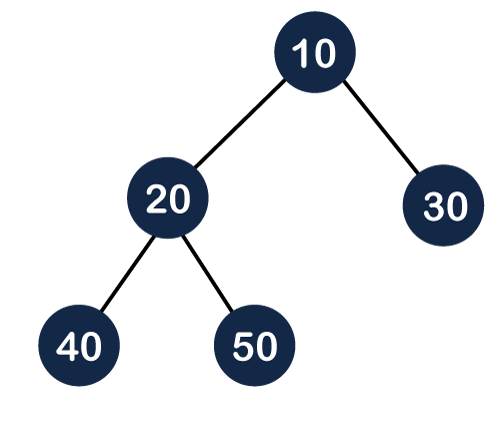
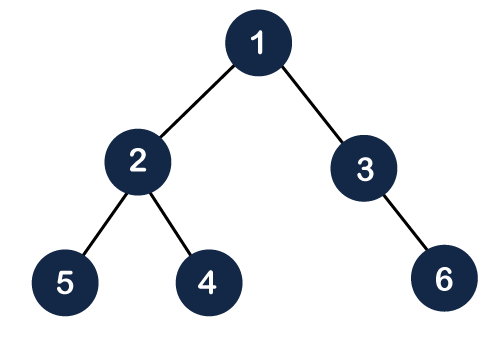
# 

A heap is a complete binary tree, and the binary tree is a tree in which the node can have the utmost two children. Before knowing more about the heap [data structure](https://www.javatpoint.com/data-structure-tutorial), we should know about the complete binary tree.

### What is a complete binary tree?

A complete binary tree is a [binary tree](https://www.javatpoint.com/binary-tree) in which all the levels except the last level, i.e., leaf node should be completely filled, and all the nodes should be left justified.

**Let's understand through an example.**

****

In the above figure, we can observe that all the internal nodes are completely filled except the leaf node; therefore, we can say that the above tree is a complete binary tree.

The above figure shows that all the internal nodes are completely filled except the leaf node, but the leaf nodes are added at the right part; therefore, the above tree is not a complete binary tree.

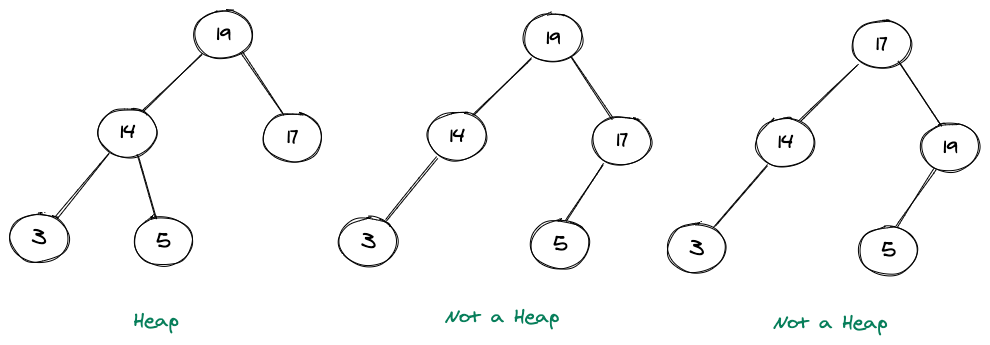
#### Note: The heap tree is a special balanced binary tree data structure where the root node is compared with its children and arrange accordingly.

A Heap is a special type of tree that follows two properties. These properties are :

* All leaves must be at h or h-1 levels for some h > 0(complete binary tree property).
* The value of the node must be >= (or <=) the values of its children nodes, known as the heap property.

Consider the pictorial representation shown below:

In the pictures shown above, the leftmost tree denotes a heap (Max Heap) and the two tree to its right aren't heap as the middle tree violates the first heap property(not a complete binary tree) and the last tree from the left violates the second heap property(17 < 19).



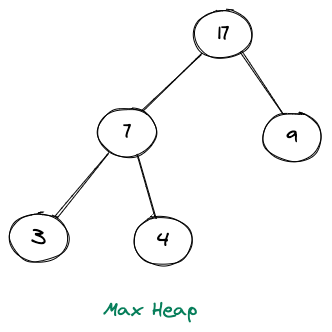
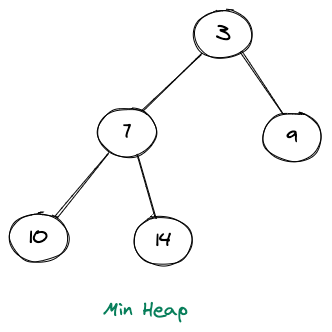
## Types of Heap

If we consider the properties of a heap, then we can have two types of heaps. These mainly are:

### Min Heap

In this heap, the value of a node must be less than or equal to the values of its children nodes.

It can be clearly seen that the value of any node in the above heap is always less than the value of its children nodes.



### Max Heap

In this heap, the value of a node must be greater than or equal to the values of its children nodes.

It can be clearly seen that the value of any node in the above heap is always greater than the value of its children nodes.

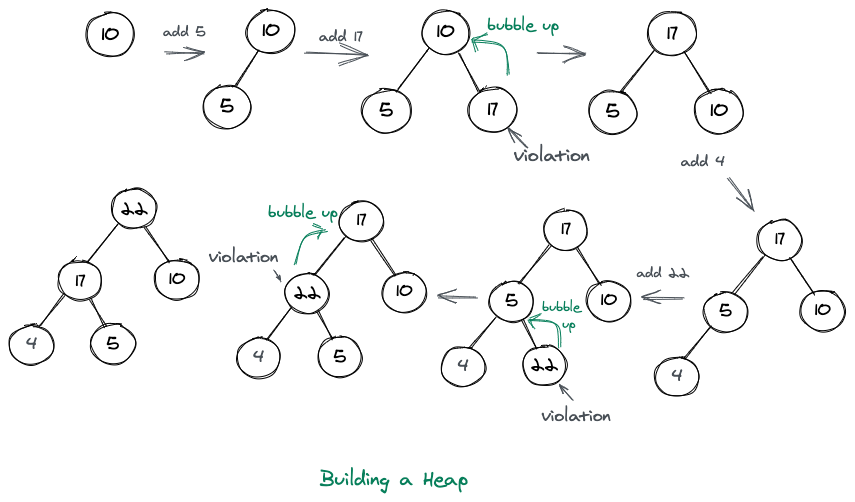
## Building a Heap

Let's look at some operations like inserting an element in a heap or deleting an element from a heap.

### 1. Inserting an Element :

* First, increase the heap size by 1.
* Then insert the new element at the available position(leftmost position ? Last level).
* Heapify the element from the bottom to the top(bubble up).

Building a heap includes adding elements into the heap. Let us consider an array of elements, namely nums = [10,5,17,4,22]. We want to make a Max Heap out of these elements, and the way we do that is shown in the pictorial representation below.

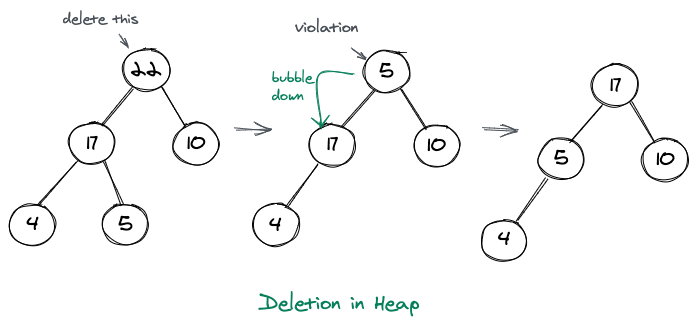


### 2. Deleting an Element :

* Copy the first element of the heap(root) into some variable
* Place the last element of the heap in the root's position
* Bubble Down to make it a valid heap

Whenever we are deleting an element, we simply delete the root element and replace it with the last element of the heap.

Consider the pictorial representation shown below:



## Heap: Time Complexity Analysis

### 1. Inserting an Element: the worst-case time complexity of Inserting an element in a binary heap is: O(log N)

### 2. Deleting an Element: the time complexity of deleting the node from the binary heap thus, in turn, is: O(log N).

### 3. Get Min/Max Element: The time complexity of extracting Min/Max is O(1).