Tree represents the nodes connected by edges. We will discuss binary tree or binary search tree specifically.

A binary tree is a data structure in which each node has at most two children generally referred to as the left child and right child. Each node contains three components:

1. Pointer to left subtree
2. Pointer to right subtree
3. Data element

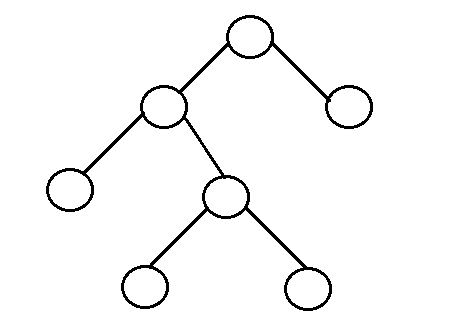
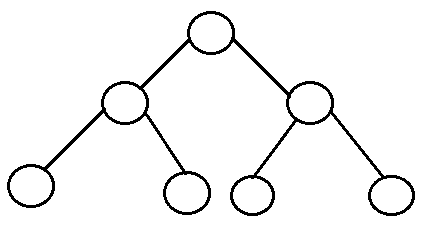
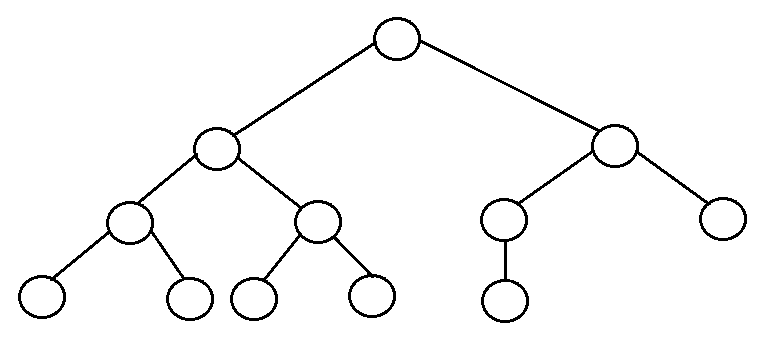
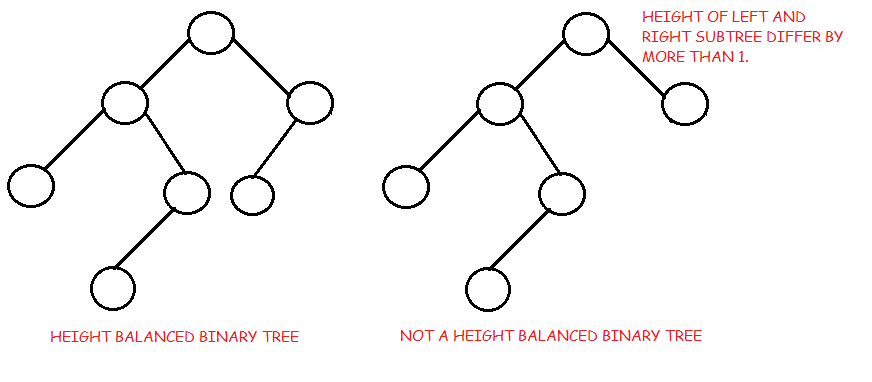
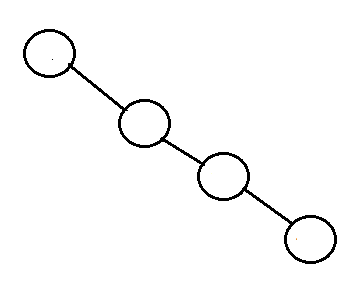


## Important Terms

Following are the important terms with respect to a tree.

* **Path − Path refers to the sequence of nodes along the edges of a tree.**
* **Root − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.**
* **Parent − Any node except the root node has one edge upward to a node called a parent.**
* **Child − The node below a given node connected by its edge downward is called its child node.**
* Leaf − The node which does not have any child node is called the leaf node.
* Depth of a node: Number of edges from the root to the node.
* Height of a node: Number of edges from the node to the deepest leaf. Height of the tree is the height of the root.

### Types of Binary Trees (Based on Structure)

* Full binary tree: It is a tree in which every node in the tree has either 0 or 2 children.  
  
* Perfect binary tree: It is a binary tree in which all interior nodes have two children and all leaves have the same depth or same level.  
  
* Complete binary tree: It is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.  
  
* Balanced binary tree: A binary tree is height balanced if it satisfies the following constraints:
  + The left and right subtrees' heights differ by at most one, AND
  + The left subtree is balanced, AND
  + The right subtree is balanced.  
    
* Degenerate tree: It is a tree where each parent node has only one child node. It behaves like a linked list.  
  

## Binary Search Tree Representation

Binary Search tree exhibits a special behavior. A node's left child must have a value less than its parent's value and the node's right child must have a value greater than its parent value.



## BST Basic Operations

The basic operations that can be performed on a binary search tree data structure, are the following −

* Insert − Inserts an element in a tree/create a tree.
* Search − Searches an element in a tree.
* Delete:

## Insert Operation

The very first insertion creates the tree. Afterward, whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

Implementation

## Search Operation

Whenever an element is to be searched, start searching from the root node, then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

Implementation

Traversal is a process to visit all the nodes of a tree. There are three ways which we use to traverse a tree −

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

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

## In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.



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. The output of inorder traversal of this tree will be −

*L → J → M → I → N → K → O*

### Algorithm

Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree.

## Pre-order Traversal

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



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. The output of pre-order traversal of this tree will be −

*I → J → L → M → K → N → O*

### Algorithm

Until all nodes are traversed −

Step 1 − Visit root node.

Step 2 − Recursively traverse left subtree.

Step 3 − Recursively traverse right subtree.

## Post-order Traversal

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.



We start from A, and following Post-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. The output of post-order traversal of this tree will be −

*L → M → J → N → O → K → I*

### Algorithm

Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Recursively traverse the right subtree.

Step 3 − Visit root node.

# 

# Complexity Analysis

