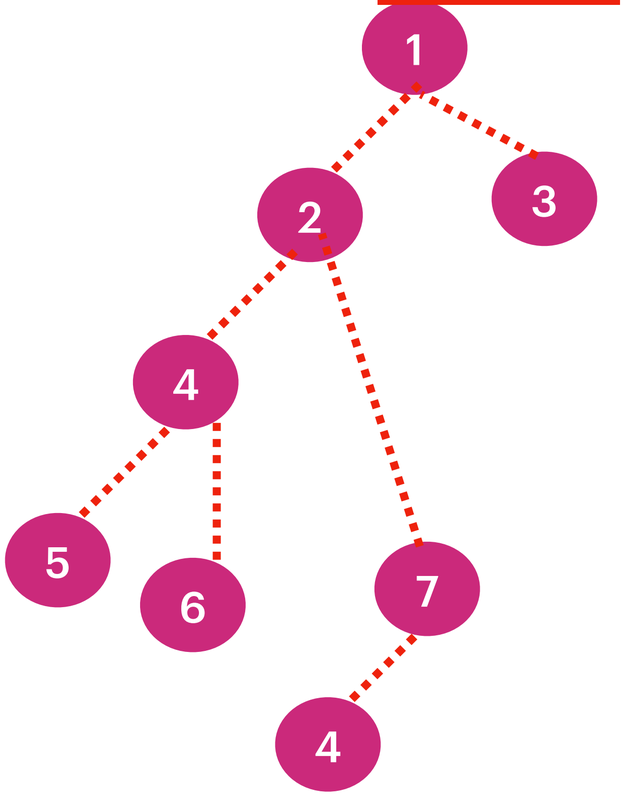
Binary Tree



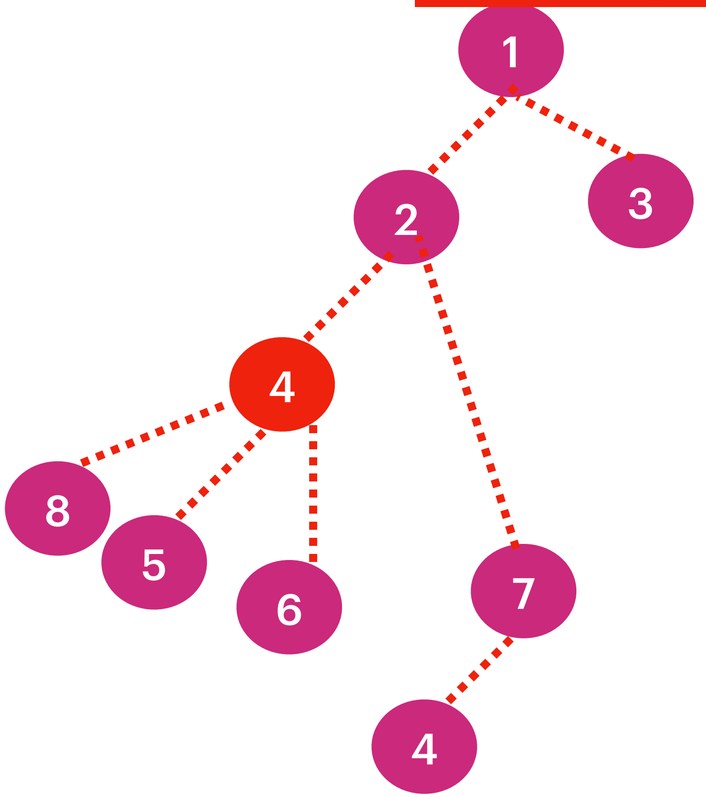
Binary Tree



Binary Tree



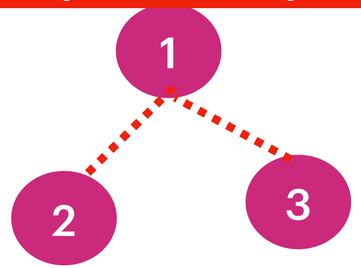
Not a Binary Tree



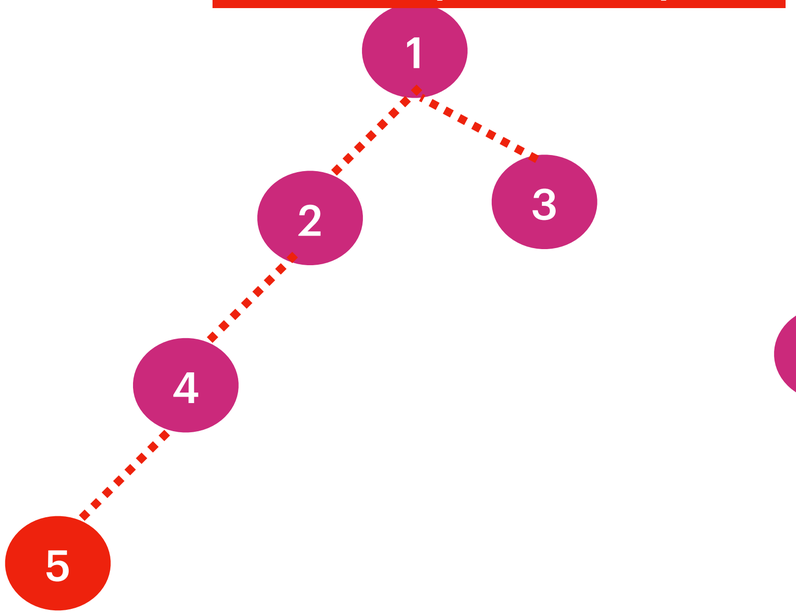
What is Complete Binary Tree ?

Each Parent node at max have two Childs,
At Each level order of elements should
Be filled from left -> right.
Once the the current level complete we can
Move on to next level.

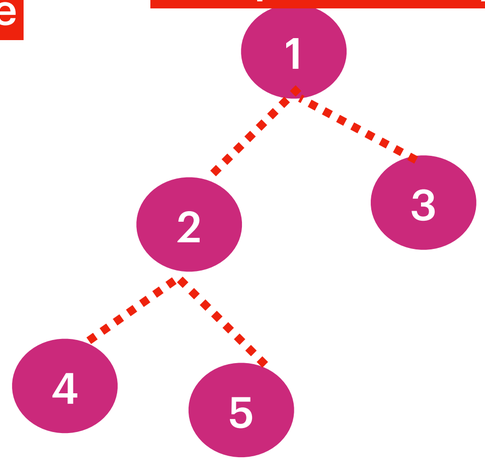
Complete Binary Tree



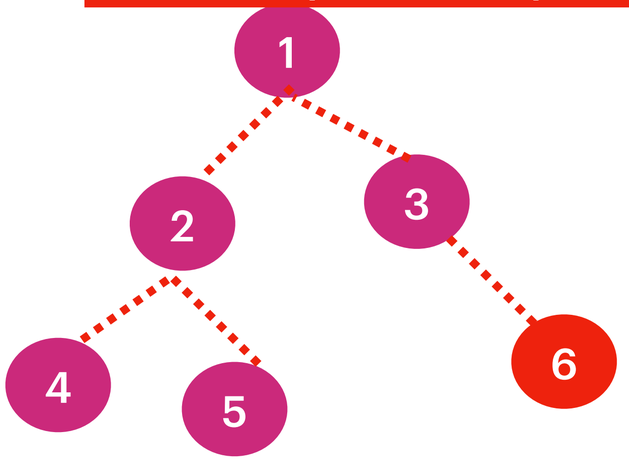
Not a Complete Binary Tree



Complete Binary Tree



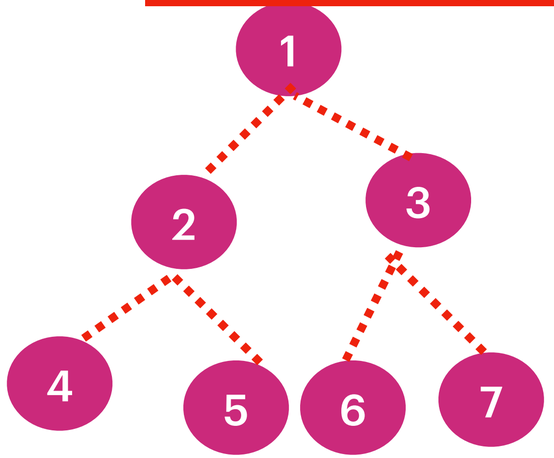
Not a Complete Binary Tree



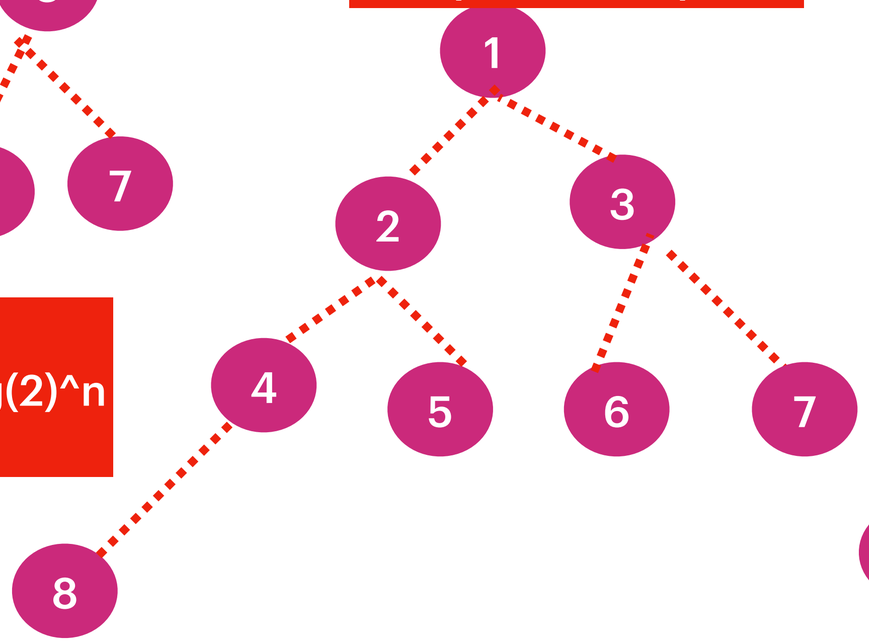
What is Complete Binary Tree ?

Each Parent node at max have two Childs,
At Each level order of elements should
Be filled from left -> right.
Once the the current level complete we can
Move on to next level.

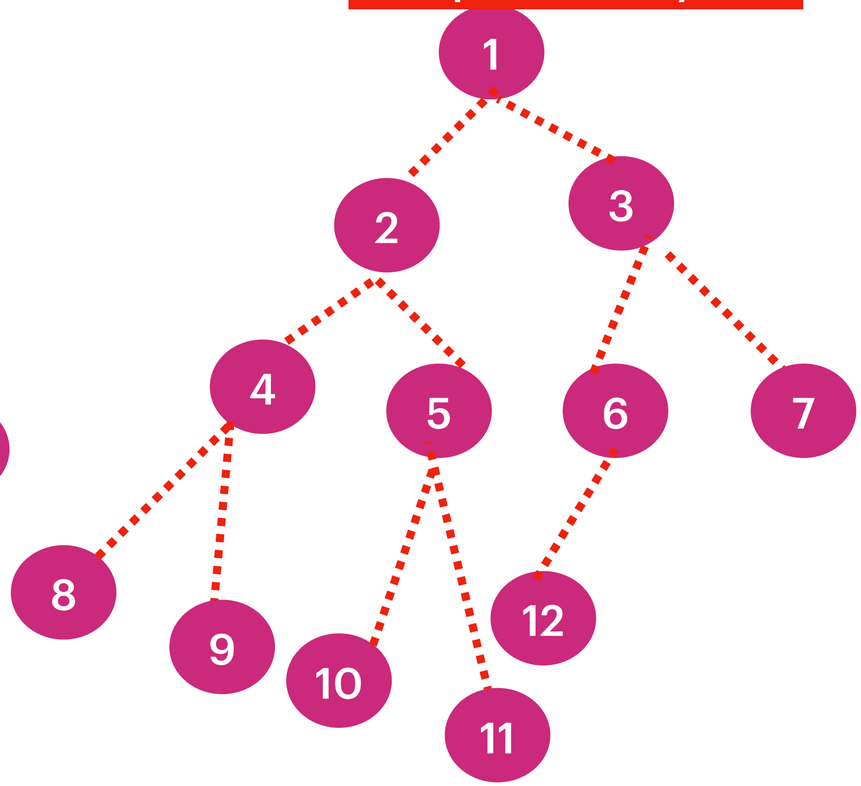
Complete Binary Tree



Complete Binary Tree

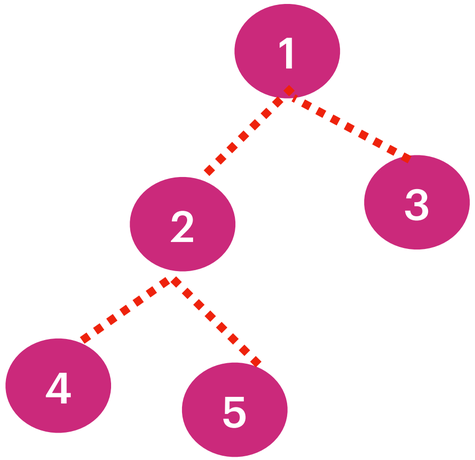


Complete Binary Tree



Height
Of Complete Binary Tree : log(2)^n

Complete Binary Tree



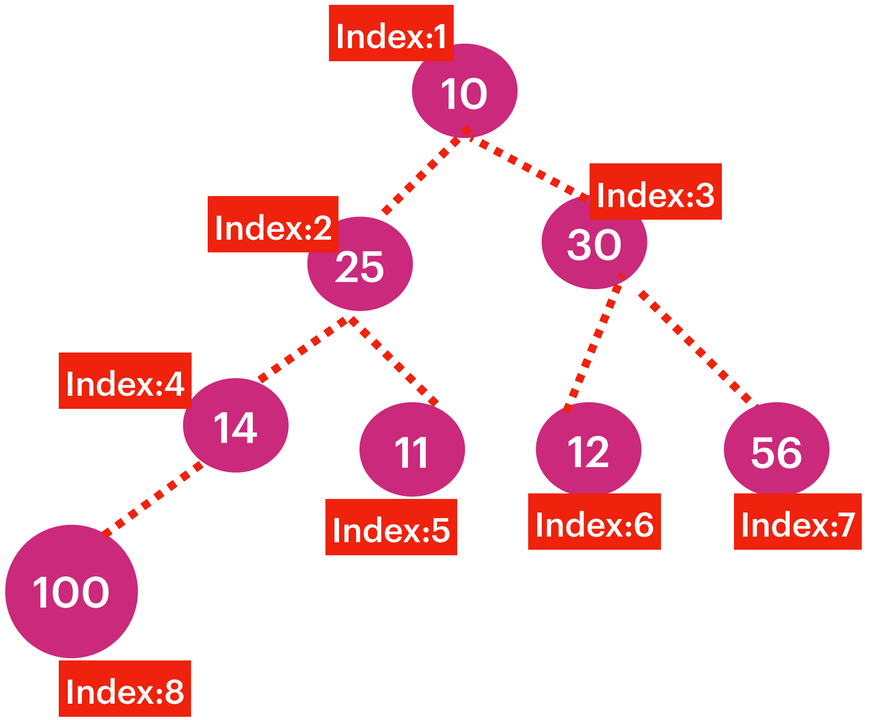
Root : 1

Leaf Nodes : 3 4 5

A Node which does not have Childs is a leafNode

Height : 3 ~ log(2)^n

Complete Binary Tree



Transform Complete Binary Tree to the Hash/Array

How to you find parent ?

How do you find left & right child ?

How do you know that current element is a leaf Node ?

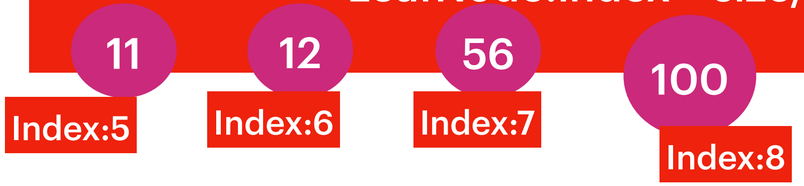
Left Child = index * 2
Right Child = index * 2 + 1

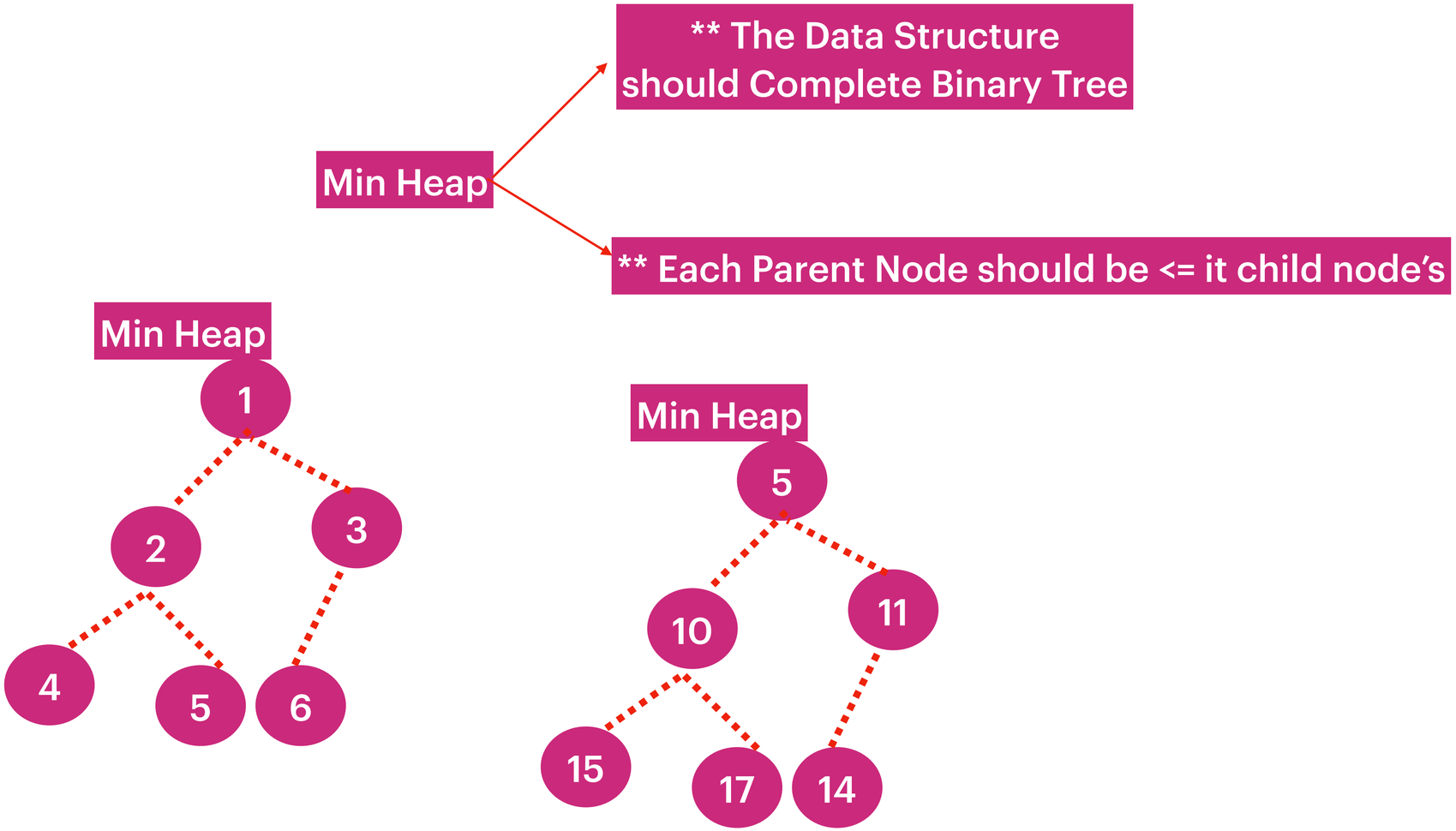
Ex: Left Child of Element 30 = 2 * index:3 = index:6 = 12
Right Child of Element 30 = 2 * index:3 + 1 = index:7= 56

Parent = index/2;

Ex: Parent Of 25 = 2/2 = index:1 = 10

How do you know that current element is a leaf Node ?
LeafNode:index > size/2




** The Data Structure
should Complete Binary Tree
Min Heap
** Each Parent Node should be <= it child node's
Min Heap
1
2
3
4
5
6
Min Heap
5
10
11
15
17
14

Insert Operation On MinHeap

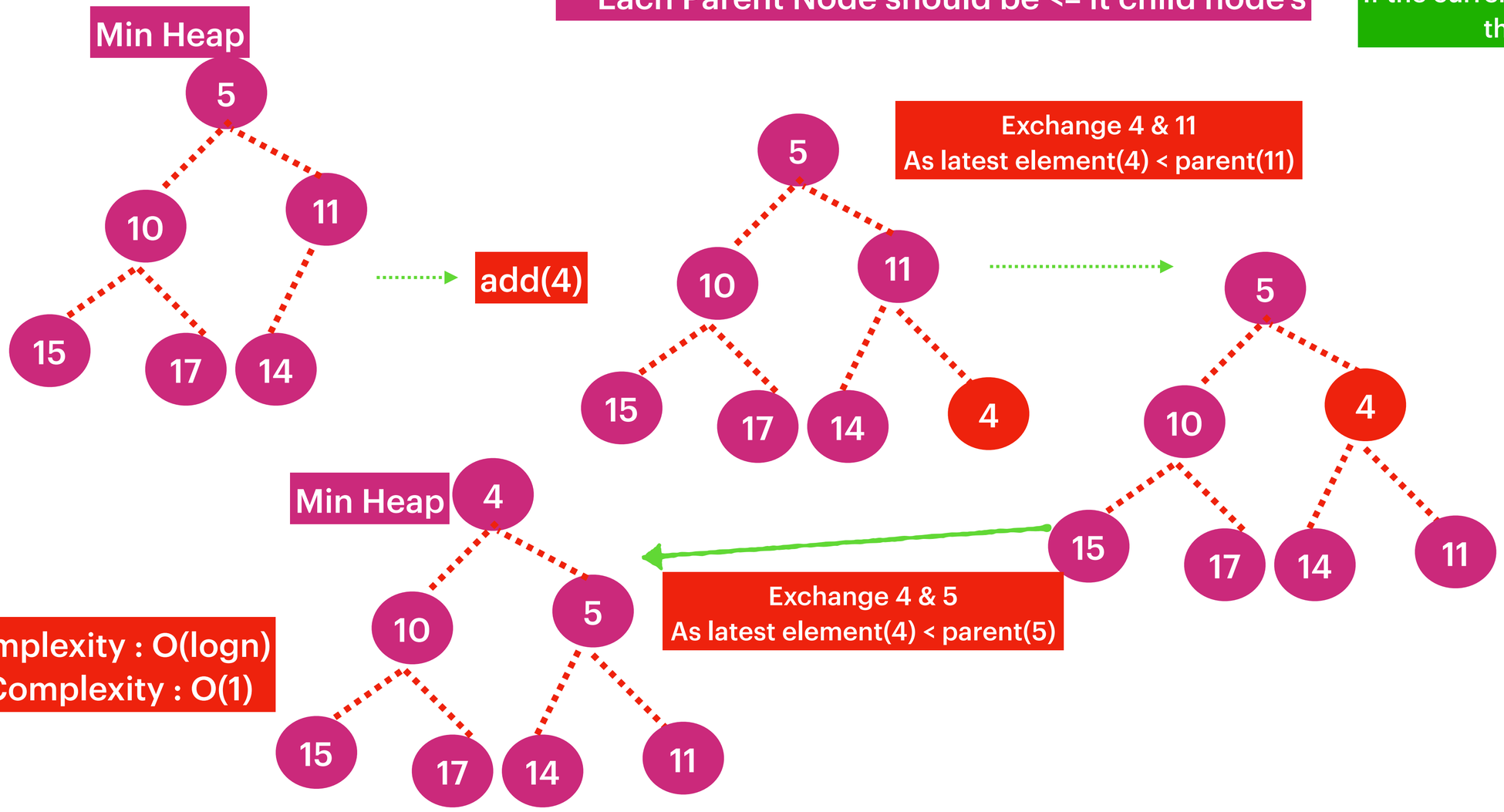
Min Heap

** The Data Structure should Complete Binary Tree

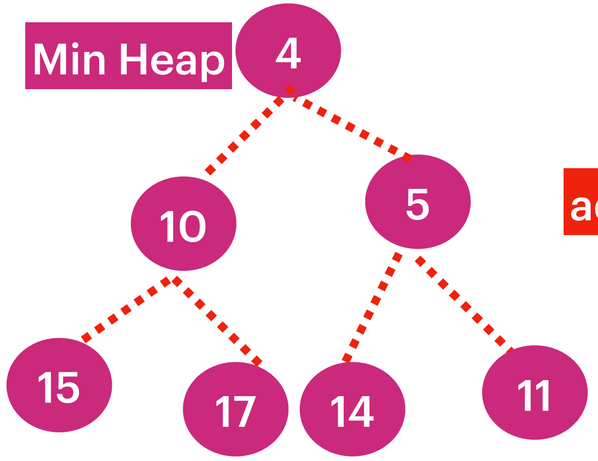
** Each Parent Node should be <= it child node's

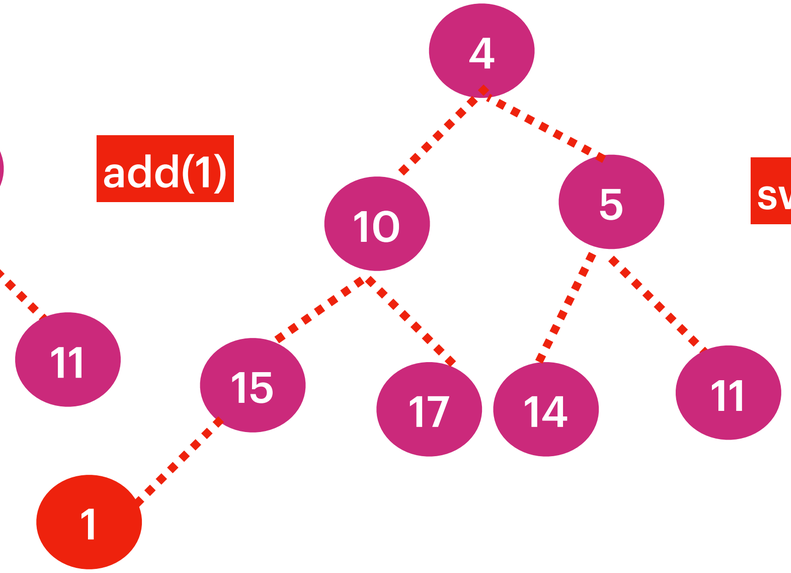
Algorithm :

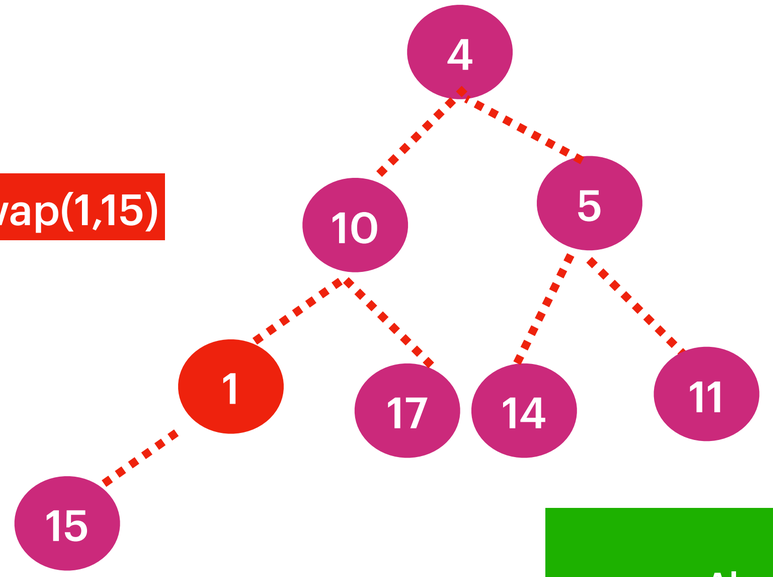
Add to the RightMost position.
If the current element > parent then swap.



Time Complexity : O(logn)
Space Complexity : O(1)

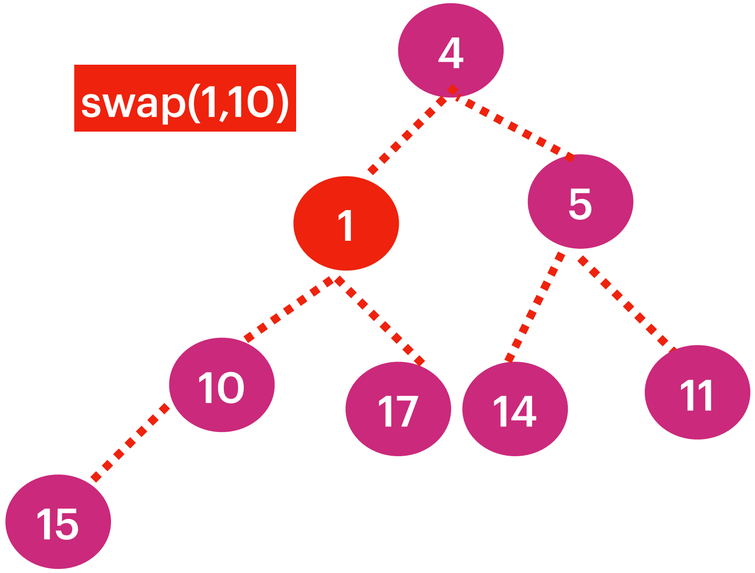


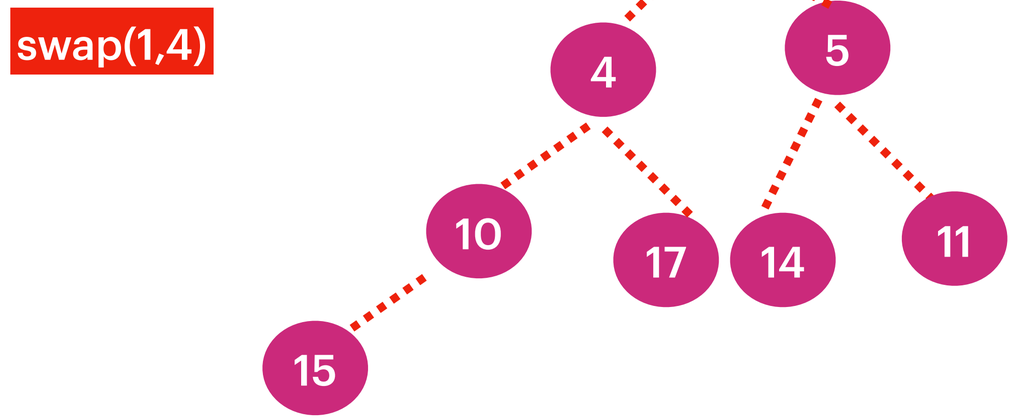




Algorithm :

Add to the RightMost position.
If the current element > parent then swap.

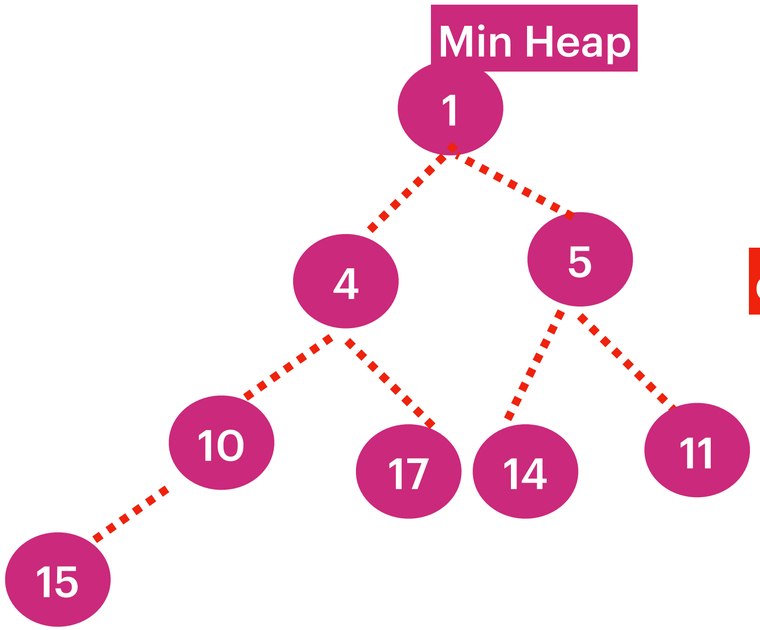




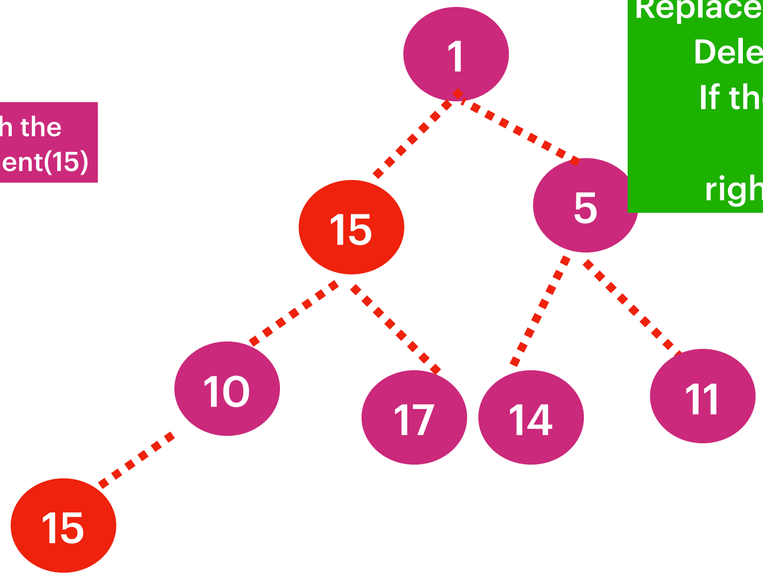
Delete Operation On MinHeap

Algorithm :

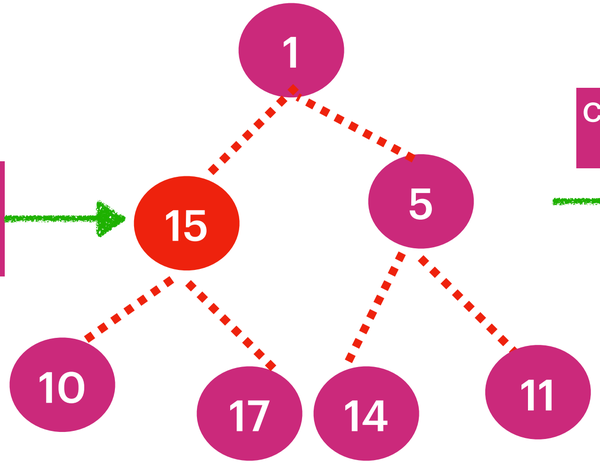
Replace with the right most element ,
Delete the right most element.
If the current Element either > (leftNode || rightNode) swap accordingly



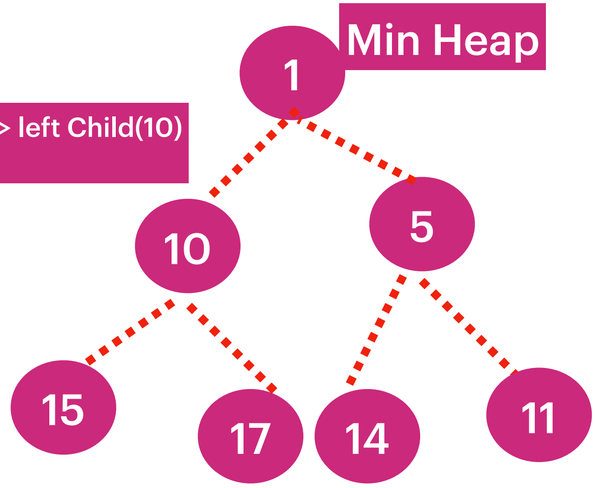
delete(4) → replace(4) with the Right Most Element(15)

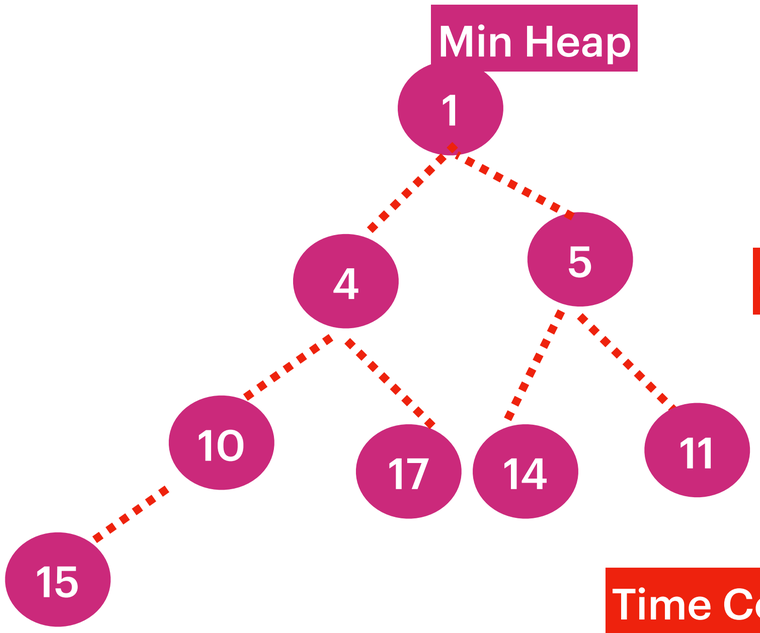


Delete the Right Most Element(15) in the max level of MinHeap



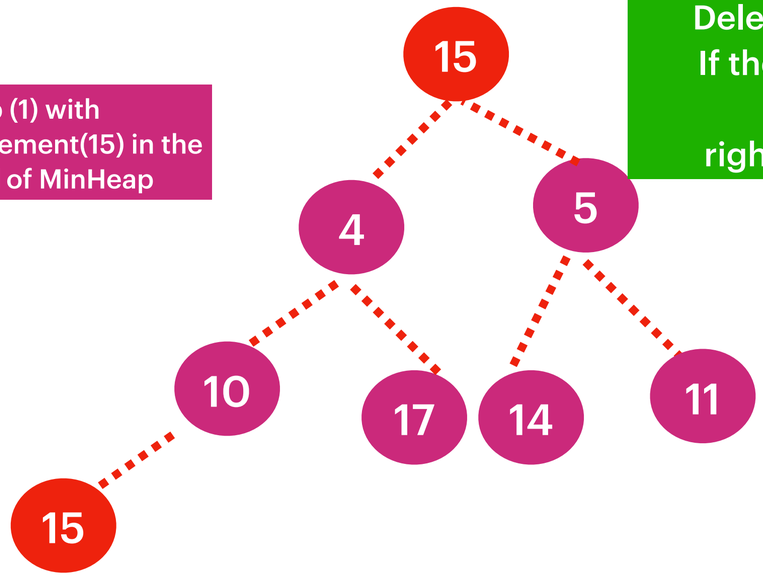
Current Parent(15) which is > left Child(10)
Swap (10,15)





Delete Operation On MinHeap

delete(1) → Swap (1) with Right Most Element(15) in the max level of MinHeap

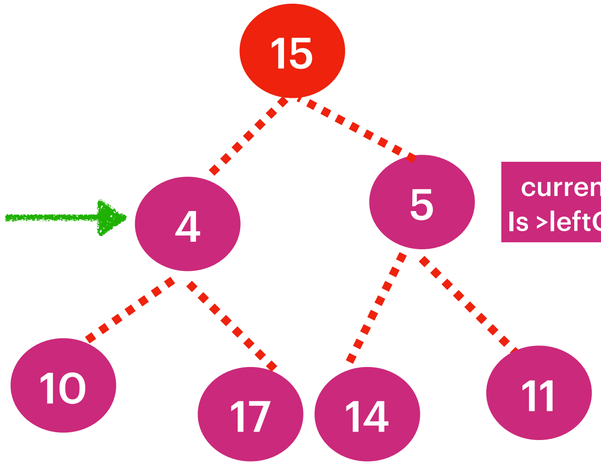


Algorithm :

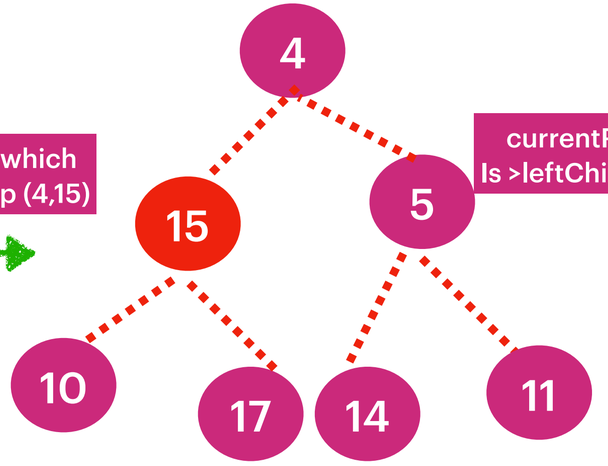
Replace with the right most element ,
Delete the right most element.
If the current Element either > (leftNode || rightNode) swap accordingly

Time Complexity : O(logn)
Space Complexity : O(1)

Delete the Right Most Element(1) in the max level of MinHeap →



currentParent(15) which Is >leftChild(4) swap (4,15) →



currentParent(15) which Is >leftChild(10) swap (10,15) →