DATA STRUCTURES AND ALGORITHMS

SLIDE 5

HEAP

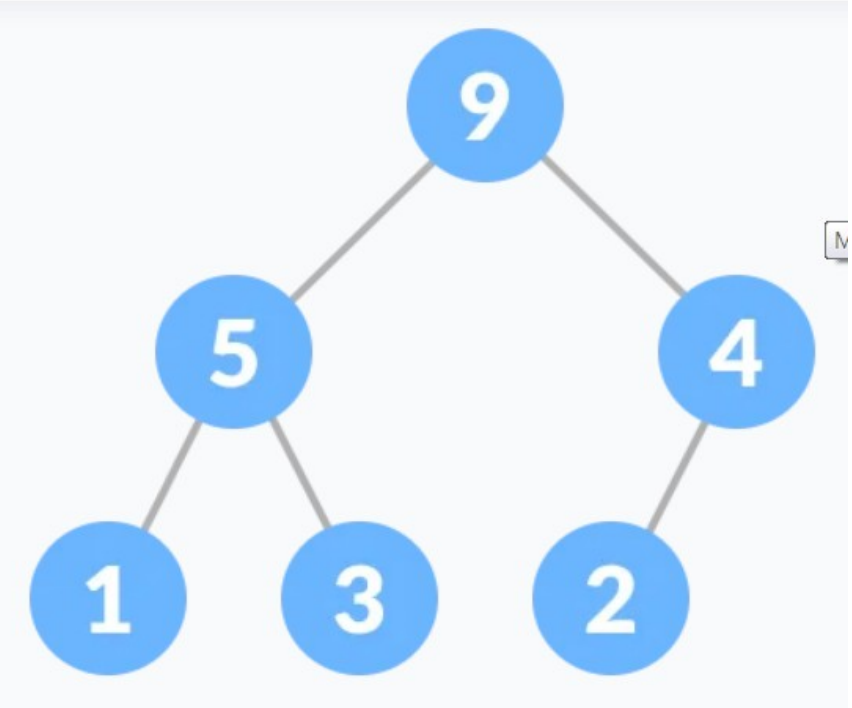
- A binary heap is a binary tree with two other constraints
- 1) Shape(Structural) Property: A binary heap is a complete binary tree, this means all of the levels of the tree are completely filled except possibly the last level. The nodes are filled from left to right.

2) Heap(order) Property: The ordering can be one of two types:

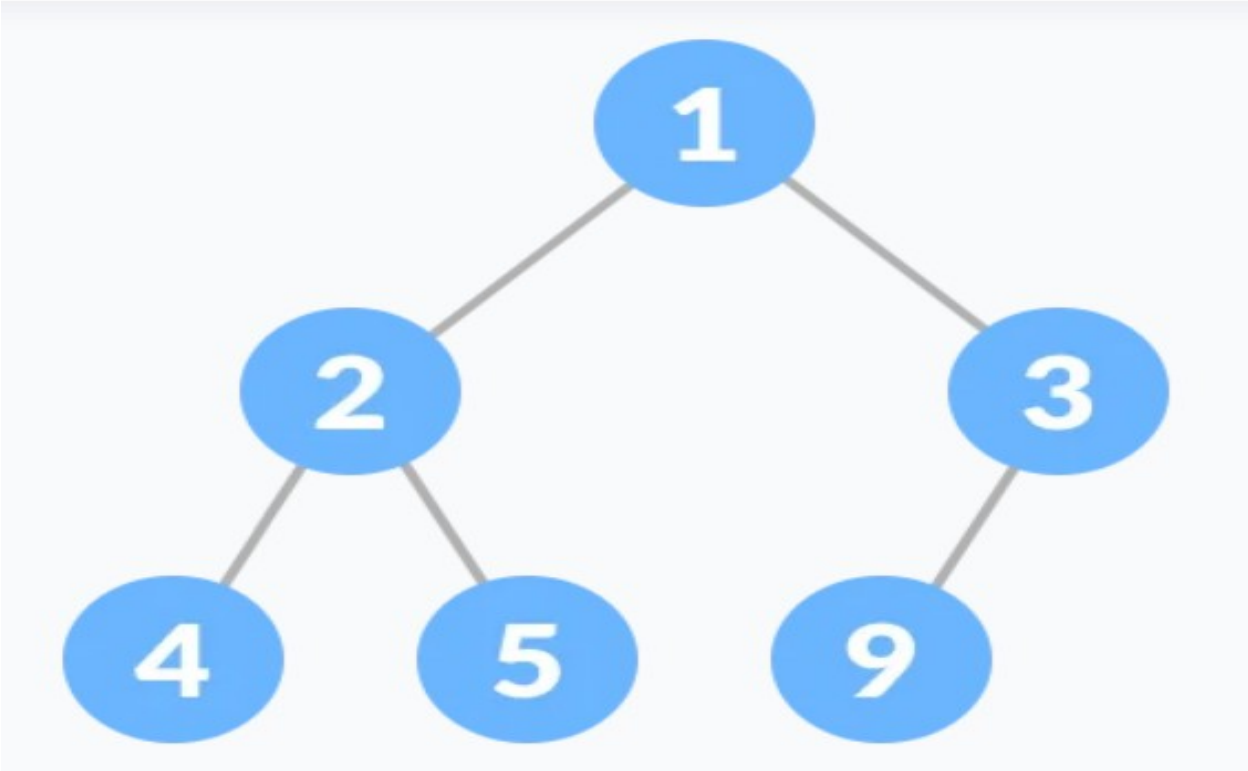
i) min-heap property: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.

ii) max-heap property: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root

Heap Property is the property of a node in which (for max heap) key of each node is always greater than its child node/s and the key of the root node is the largest among all other nodes;

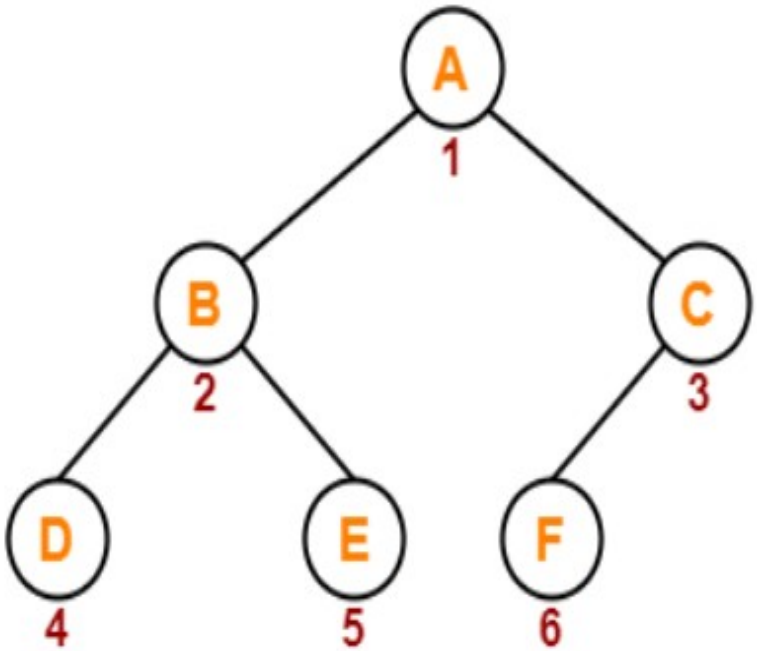


(for min heap) key of each node is always smaller than the child node/s and the key of the root node is the smallest among all other nodes.

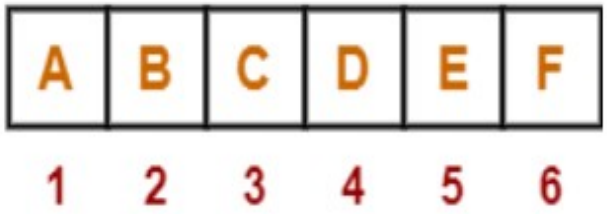


Array Representation of Binary Heap-

A binary heap is typically represented as an array.



Binary Heap



Array Representation

Upheap:

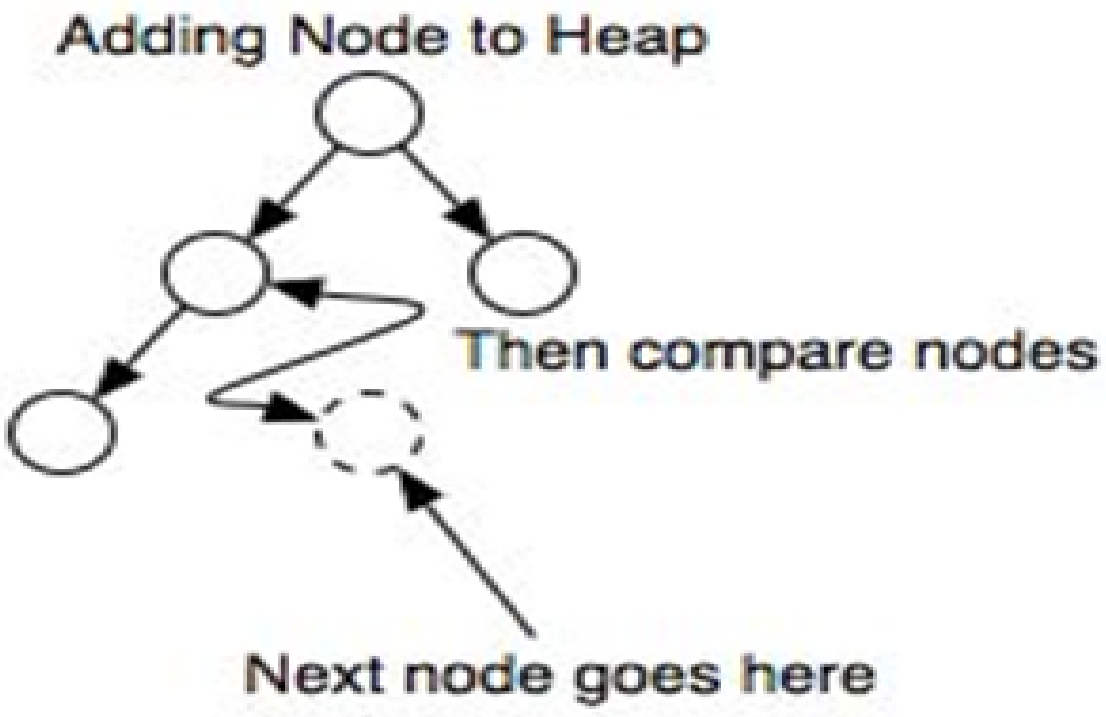
- *The upheap process is used to add a node to a heap. When you upheap a node, you compare its value to its parent node; if its value is less than its parent node, then you switch the two nodes and continue the process. Otherwise the condition is met that the parent node is less than the child node, and so you can stop the process. Once you find a parent node that is less than the node being upheaped, you know that the heap is correct--the node being upheaped is greater than its parent, and its parent is greater than its own parent, all the way up to the root.*

Downheap:

- *The downheap process is similar to the upheaping process. When you downheap a node, you compare its value with its two children. If the node is less than both of its children, it remains in place; otherwise, if it is greater than one or both of its children, then you switch it with the child of lowest value, thereby ensuring that of the three nodes being compared, the new parent node is lowest. Of course, you cannot be assured that the node being downheaped is in its proper position -- it may be greater than one or both of its new children; the downheap process must be repeated until the node is less than both of its children*.

INSERTING A NODE

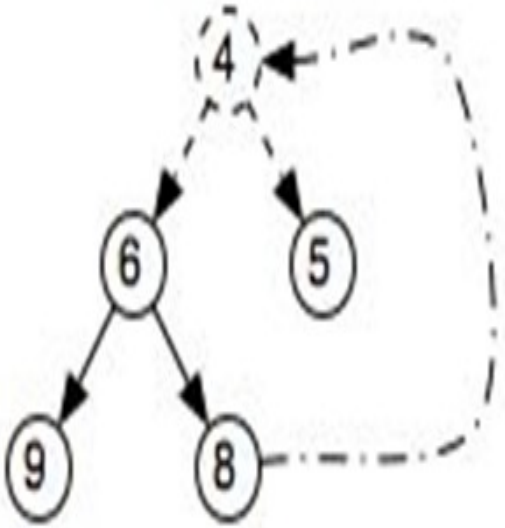
When you add a new node to a heap, you add it to the rightmost unoccupied leaf on the lowest level. Then you upheap that node until it has reached its proper position. In this way, the heap's order is maintained and the heap remains a complete tree.



DELETING A NODE

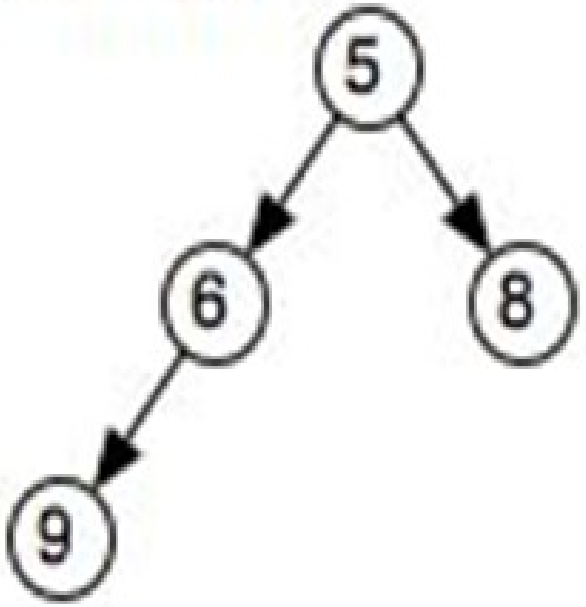
when you take the node out of the tree, you replace it with "last" node in the tree(the node on the last level and rightmost). Once the top node has been replaced, you downheap the node that was moved until it reaches its proper position.

Remove the root node

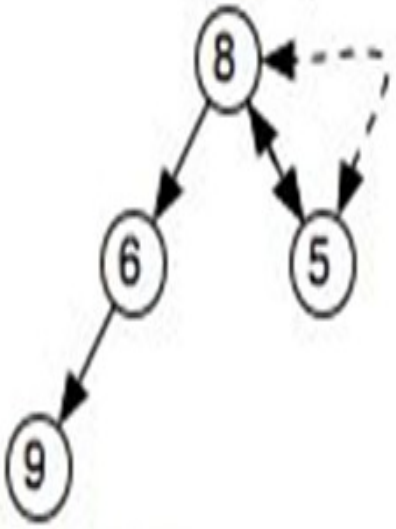


Replace the root node

New Heap



Perform downheap



(Here, 8 and 5 switch)

INSERTING AN ELEMENT INTO A MAX HEAP

Step 1 - Create a new node at the end of heap.

Step 2 - Assign new value to the node.

Step 3 - Compare the value of this child node with its parent.

Step 4 - If value of parent is less than child, then swap them.

Step 5 - Repeat step 3 & 4 until Heap property holds.

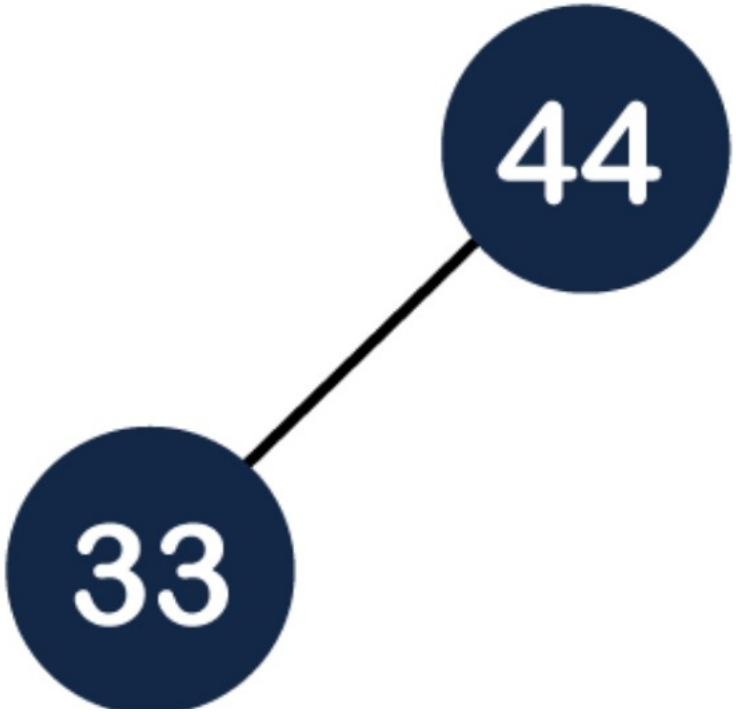
Insertion in the Heap tree

44, 33, 77, 11, 55, 88, 66

Suppose we want to create the max heap tree. To create the max heap tree, we need to consider the following two cases:

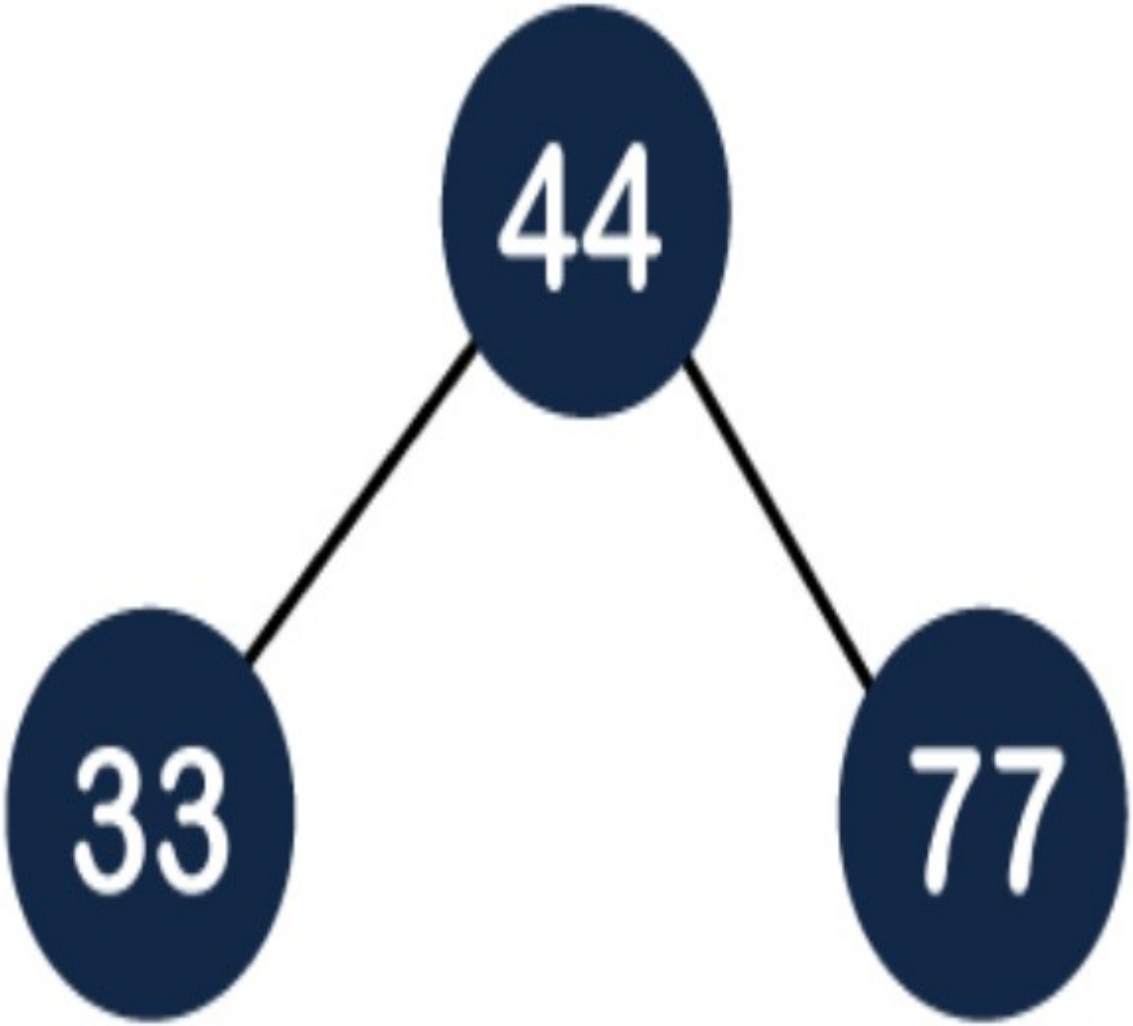
- First, we have to insert the element in such a way that the property of the complete binary tree must be maintained.
- Secondly, the value of the parent node should be greater than the either of its child.

Step 1: First we add the 44 element in the tree as shown below:

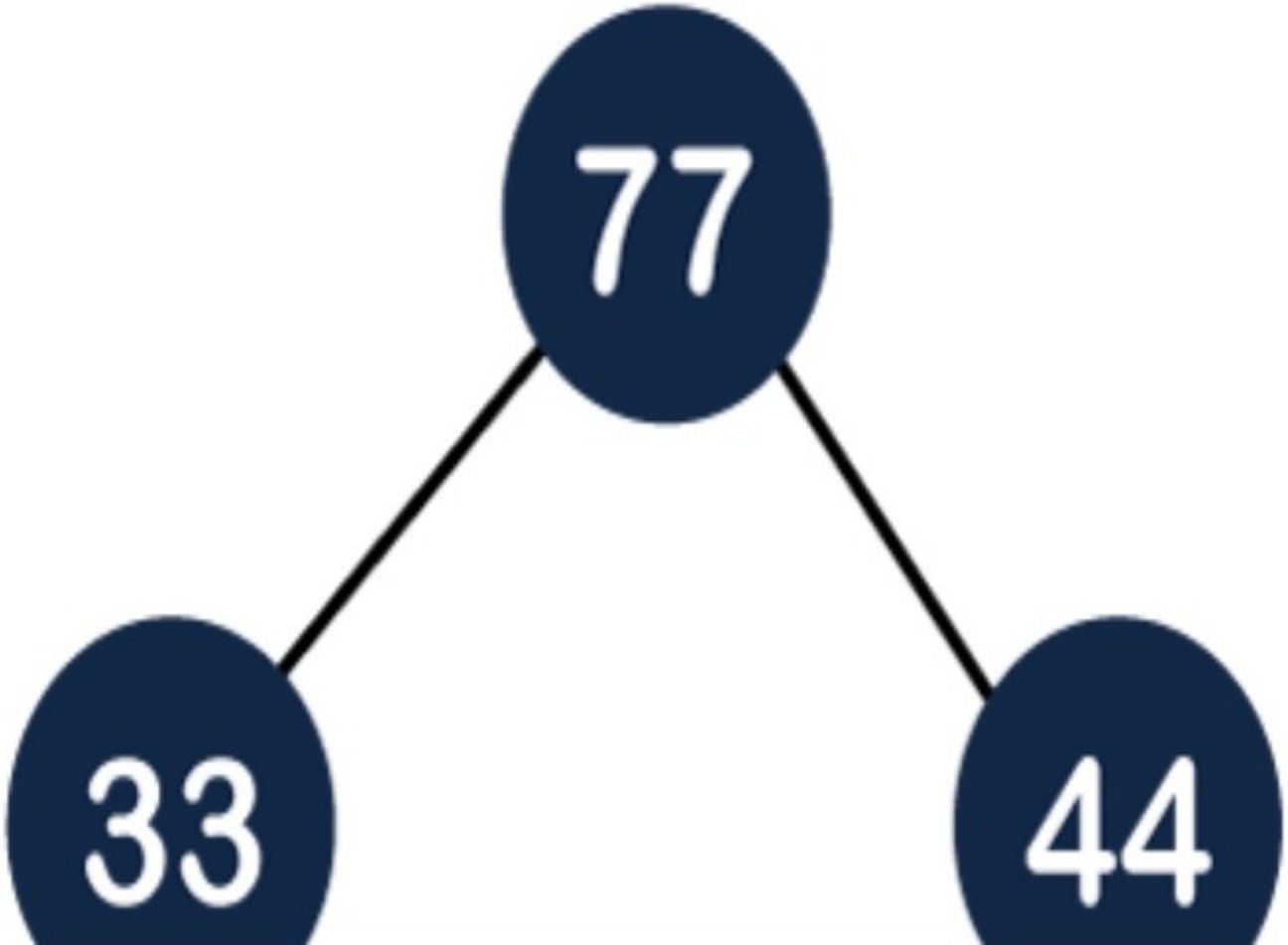


Step 2: The next element is 33. As we know that insertion in the binary tree always starts from the left side so 44 will be added at the left of 33 as shown below:

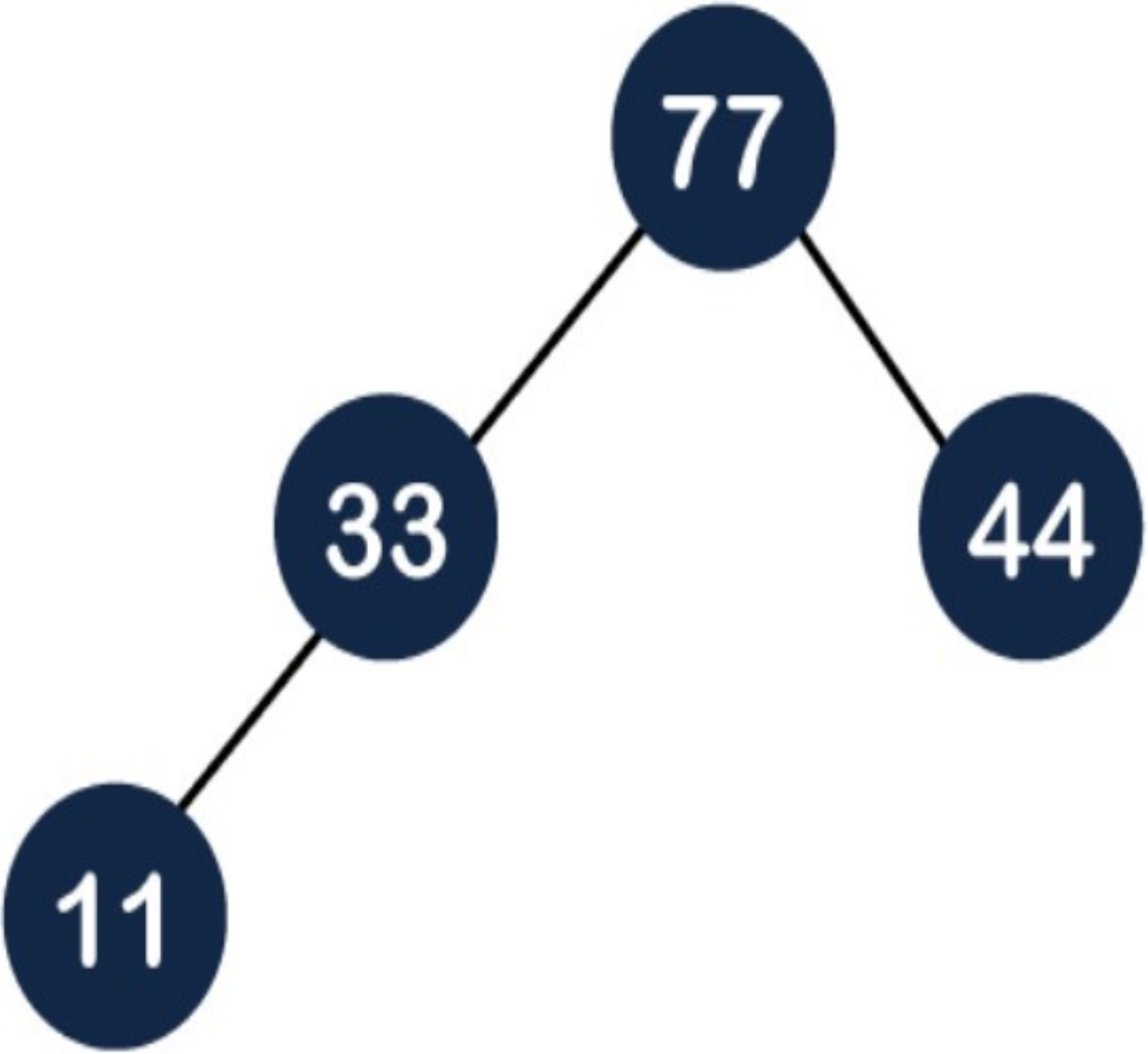
Step 3: The next element is 77 and it will be added to the right of the 44 as shown below:



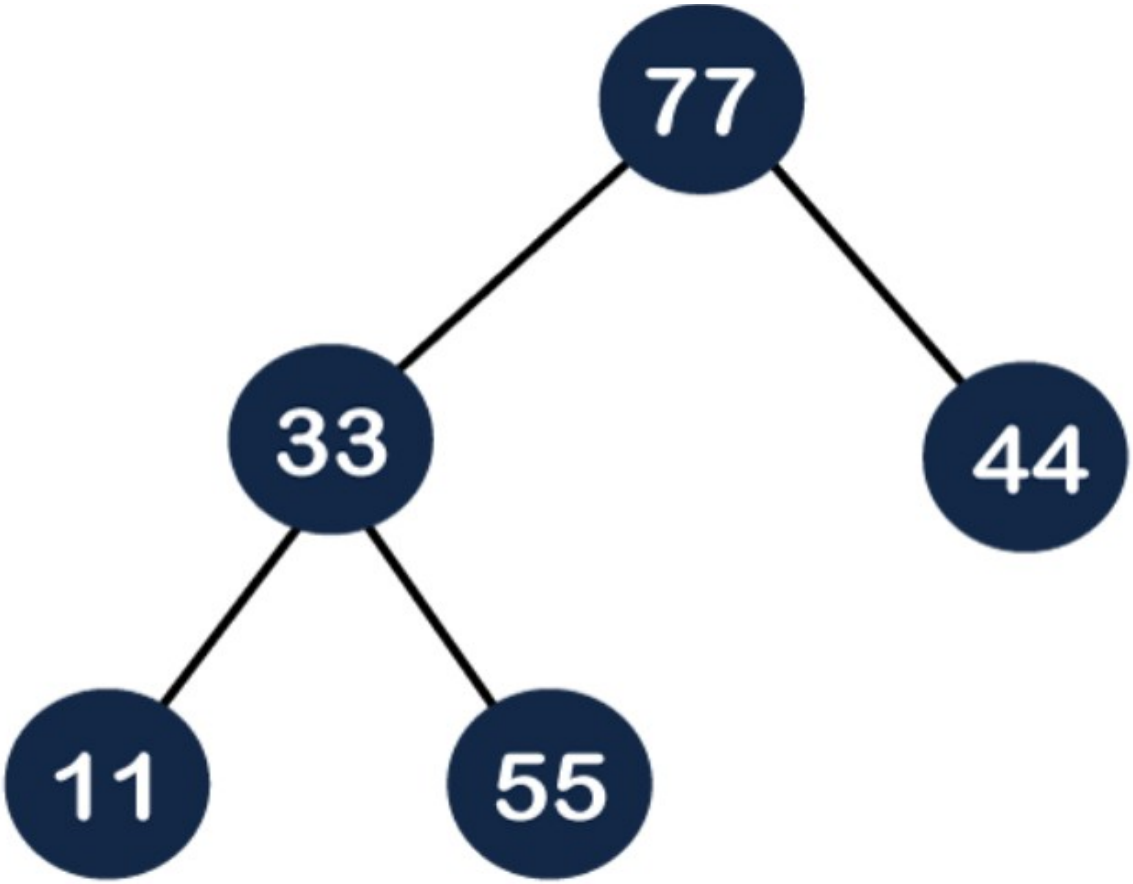
As we can observe in the above tree that it does not satisfy the max heap property, i.e., parent node 44 is less than the child 77. So, we will swap these two values as shown below:



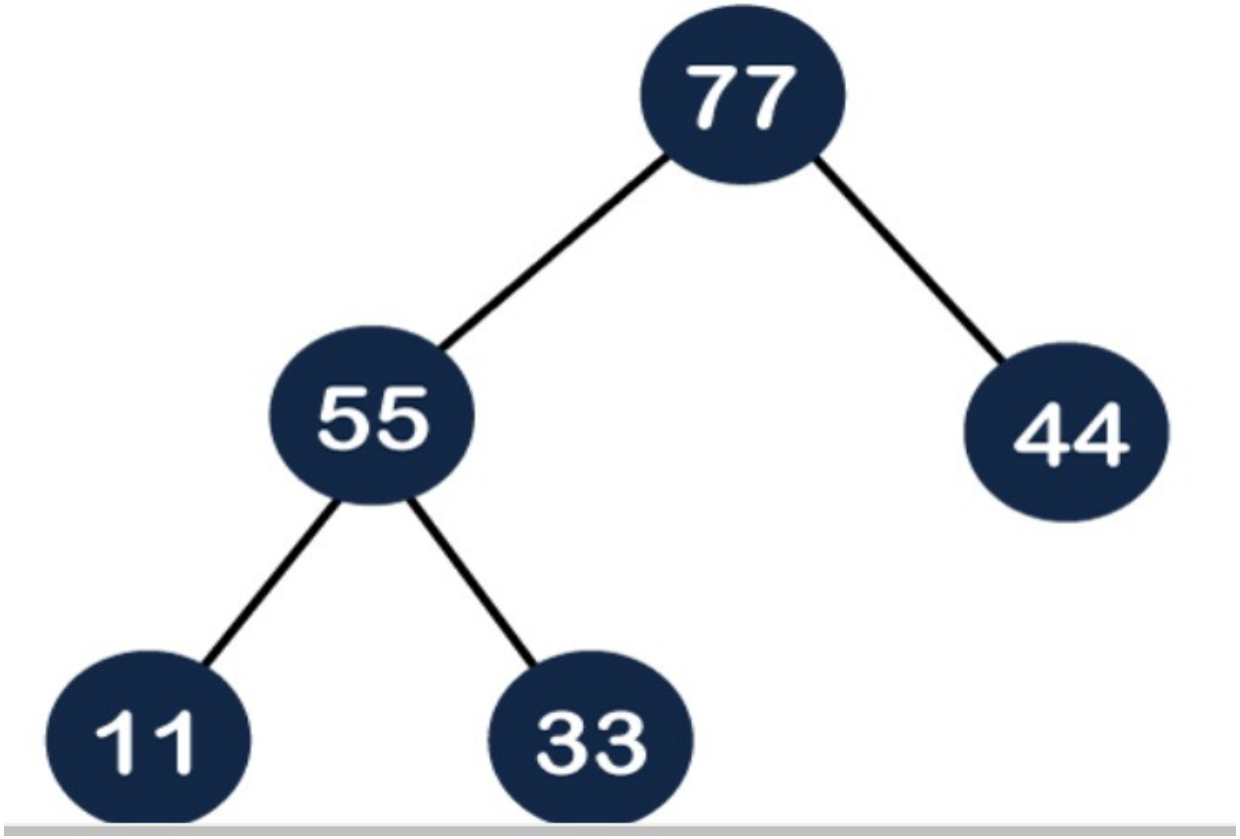
4: The next element is 11. The node 11 is added to the left of 33 as shown below:



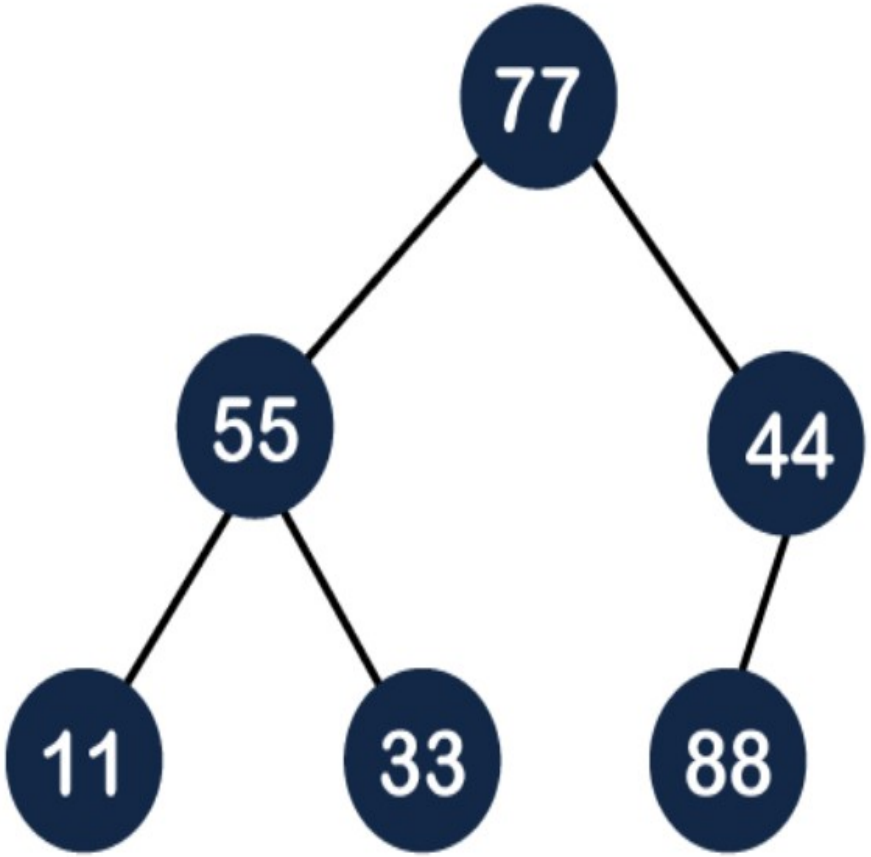
Step 5: *The next element is 55. To make it a complete binary tree, we will add the node 55 to the right of 33 as shown below:*



The above figure does not satisfy the property of the max heap because 33 < 55, so we will swap these two values as shown below:



Step 6: *The next element is 88. The left sub tree is completed so we will add 88 to the left of 44 as shown below:*



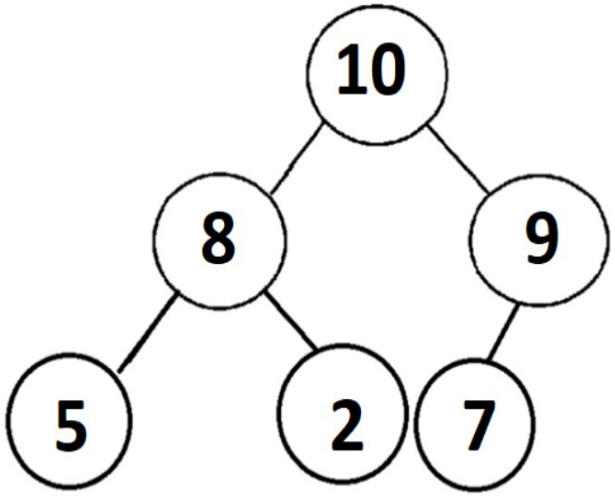
Since 44<88, swap these two values as shown below: Again, it is violating the max heap property because 88>77 so we will swap these two values below:

Step 7: *The next element is 66. To make a complete binary tree, we will add the 66 element to the right side of 77 as shown below*

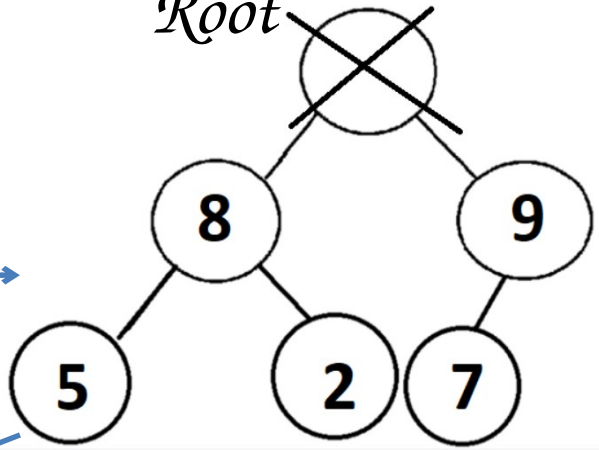
Max Heap Deletion Algorithm

Let us derive an algorithm to delete from max heap. Deletion in Max (or Min) Heap always happens at the root to remove the Maximum (or minimum) value.

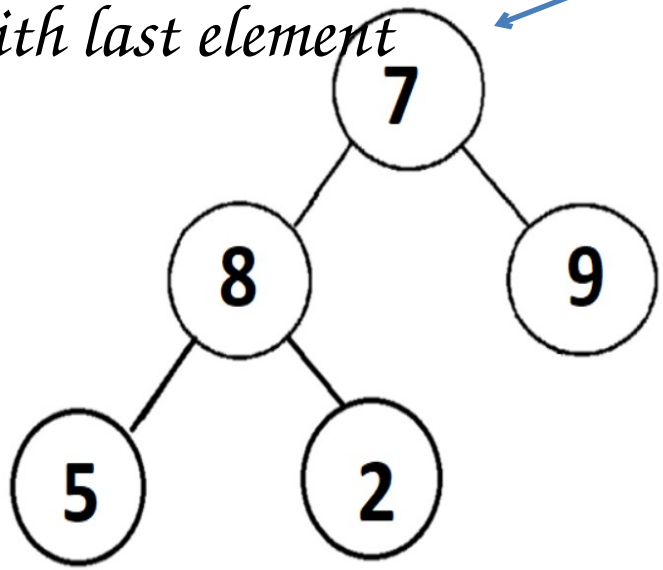
```
Step 1 - Remove root node.
Step 2 - Move the last element of last level to root.
Step 3 - Compare the value of this child node with its parent.
Step 4 - If value of parent is less than child, then swap them.
Step 5 - Repeat step 3 & 4 until Heap property holds.
```



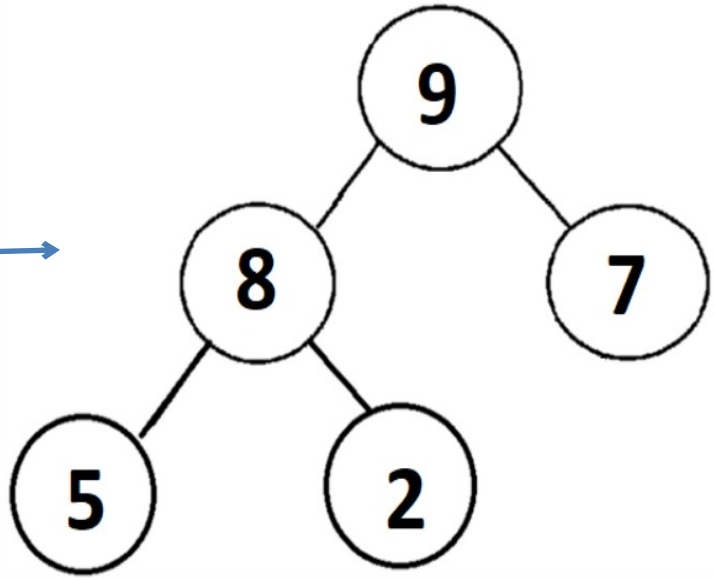
1. Deletete Root



2. Replace Root with last element



3. Heapify



4.Repeat process with other elements

END.

WAIRAGU G.R.