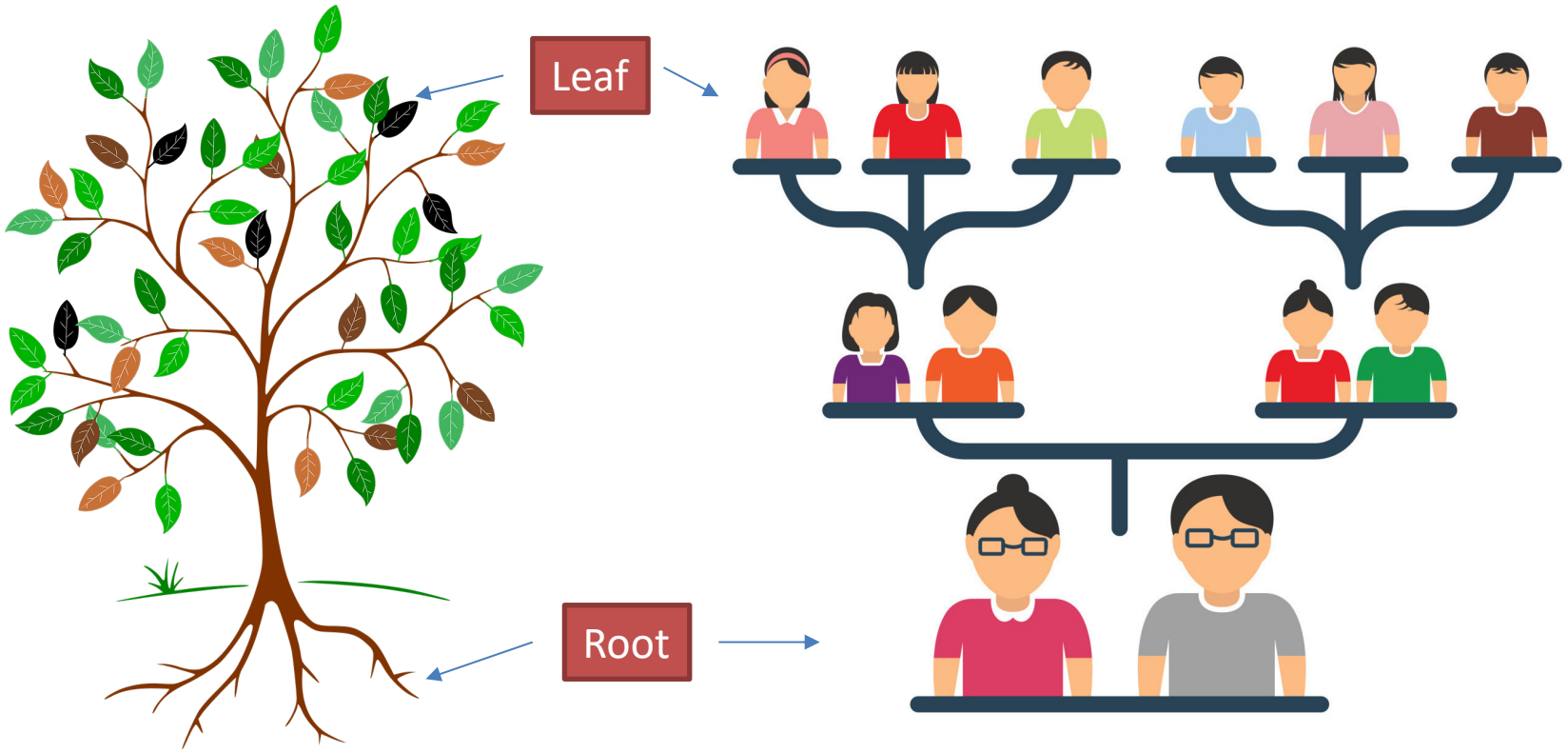
Tree

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.

Suppose you have to store all the records of your family hierarchy, there you need tree data structure.



Tree Terminologies

incapp

Node

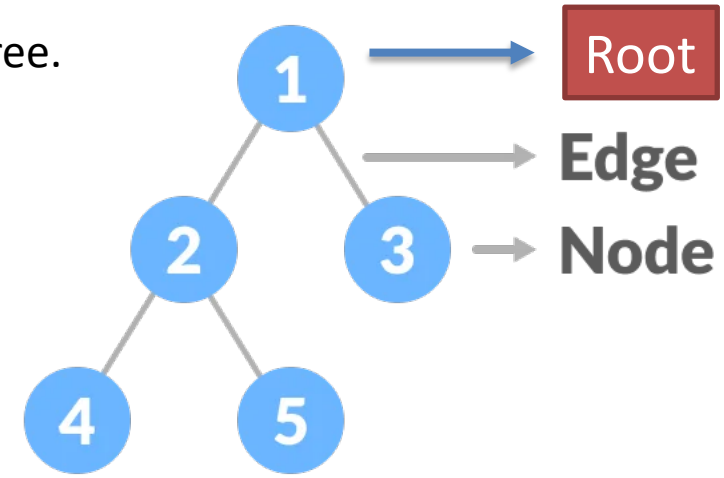
A node is an entity that contains a key or value and link to its child nodes.
The last nodes of each path are called **leaf nodes or external nodes** that do not contain a link/pointer to child nodes.
The node having at least a child node is called an **internal node**.

Edge

It is the link between any two nodes.

Root

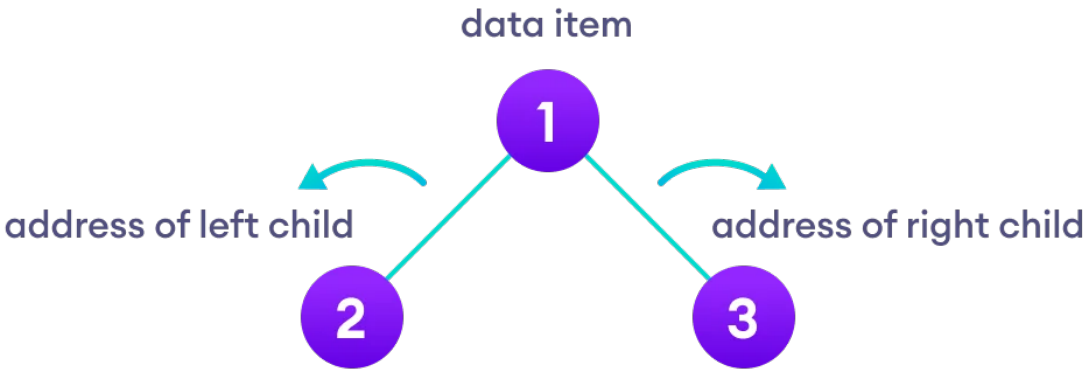
It is the topmost node of a tree.



Binary Tree

A binary tree is a tree data structure in which each parent node can have at most two children. Each node of a binary tree consists of three items:

- data item
- address of left child
- address of right child



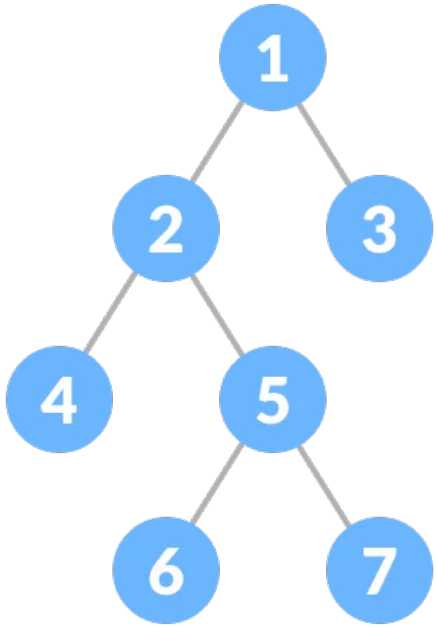
Types of Binary Tree

incapp

- Full Binary Tree
- Perfect Binary Tree
- Complete Binary Tree
- Degenerate or Pathological Tree
- Skewed Binary Tree
- Balanced Binary Tree

Full Binary Tree

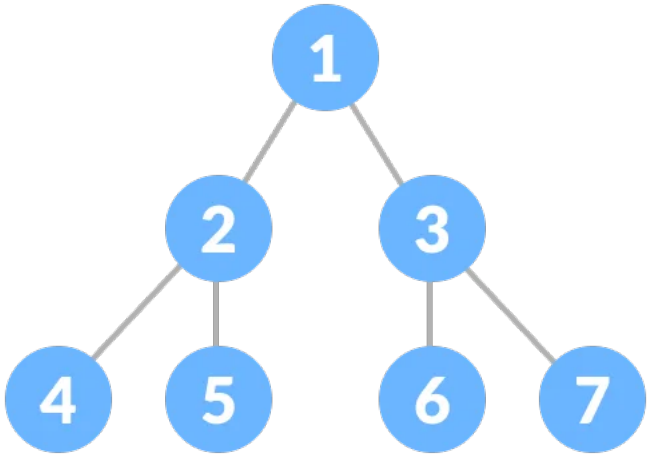
A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.



Perfect Binary Tree

incapp

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.

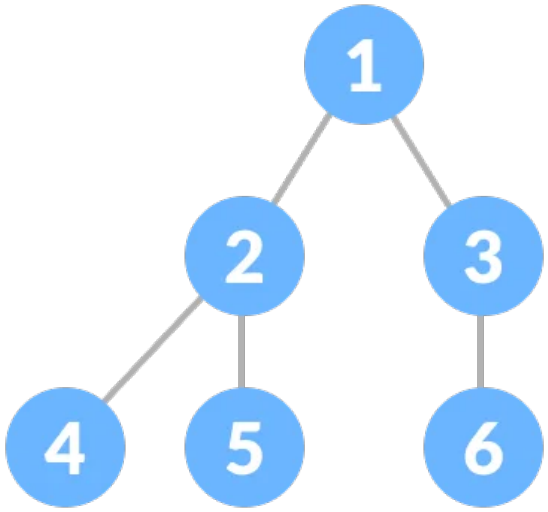


Complete Binary Tree

incapp

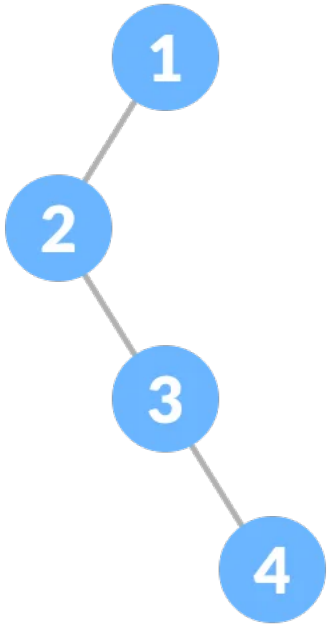
A complete binary tree is just like a full binary tree, but with two major differences

- Every level must be completely filled
- All the leaf elements must lean towards the left.
- The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.



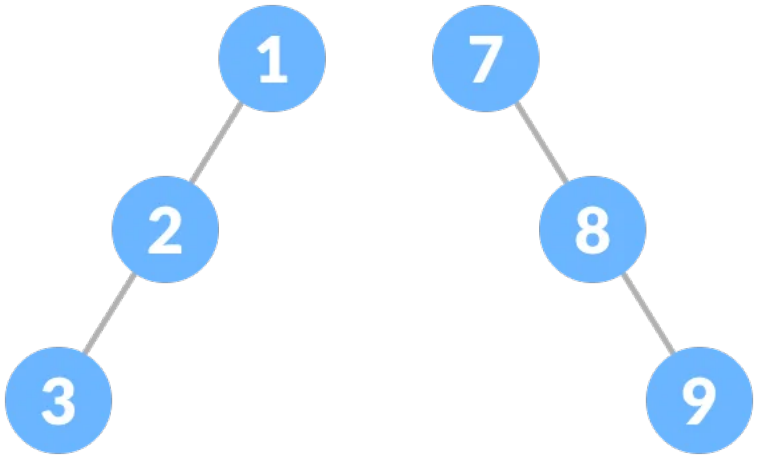
Degenerate or Pathological Binary Tree

A degenerate or pathological tree is the tree having a single child either left or right.



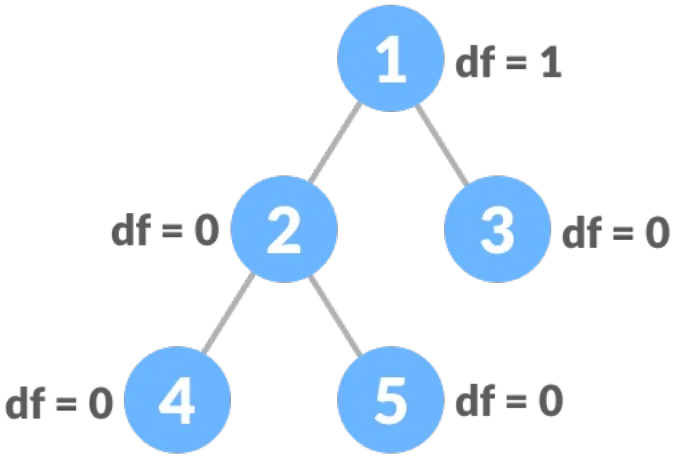
Skewed Binary Tree

A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: **left-skewed binary tree** and **right-skewed binary tree**.



Balanced Binary Tree

It is a type of binary tree in which the difference between the height of the left and the right subtree for each node is either 0 or 1.



Binary Tree Traversals

incapp

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree.

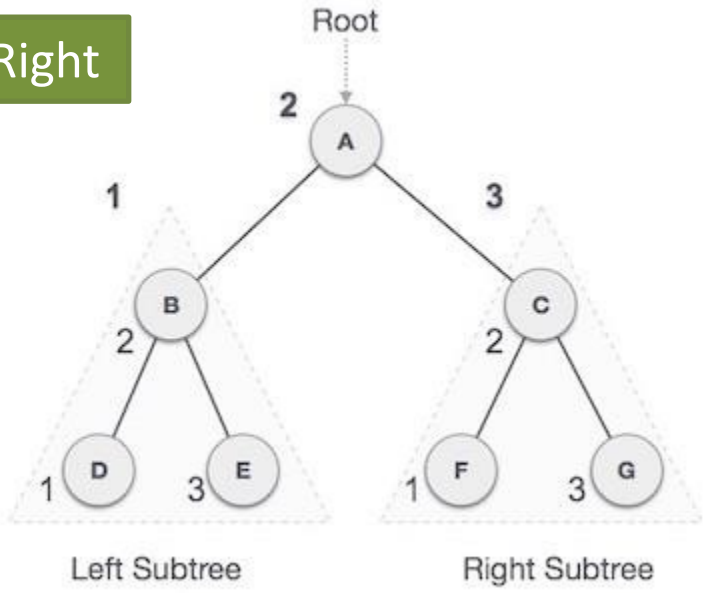
There are three ways which we use to traverse a tree:

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

In-Order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree.

Left > **Root** > Right



We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** is also traversed in-order. The process goes on until all the nodes are visited.

In-order traversal: **D** → **B** → **E** → **A** → **F** → **C** → **G**

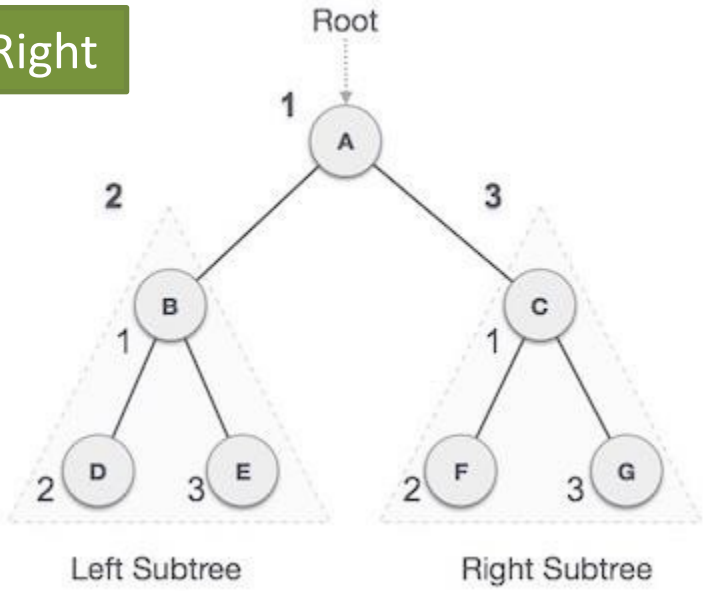
Algo for In-Order Traversal

Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Visit root node.
Step 3 – Recursively traverse right subtree.

Pre-Order Binary Tree Traversals

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Root > Left > Right



We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited.

Pre-order traversal: **A** → **B** → **D** → **E** → **C** → **F** → **G**

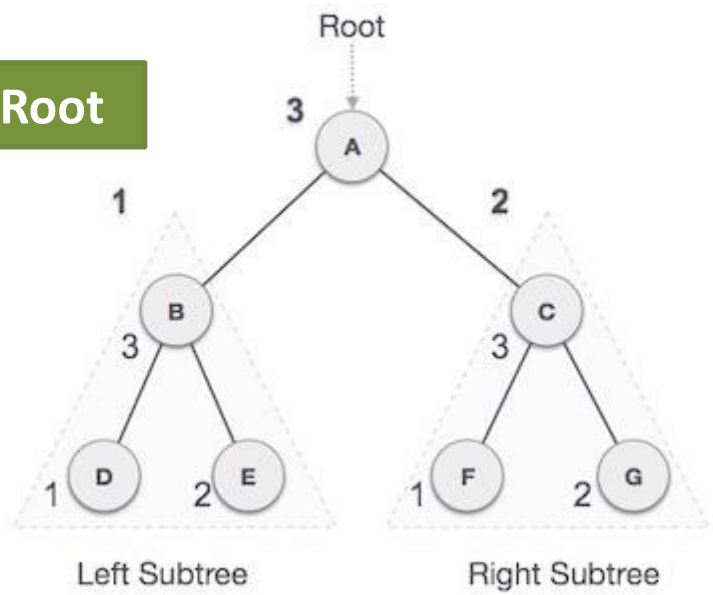
Algo for Pre-Order Traversal

Until all nodes are traversed –
Step 1 – Visit root node.
Step 2 – Recursively traverse left subtree.
Step 3 – Recursively traverse right subtree.

Post-Order Binary Tree Traversals

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

Left > Right > **Root**



We start from **A**, and following pre-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited.

Post-order traversal: **D** → **E** → **B** → **F** → **G** → **C** → **A**

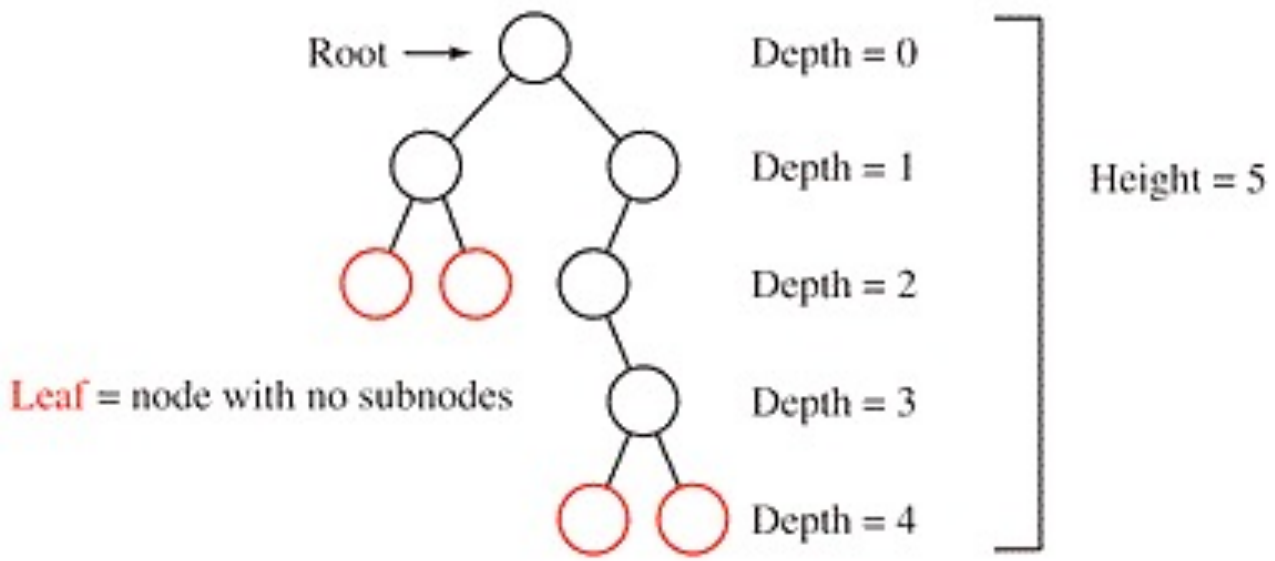
Algo for Post-Order Traversal

Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Recursively traverse right subtree.
Step 3 – Visit root node.

Height & Depth of Binary Tree

A node's **height is the number of edges to its most distant leaf node.** On the other hand, a **node's depth is the number of edges back up to the root**.

Depth of the tree = **Height** of the tree - 1



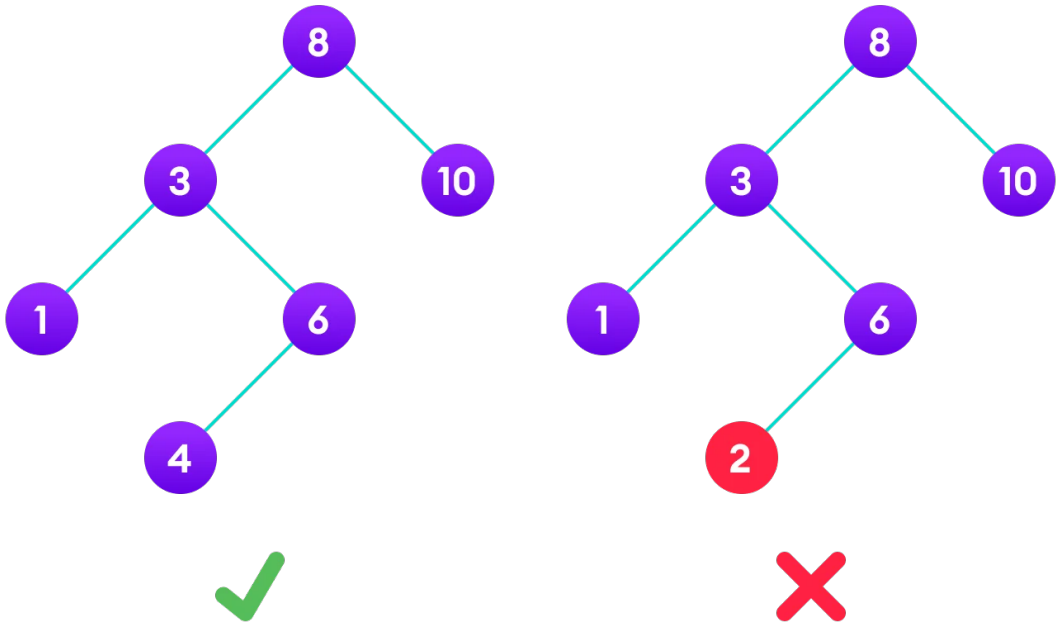
Algo for Height of Binary Tree

```
1.  FindHeight( Node root)
2.          If root=NULL
3.                  return 0
4.          Else
5.                  int l=FindHeight (root->left)
6.                  int r=FindHeight(root->right)
7.                  Return max(l,r)+1
8.          [End If]
9.  [End]
```

Binary Search Tree (BST)

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than or equal to its parent node's key.
- Both subtrees of each node are also BSTs i.e. they have the above two properties



Binary Tree (BT) vs Binary Search Tree (BST)

incapp

Binary Tree	Binary Search Tree
Elements in this tree are in random order.	Elements in this tree are in a fixed order. Its left child should have a value less than the parent node, and the right child should have a value greater than the parent value.
Operations like searching, inserting, and deletion take longer, i.e., O(n) time to execute because elements are not sorted.	Operations like searching, inserting, and deletion take shorter, i.e., O(log n) time to execute because elements are sorted.
Types of binary trees: Full Binary Tree, Complete Binary Tree, Perfect Binary Tree, and Extended Binary Tree etc.	Types of binary search trees: AVL trees, splay trees, and Tango trees etc.

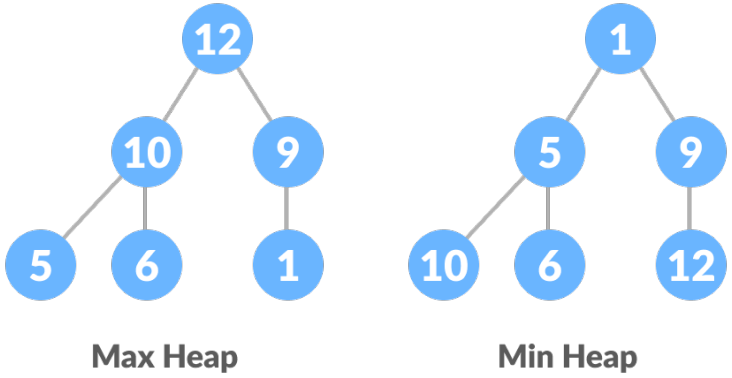
Heap Sort

Heap Sort is a comparison-based sorting algorithm that uses a binary heap data structure to sort elements.

Heap Data Structure

Heap is a special tree-based data structure. A binary tree is said to follow a heap data structure if,

- It is a complete binary tree.
- All nodes in the tree follow the property that they are greater than their children i.e. the largest element is at the root and both its children and smaller than the root and so on. Such a heap is called a max-heap. If instead, all nodes are smaller than their children, it is called a min-heap

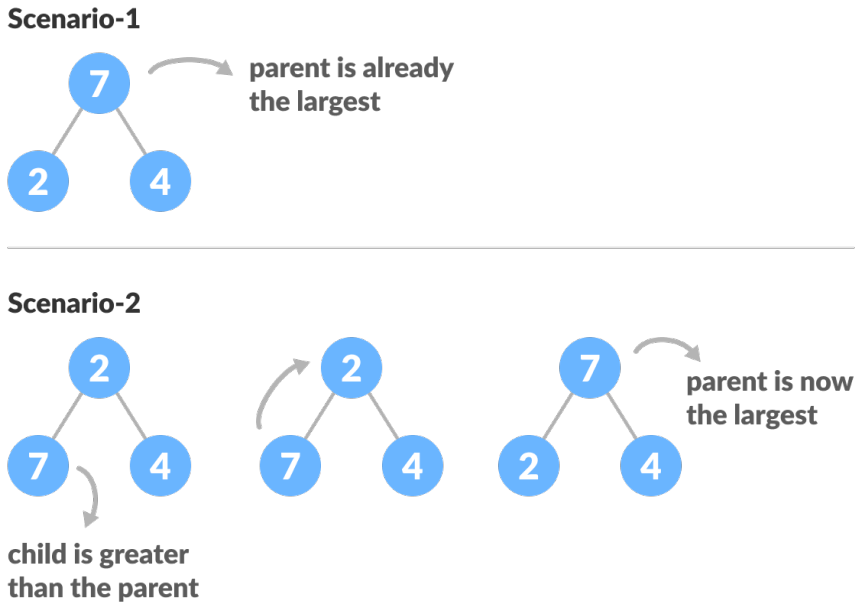


Heap Sort

Heap Sort works by first building a max-heap from the input array, and then repeatedly extracting the maximum element from the heap and placing it at the end of the array. The size of the heap is reduced by one after each extraction, and the heap is restored to a max-heap using the heapify operation.

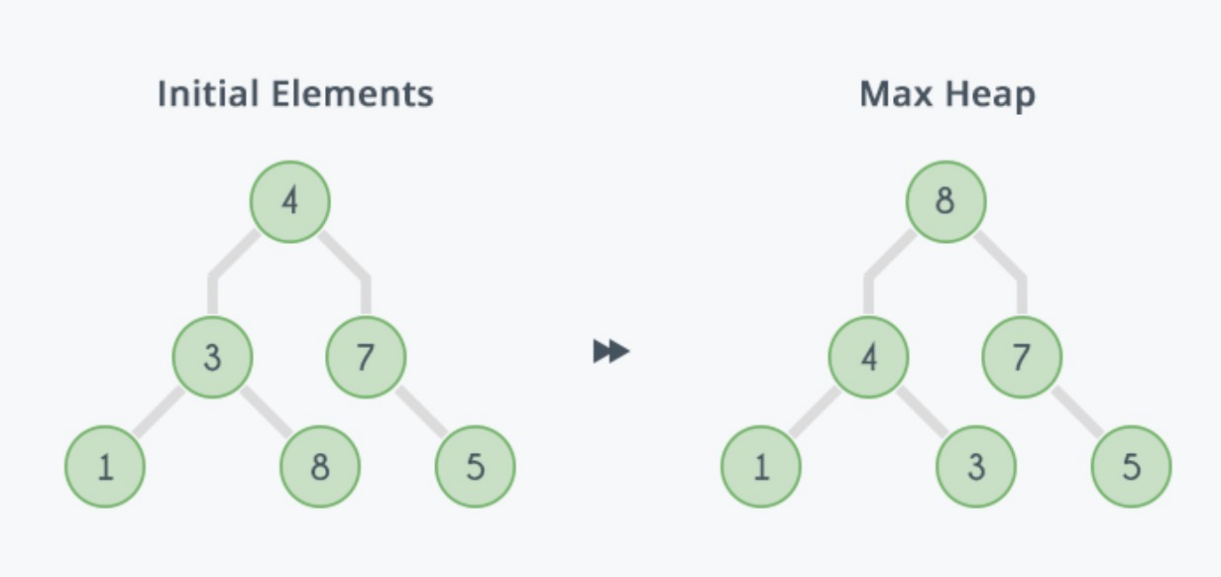
Heapify

Heapify is the process of creating a heap data structure from a binary tree represented using an array. It is used to create Min-Heap or Max-heap. Start from the last index of the non-leaf node whose index is given by n/2 – 1. Heapify uses recursion.

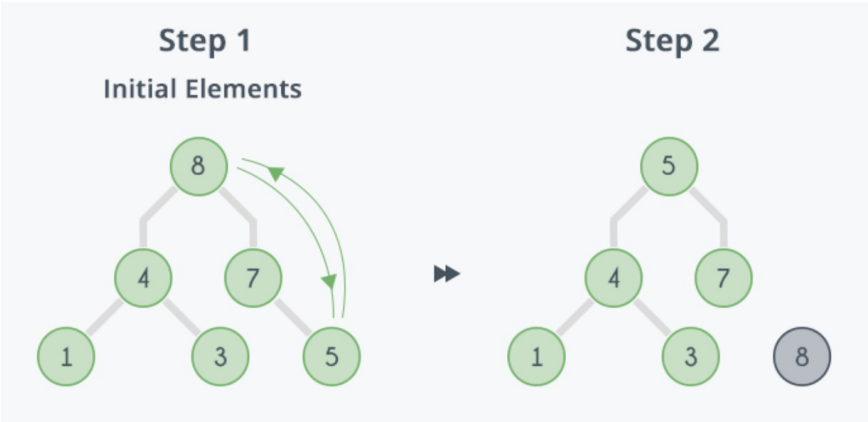


Heap Sort

Initially there is an unsorted array having 6 elements and then max-heap will be built.

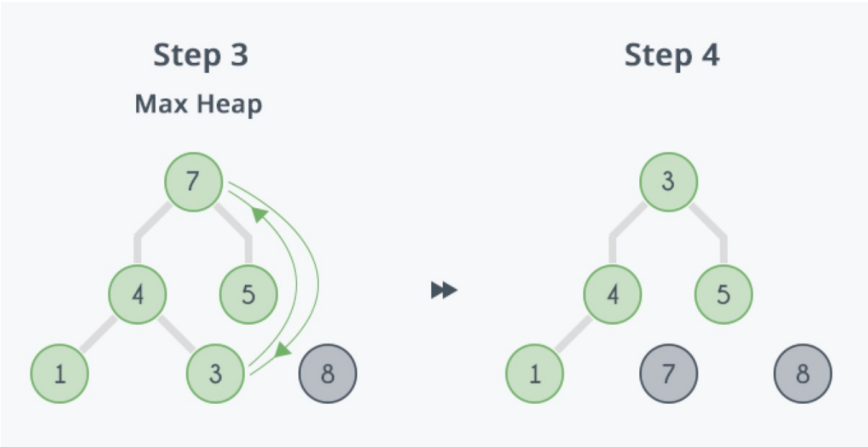


Heap Sort



Step 1: 8 is swapped with 5.

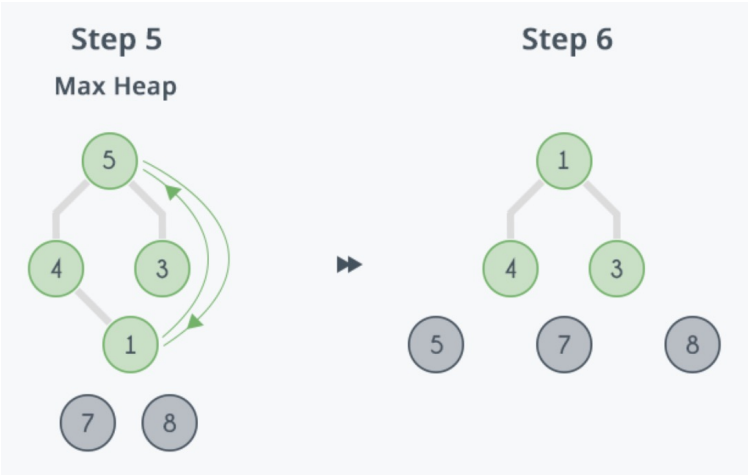
Step 2: 8 is disconnected from heap as 8 is in correct position now and.



Step 3: Max-heap is created and 7 is swapped with 3.

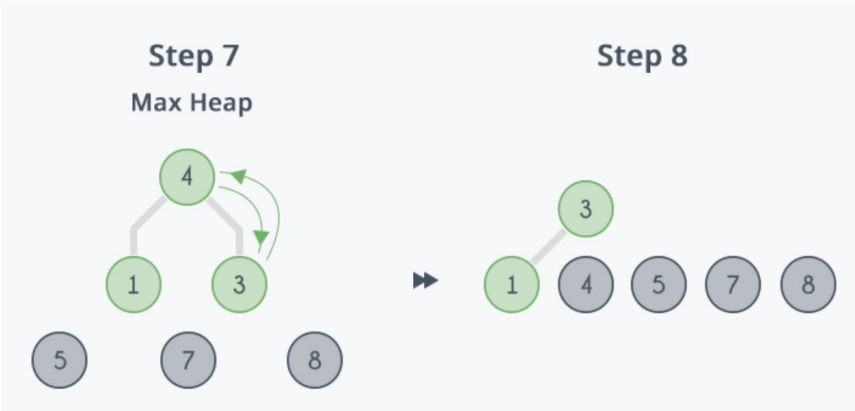
Step 4: 7 is disconnected from heap.

Heap Sort



Step 5: Max heap is created and 5 is swapped with 1.

Step 6: 5 is disconnected from heap.



Step 7: Max heap is created and 4 is swapped with 3.

Step 8: 4 is disconnected from heap.

Heap Sort

Step 9: Max heap is created and 3 is swapped with 1.

Step 10: 3 is disconnected.

Algo for Heap Sort

Step 1: Build Heap. Build a heap from the input data. Build a max heap to sort in increasing order, and build a min heap to sort in decreasing order.
Step 2: Swap Root. Swap the root element with the last item of the heap.
Step 3: Reduce Heap Size. Reduce the heap size by 1.
Step 4: Heapify. Heapify the remaining elements into a heap of the new heap size by calling heapify on the root node.
Step 5: Call Recursively. Repeat steps 2,3,4 as long as the heap size is greater than 2.