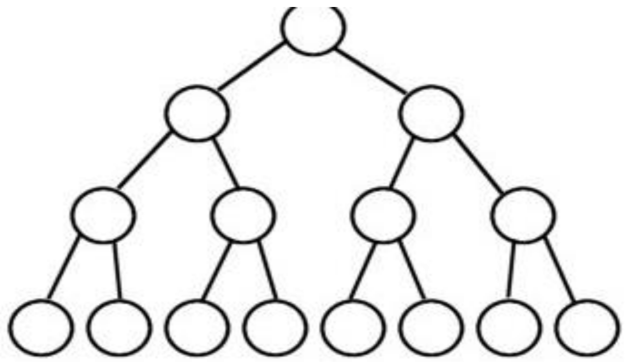
Heap Data Structure

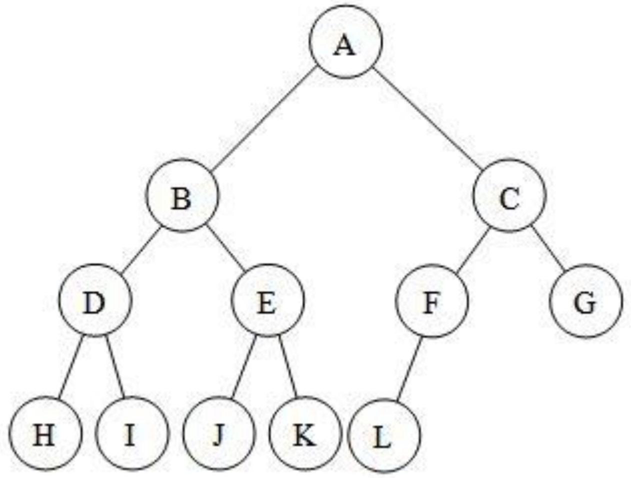
Abdul Mateen
Assistant Professor
PUCIT

Full & Complete Binary Tree

A **full binary tree** (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children:



A **complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.



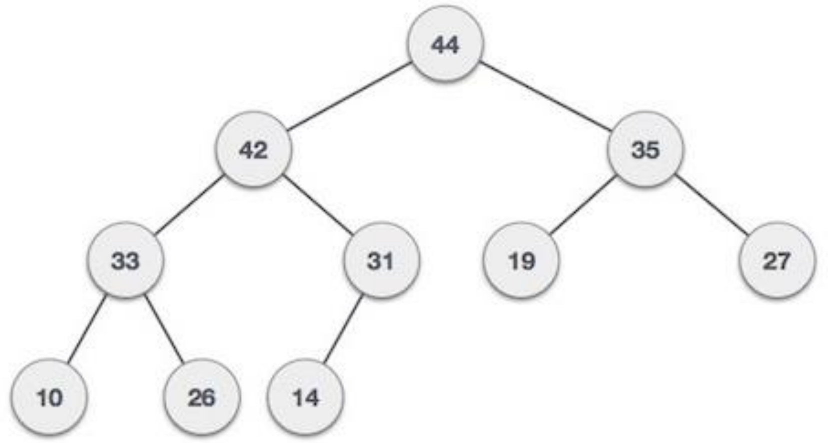
Heap Data Structure

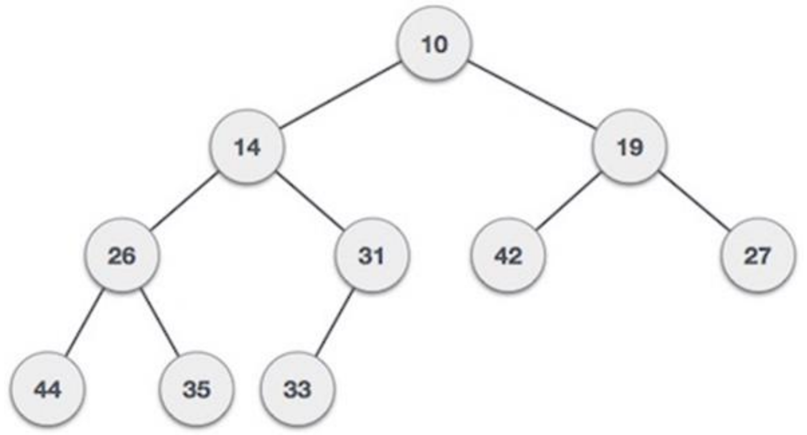
Heap is a special case of balanced binary tree data structure with two properties:

1. It is a complete binary tree
2. Either each node is greater than its descendent nodes (which includes left, right child nodes and their child nodes as well) called **max heap** or each node is smaller than its descendent nodes (which includes left, right child nodes and their child nodes as well) called **min heap.**

The upper right figure represents **max heap**, where root has value 44, where all node in left & right sub-trees have smaller values. The same property exist in every node. For example left child of 44 has value 42, where left and right sub-tree of 42 has values smaller than 42.

The bottom right figure represents **min heap**, where smallest value 10 is on top. Similarly on right of root is 19, where both child of 19 are larger than 19.



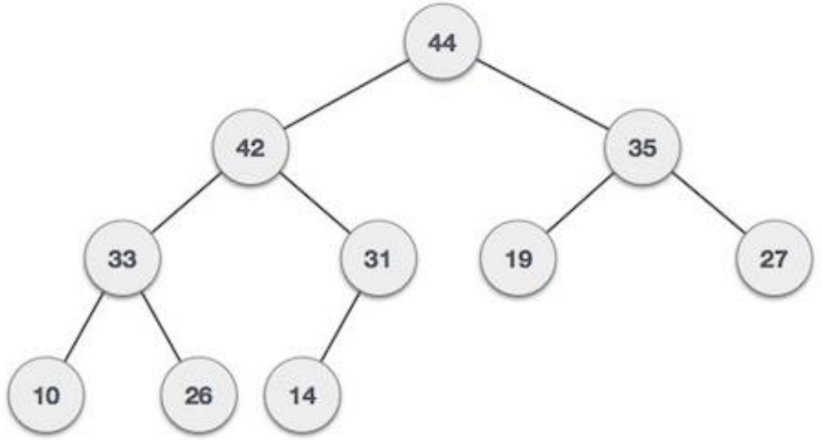


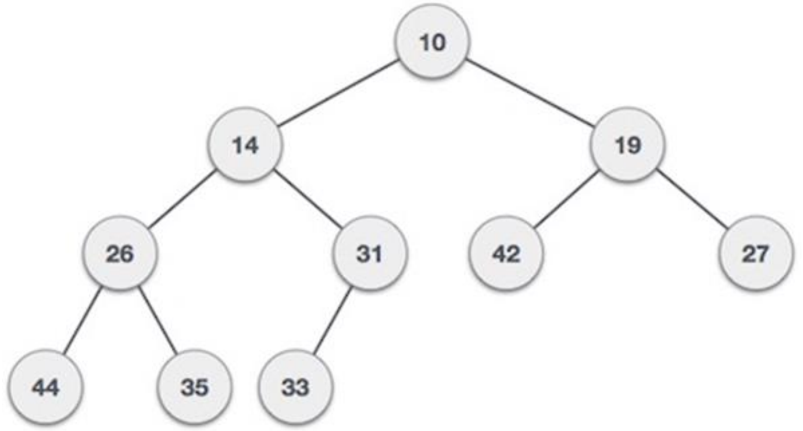
Heap & Sorting

Please, don’t confuse or mix heap with sorting or BST.

In both max and min heap figures, you should notice that there is no ordering, no distinction with left or right child (either of them can be larger than other). Only two properties are kept in heap. One it is always **complete binary tree**, second it keep property of max or min in all nodes (Comparing with it’s sub-tree).

Due to the complete binary tree property of heap, it is possible to implement heap using **arrays** because height is guaranteed to be lgn and there is no rotation required in heap, therefore heap has neither space inefficiency nor it has any time inefficiency.





Heap Operations

Heaps are used to implement **priority** queue, therefore normally we discuss two main operations in heaps i.e. insertion and deletion.

Insertion operation insert new value in last available index as heap is complete binary tree, which means all index will be filled except left most indexes. After placing value, the value is compared with parent node, if value satisfies min or max heap (in case of max heap, new value should be smaller than parent and in min heap, new value should be larger than parent), otherwise swap operation is performed and new value is shifted up at parent. Again parent is check for heap property with grand parent and swapped if required. This process continues till root node, in worst case.

Delete operation always delete root node that has highest priority in either max or min heap. After deletion last value is placed at root node, size is decreased by one. Now, again comes the heap property. Root node is compared with max or min child (in case of max heap, max(left, right) is selected and in case of min heap, min(left, right) is selected). If heap property not satisfy (which in general does not satisfy at root but in child nodes, it may), again swap operation is performed. Again property check applied and process may repeat till leaf node in worst case.

In next slides, we will elaborate both operations in steps with example:

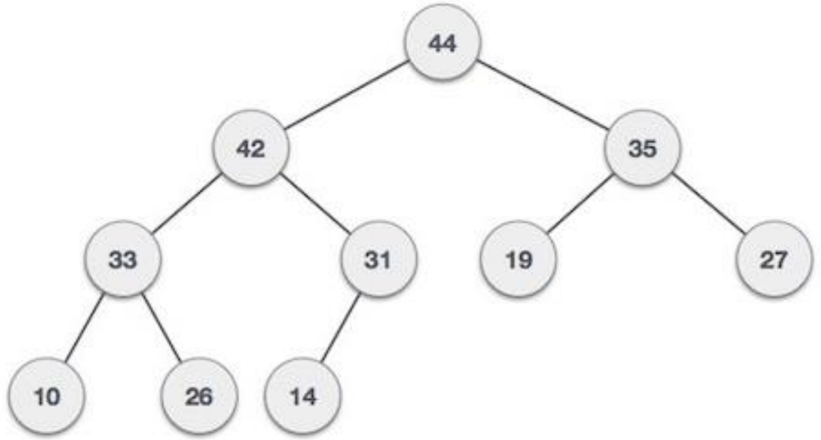
Insert Operation

Consider max heap.

Insert 15.

Insert 41

Insert 56



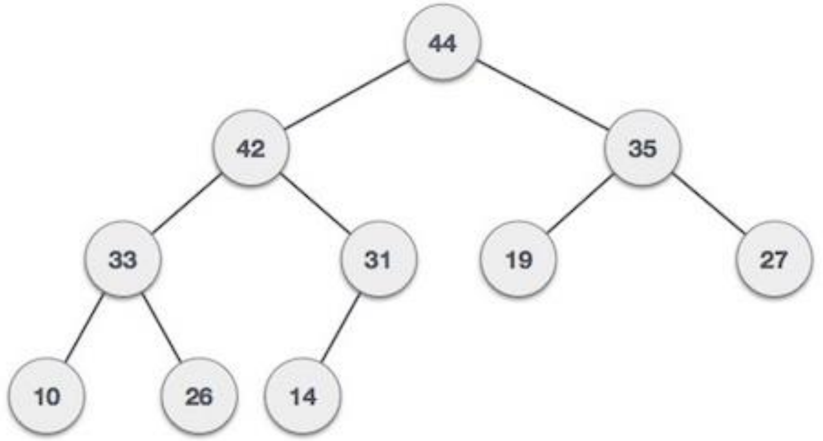
Insert Operation Steps

Normally heap has fixed size, depending on the capacity of system. For example, number of print jobs possible in print queue of printer, number of processes executing concurrently etc. To insert a new value following steps are required (consider max heap):

- Assign new value at the end (according to the current size)
- Compare the current value with its parent
- If value of parent is less than child, then swap them
- Repeat previous two steps until heap property holds or root node is compared
- Finally increase current size by 1

Delete Operation

Consider max heap & apply delete operation at least 3 times.



Delete Operation Steps

As discussed earlier that normally root value is deleted (having highest priority), otherwise deleting a node other than root means leaving the system due to any reason, which is possible but not currently in our scope. To delete root following steps are required (consider max heap):

- Save root value in some variable
- Copy last value at root
- Decrease size by 1
- Select max value from left and right child and compare with current node (at start it is root node)
- If value of parent is less than child, then swap them
- Repeat previous two steps until heap property holds or leaf node is reached