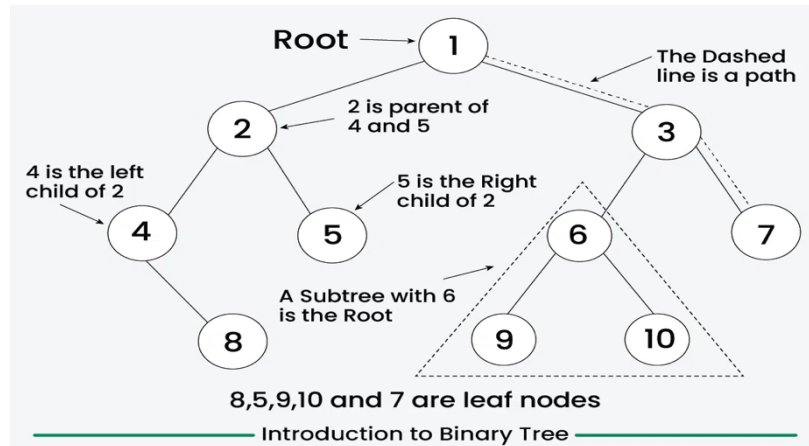
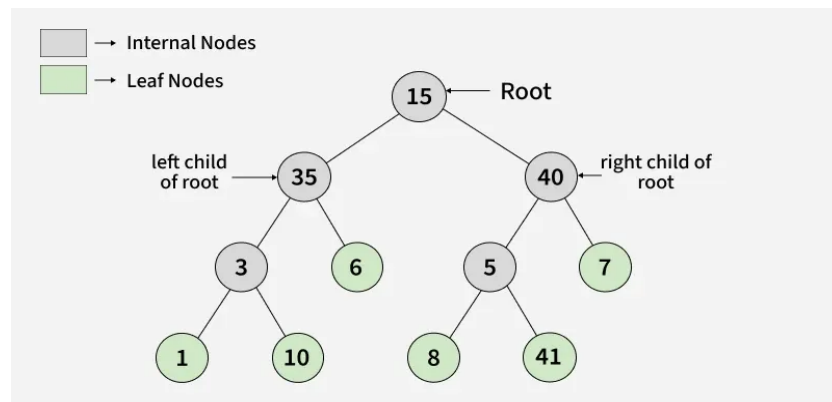


Binary Tree Data Structure

A **Binary Tree Data Structure** is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child.



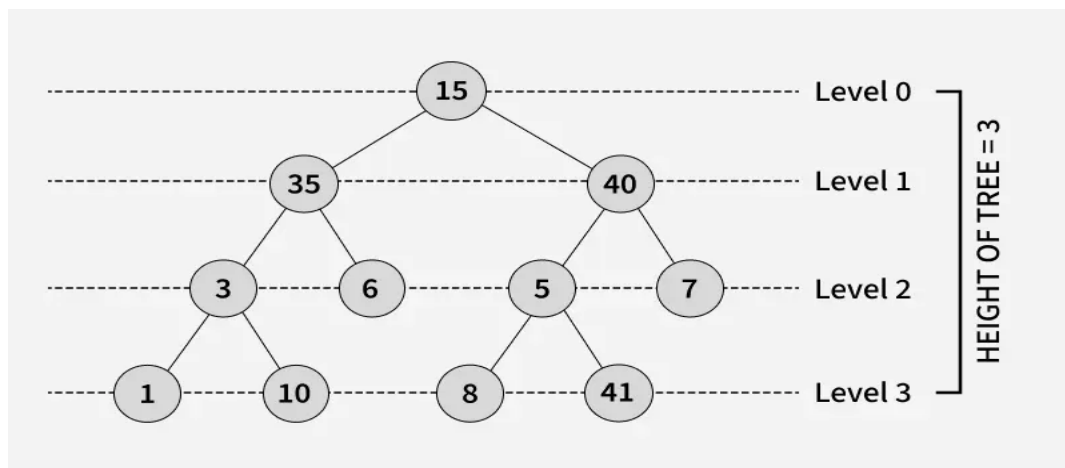
Binary Tree is a non-linear and hierarchical data structure where each node has **at most two children** referred to as the left child and the right child. The topmost node in a binary tree is called the root, and the bottom-most nodes(having no children) are called leaves.



Terminologies in Binary Tree

- **Parent Node:** A node that is the **direct ancestor** of a node(its child node).
- **Child Node:** A node that is the **direct descendant** of another node (its parent).
- **Ancestors of a node:** All nodes on the path from the root to that node (including the node itself).

- **Descendants of a node:** All nodes that lie in the subtree rooted at that node (including the node itself).
- **Subtree of a node:** A tree consisting of that node as root and all its descendants.
- **Edge:** The link/connection between a parent node and its child node.
- **Path in a binary tree:** A sequence of nodes connected by edges from one node to another.
- **Leaf Node:** A node that does not have any children or both children are null.
- **Internal Node:** A node that has at least one child.
- **Depth/Level of a Node:** The number of edges in the path from root to that node. The depth/level of the **root** node is zero.
- **Height of a Binary Tree:** The number of edges on the longest path from root to a leaf.



Properties of Binary Tree

- The maximum number of nodes at level **L** of a binary tree is 2^L .
- The maximum number of nodes in a binary tree of height **H** is $2^{H+1} - 1$.
- Total number of leaf nodes in a binary tree = total number of nodes with 2 children + 1.
- In a Binary Tree with **N** nodes, the minimum possible height or the minimum number of levels is $\lceil \log_2 N \rceil$.
- A Binary Tree with **L** leaves has at least $\lceil \log_2 L \rceil + 1$ levels.

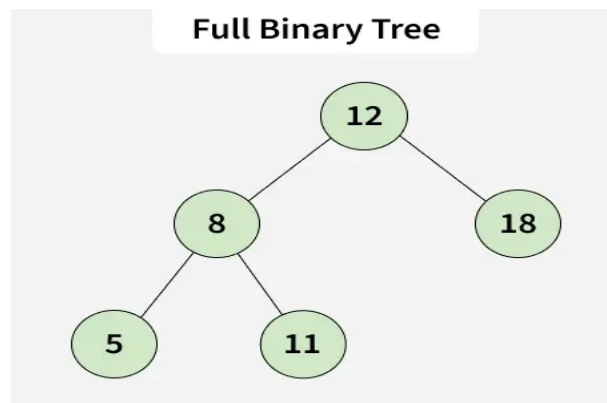
Types of Binary Tree

1. On the basis of number of children:

- Full Binary Tree
- Degenerate Binary Tree
- Skewed Binary Trees

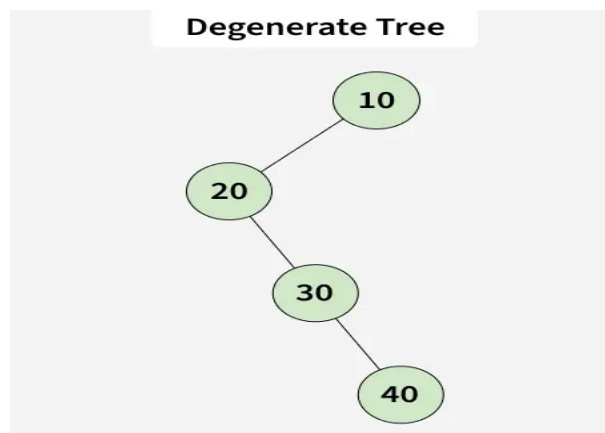
Full Binary Tree

A Binary Tree is a full binary tree if every node has 0 or 2 children. We can also say a full binary tree is a binary tree in which all nodes except leaf nodes have two children. A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children. It is also known as a proper binary tree.



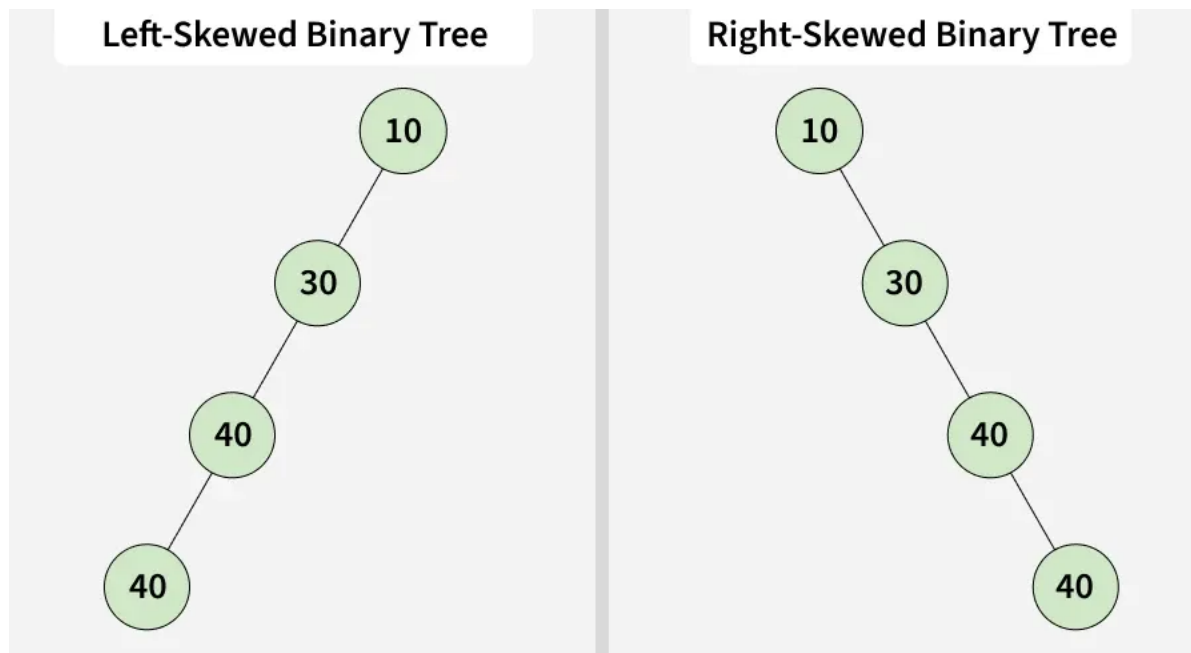
Degenerate (or pathological) tree

A Tree where every internal node has one child. Such trees are performance-wise same as linked list. A degenerate or pathological tree is a 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.



2. On the basis of level completion:

- Complete Binary Tree
- Perfect Binary Tree
- Balanced Binary Tree

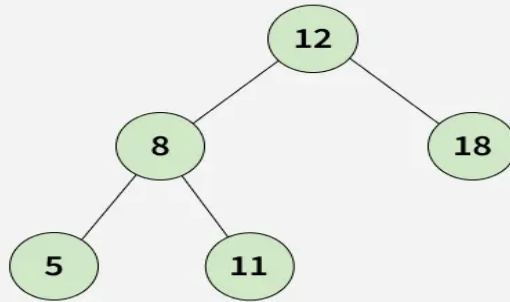
A. Complete Binary Tree

A Binary Tree is a Complete Binary Tree if all the levels are completely filled except possibly the last level and the last level is filled from left side.

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

- Every level except the last level must be completely filled.
- All the leaf elements must lean towards the left.

Complete Binary Tree



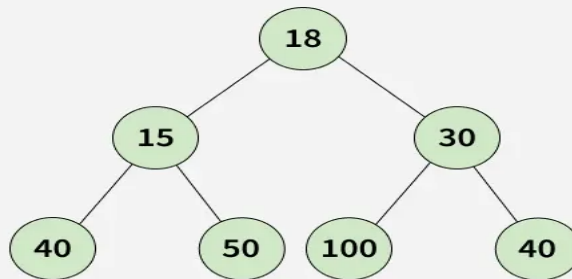
B. Perfect Binary Tree

A Binary tree is a Perfect Binary Tree in which all the internal nodes have two children and all leaf nodes are at the same level.

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.

In a Perfect Binary Tree, the number of leaf nodes is the number of internal nodes plus 1.

Perfect Binary Tree

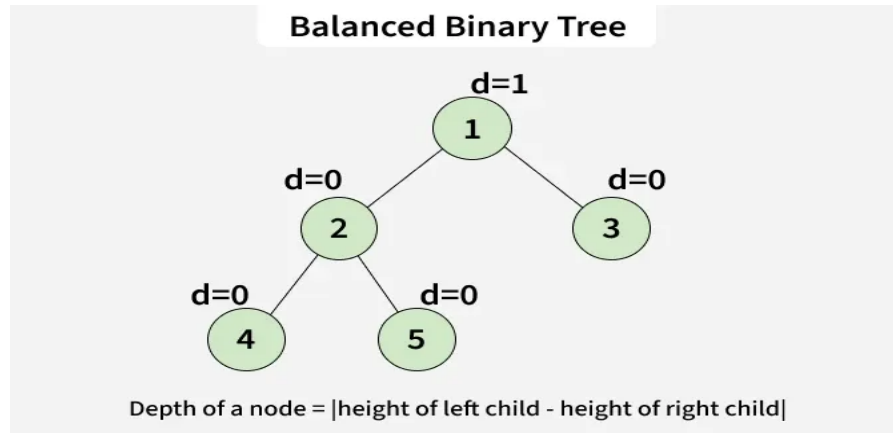


C. Balanced Binary Tree

A binary tree is balanced if the height of the tree is $O(\log n)$ where n is the number of nodes.

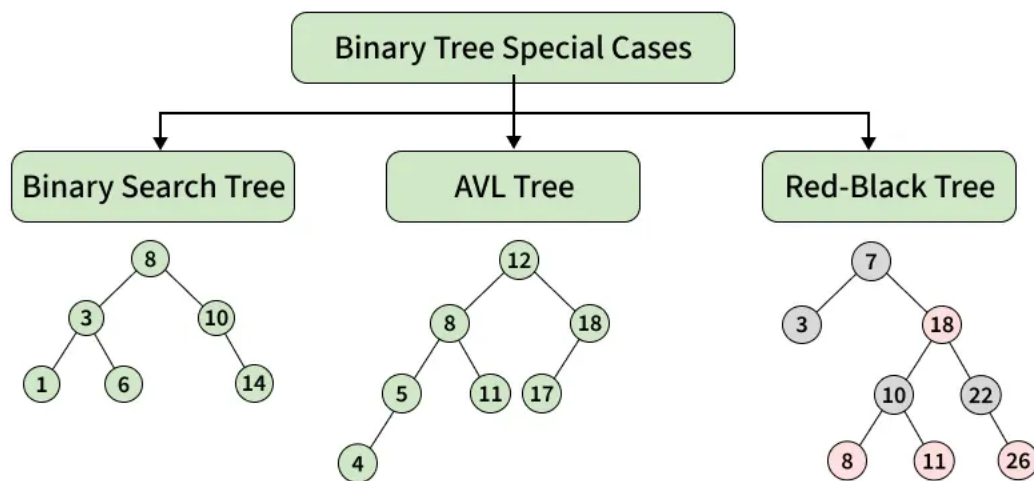
For example, the AVL tree maintains $O(\log n)$ height by making sure that the difference between the heights of the left and right subtrees of every node is at most 1.

Following is an example of balanced binary tree. Here, $d = \text{depth of node} = |\text{Height of Left child} - \text{Height of Right child}|$.



Some Special Types of Trees:

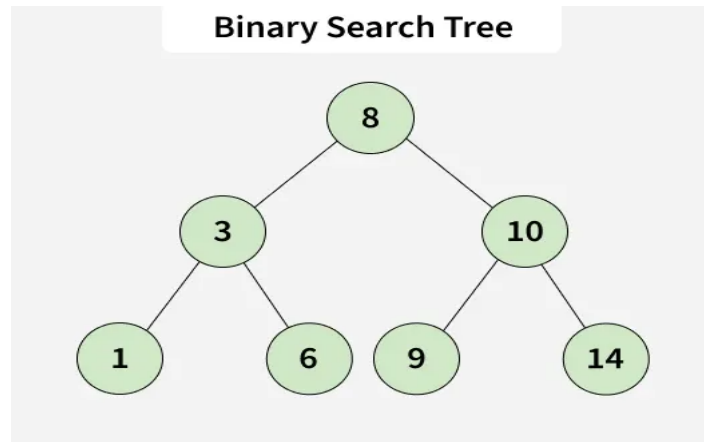
On the basis of node values, the Binary Tree can be classified into the following special types:



1. Binary Search Tree

Binary Search Tree is a node-based binary tree data structure that has the following properties:

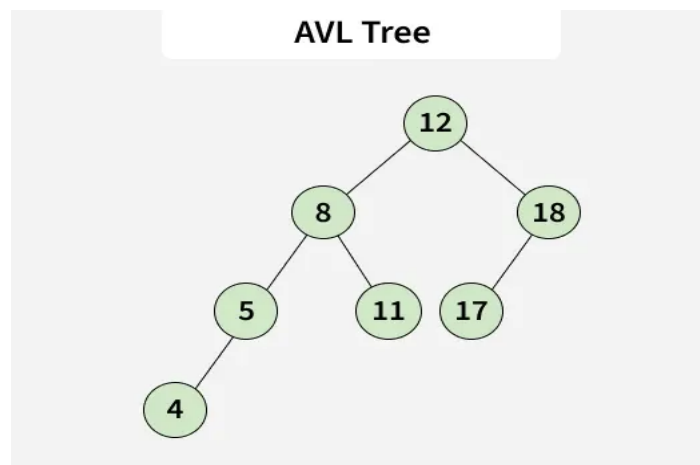
- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree must also be a binary search tree.



2. AVL Tree

AVL tree is a **self-balancing Binary Search Tree (BST)** where the difference between heights of left and right subtrees of all nodes is at most 1.

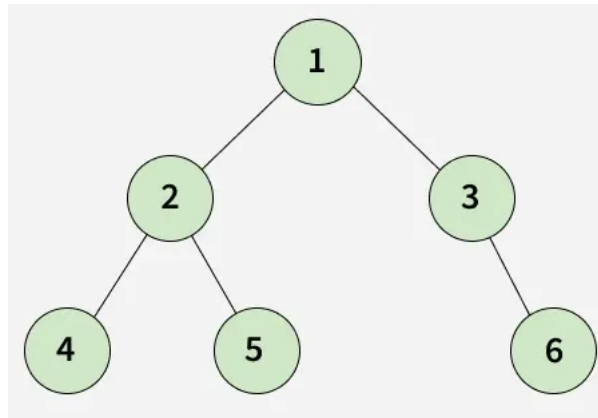
The below tree is AVL because the difference between the heights of left and right subtrees for each node ≤ 1 .



Basic Operations

1- Inorder Traversal of Binary Tree

Inorder Traversal is a method to traverse a tree such that for each node, you first traverse its left subtree, then visit the node itself, and finally traverse its right subtree.



Output: [4, 2, 5, 1, 3, 6]

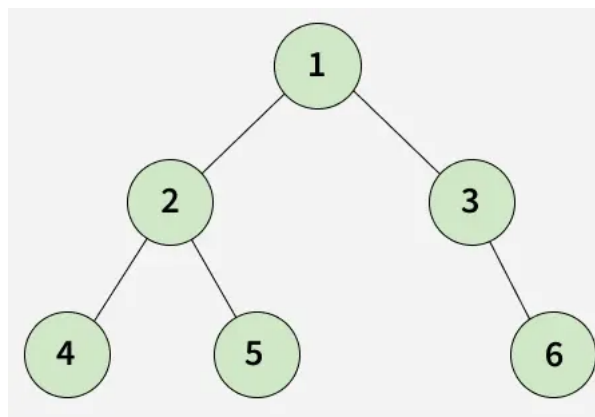
Time Complexity: $O(n)$

Auxiliary Space: $O(h)$, h is the height of the tree

- In the worst case, h can be the same as n (when the tree is a skewed tree)
- In the best case, h can be the same as $\log n$ (when the tree is a complete tree)

2- Preorder Traversal of Binary Tree

Preorder Traversal is a method to traverse a tree such that for each node, you first visit the node itself, then traverse its left subtree, and finally traverse its right subtree.



Output: [1, 2, 4, 5, 3, 6]

Time Complexity: $O(n)$

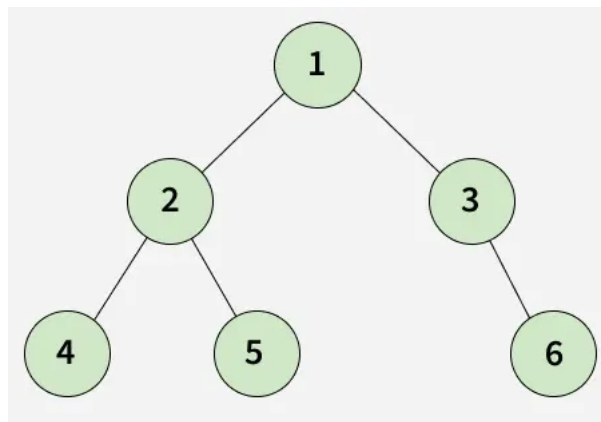
Auxiliary Space: $O(h)$, h is the height of the tree

- In the worst case, h can be the same as n (when the tree is a skewed tree)
- In the best case, h can be the same as $\log n$ (when the tree is a complete tree)

3- Postorder Traversal of Binary Tree

Given a **root** of the binary tree, return the **postorder traversal** of the binary tree.

Postorder Traversal is a method to traverse a tree such that for each node, you first traverse its left subtree, then its right subtree, and finally visit the node itself.



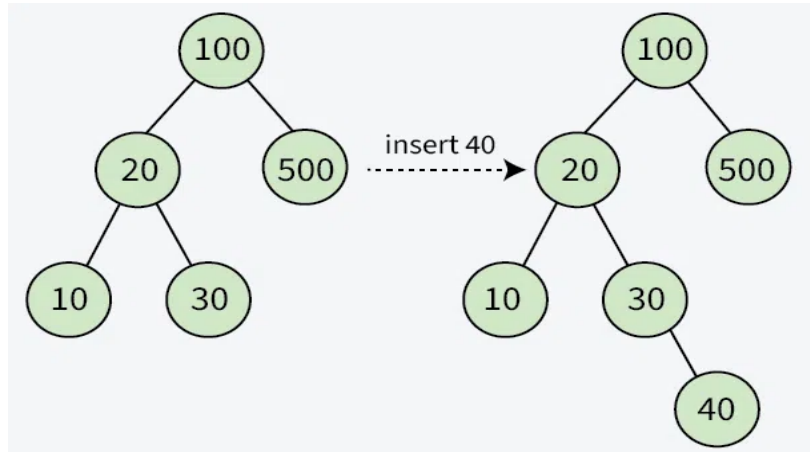
Time Complexity: $O(n)$

Auxiliary Space: $O(h)$, h is the height of the tree

4- Insertion in Binary Search Tree (BST)

Given the **root** of a Binary Search Tree, we need to insert a new node with given value in the **BST**. All the nodes have distinct values in the BST and we may assume that the new value to be inserted is not present in BST.

Example:



5- Deletion in Binary Search Tree (BST)

Given the **root** of a Binary Search Tree (BST) and an integer **x**, delete the node with value **x** from the BST while maintaining the BST property.

Example: **Input:** $x = 15$ delete it

