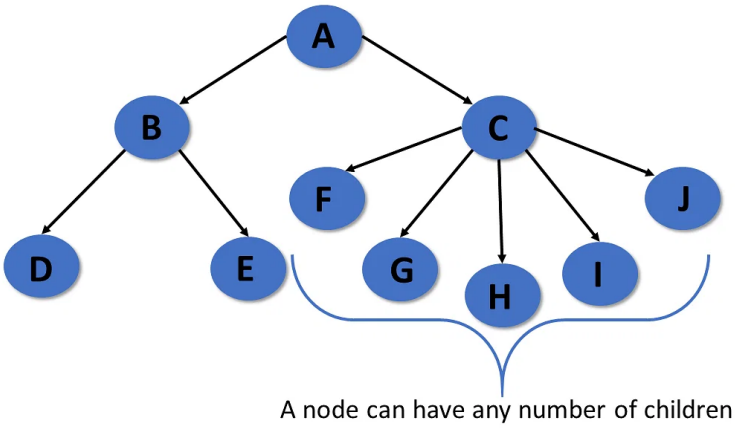
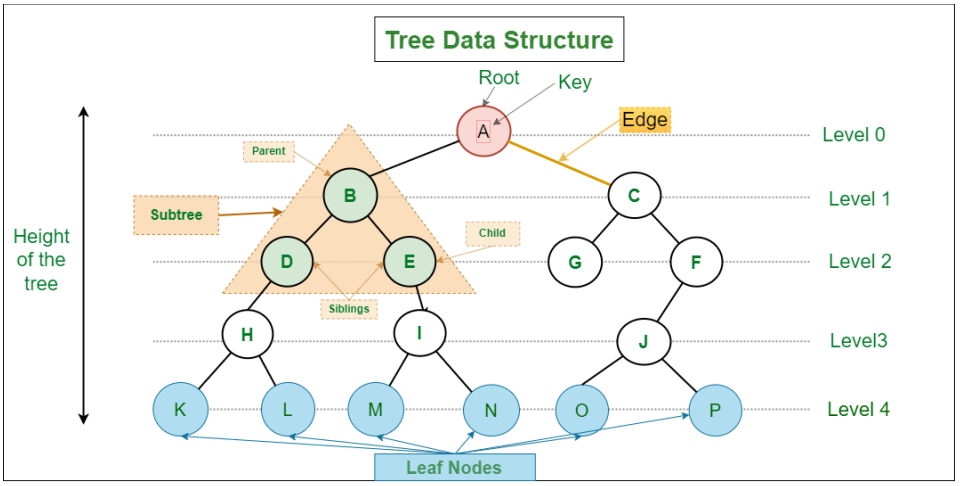
Tree Data Structure

What is a Tree data structure?

A Tree is a non-linear hierarchical data structure like a file system in computers. A tree is a collection of nodes and each node is connected to each other through edges, where each node in a tree can point to any number of nodes.

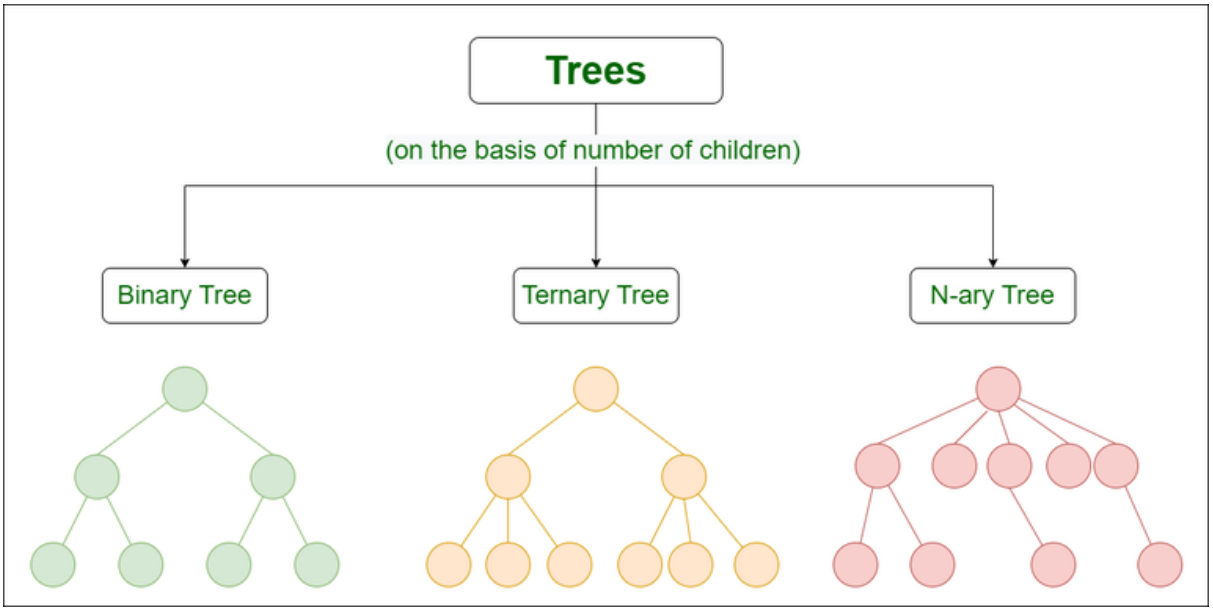


The **topmost** node of the tree is called the **root node** , and the nodes below it are called the **child nodes**. Each node can have multiple child nodes, and these child nodes can also have their own child nodes.



Types of Tree Data Structure:

Trees can be categorized into various types based on the number of children each node can have. The primary categories include:



1. **Binary Tree:**
    a. In a binary tree, each node can have at most two children or child nodes. These are usually referred to as the left child and the right child.
2. **Ternary Tree:**
    a. In a ternary tree, each node can have at most three children or child nodes. These are usually referred to as the left child, mid child and the right child.
3. **N-ary Tree or Generic Tree**
    a. An N-ary tree is a more general concept where each node can have any number of children.

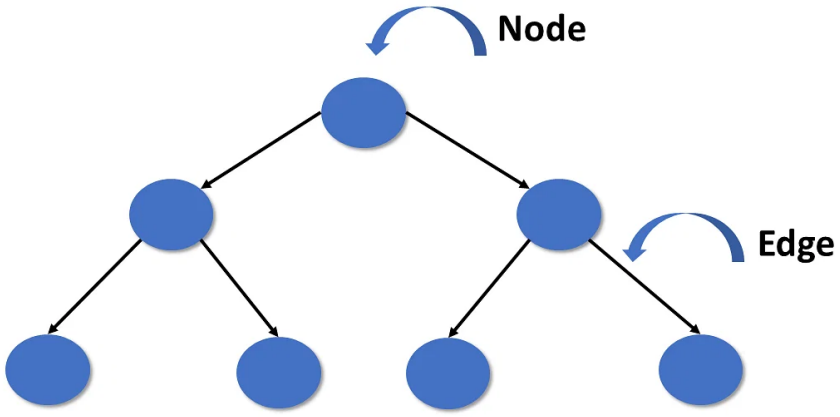
Why Tree is considered a non-linear data structure?

The data in a tree are not stored in a sequential manner i.e., they are not stored linearly. Instead, they are arranged on multiple levels or we can say it is a hierarchical structure. For this reason, the tree is considered to be a non-linear data structure.

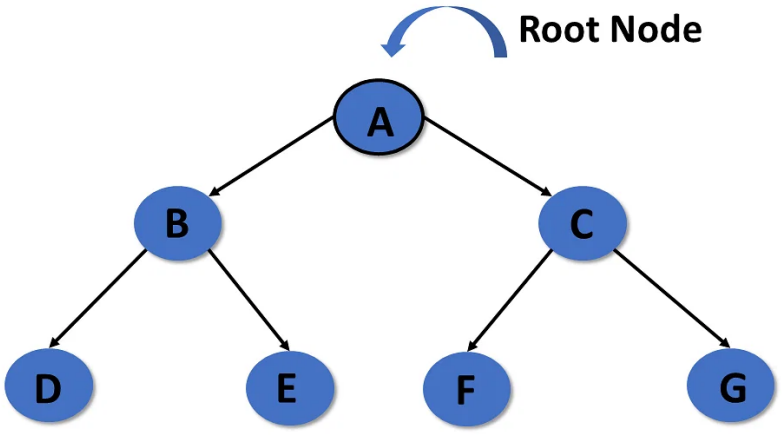
Basic Terminologies In Tree Data Structure:

1. **Node**
    a. A tree is a collection of nodes and these nodes are connected with each other via edges (lines).
    b. Like linked lists, nodes in tree data structures also have two main components:
        - **Data Part:** This part of the node stores the actual data or value associated with the node.
        - **Pointer Part:** This part of the node contains references or pointers to other nodes, allowing nodes to be connected. The number of pointers in the pointer part depends on the type of tree and its structure.

- For instance:
    - In a binary tree, nodes typically have two pointers: a left pointer and a right pointer, because in a binary tree, each node can have at most two children or child nodes.
    - In a ternary tree, nodes might have three pointers, left, mid and right.
    - In a general tree, nodes can have references to an arbitrary number of child nodes.

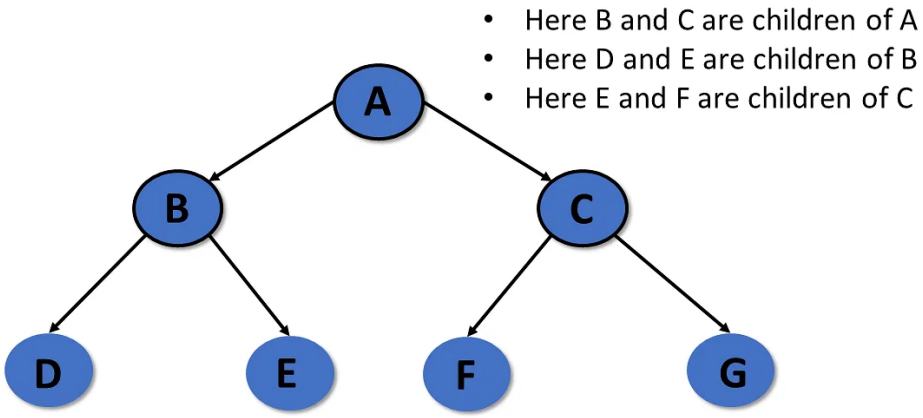


2. **Edge:**
    a. Edge is a line used to connect two or more nodes.
2. **Root Node**
    a. The first node or the topmost node of the the tree is called the root node.



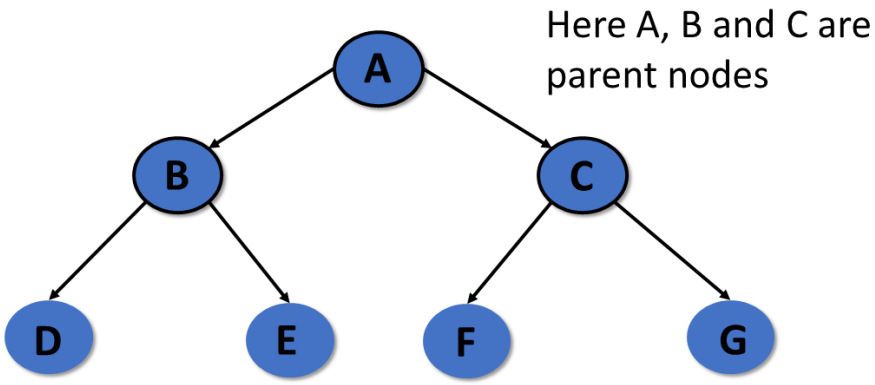
4. **Child Node or Children**

    a. Child is a node that has a parent node.



5. **Parent Node or Parent**

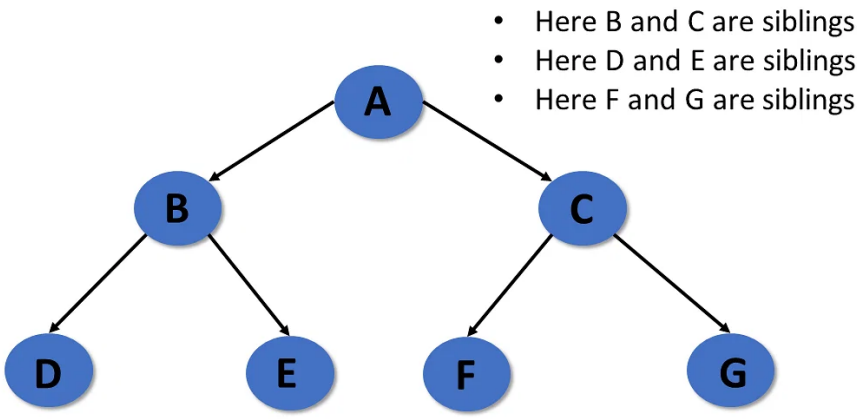
    a. Parent is a node that has an edge to a child node.



6. **Internal node:**

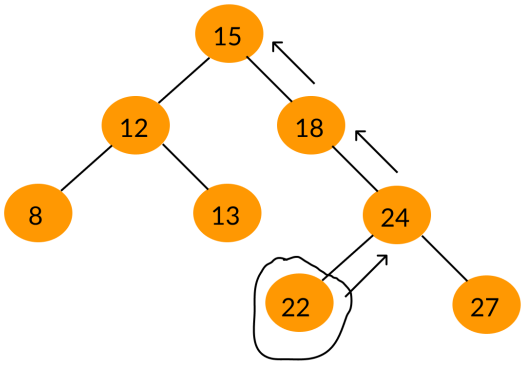
    a. A node that have at least one children is called Internal Node.

7. **Siblings (means Real Brother and Sister)**
    a. Children of the same parent node are called siblings.
    b. In simple word, if two or more child nodes having same parent node, then these child nodes is considered as siblings.
    c. In Simple Words, Siblings are Children of same parent.



8. **Ancestor (means पूर्वज)**
    a. Suppose we have to find the ancestors of node 22 in the below given tree.
    b. Now, to find the ancestors of node 22 what you have to do is,
        i. Pick the node whose ancestor you want to find. Now, move toward the root node, and each node that comes into the path from that node to the root node, that nodes are the ancestor of that particular node.
    c. So, Ancestors of 22 are 24, 18, and 15.

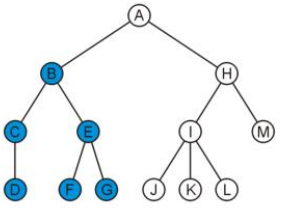
Ancestors of a node in a binary tree



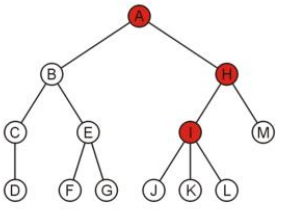
Ancestors of 22 are 24, 18 and 15

9. **Descendant (means वंशज)**
   a. Suppose we have to find the descendants of node B in the below given tree.
   b. Now, to find the descendants of node B what you have to do is,
      i. Pick the node whose descendant you want to find. Now, move toward the leaf node, and each node that comes into the path from that node to the leaf node, that nodes are the descendants of that particular node.

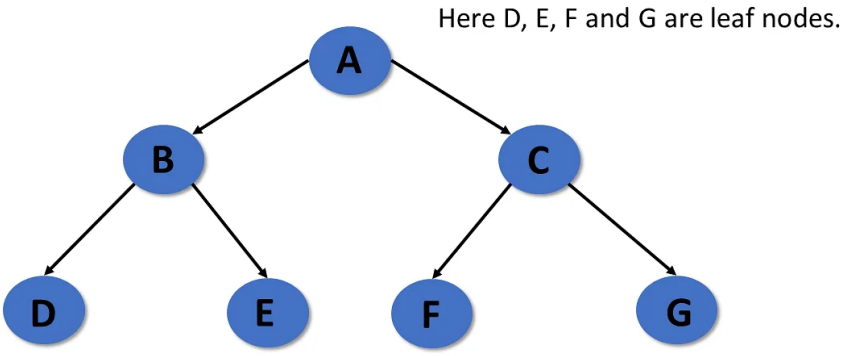
The descendants of node B are B, C, D, E, F, and G:



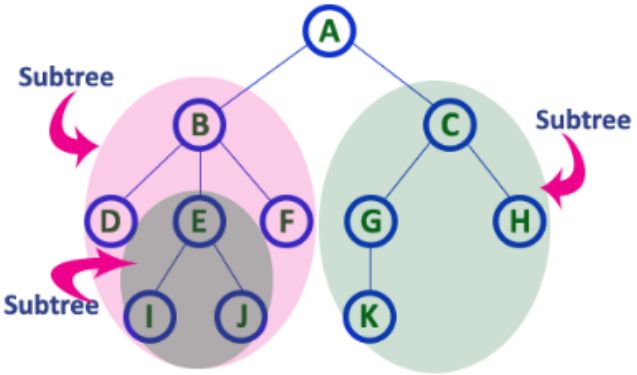
The ancestors of node I are I, H, and A:



10. **Leaf Node**
    a. A leaf is a node that does not have a child node in the tree.



11. **Subtree**
    a. A subtree is a portion of a tree that is itself a valid tree.



Properties of a Tree:

1. **Number of edges:**
    - In a tree with N nodes, there are (N-1) edges.
2. **Size of a Tree:**
    - Size of a Tree = The total number of nodes in the tree.
3. **Depth of a node:**

- **Depth of a node = The depth of a node is defined as the number of edges in the path from the root node to that specific node.**

4. **Height of a node:**
    - The height of a node can be defined as the length of the longest path from that node to the deepest leaf node of the tree.
    - Height of a Node = number of edges from that node to the deepest leaf node.
5. **Height of the Tree:**
    - The height of a tree is the length of the longest path from the root of the tree to the deepest leaf node of the tree.
    - Height of the Tree = number of edges from that root node to the deepest leaf node.
6. **Degree of a Node:**
    - The degree of a node = the total number of nodes connected to that specific node. So, in binary tree, the maximum degree you can get is 2.
    - The degree of a leaf node must be **0**.

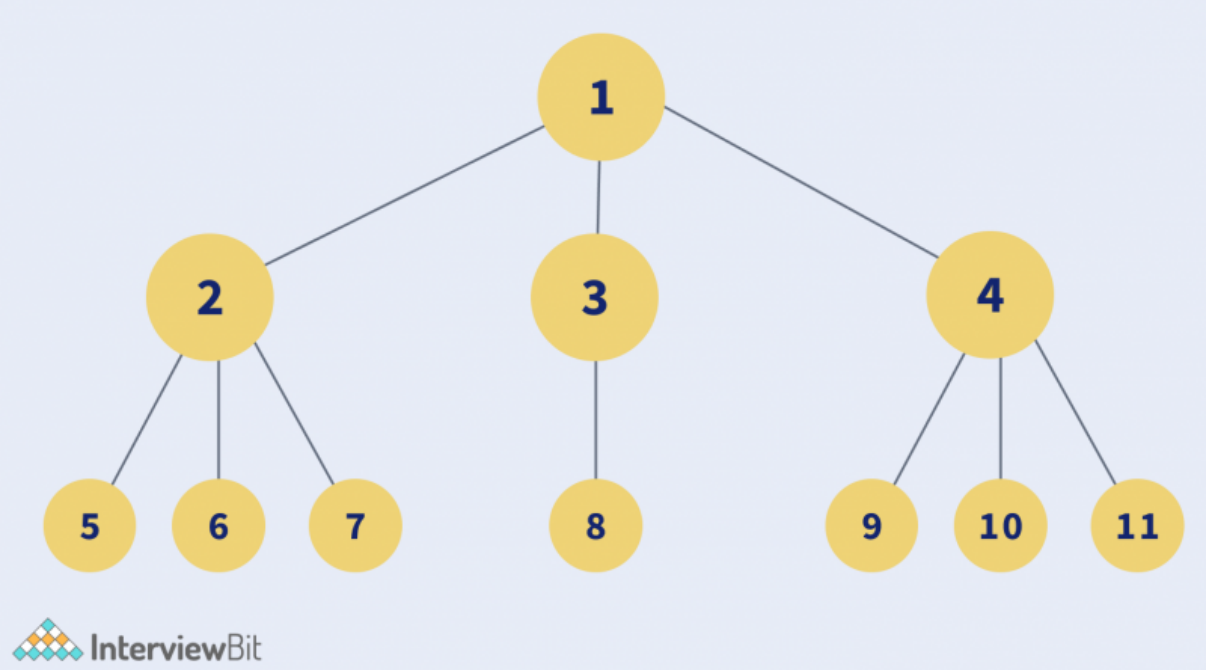
Some more properties are:

1. **Trees are Acyclic:** A fundamental property of trees is that they do not contain any cycles or loops.
2. **Trees do not have Self-Loops:** Trees also do not have self-loops, which means that a node in a tree cannot have an edge that points directly back to itself. In other words, a node cannot be connected to itself in a tree.

What is N-ary Tree in Tree Data Structure ?

N-ary trees are tree data structures that allow us to have up to **n** children nodes for each of the nodes, differing from the standard binary trees which allow only up to **2** children nodes for each node.

In simple words, In N-ary Tree, each node can have “n” number of children (child node).



What is Binary trees ?

A binary tree is a type tree data structure in which each node has at the most two children (child nodes), which are referred to as the “left child” and the “right child.”

In short, A tree is called Binary Tress, if each node has 0 child, 1 child or 2 child. Binary Tree ⇒ Each Node has ≤ 2 child.

> 💡 ***Empty tree is also a valid binary tree.***

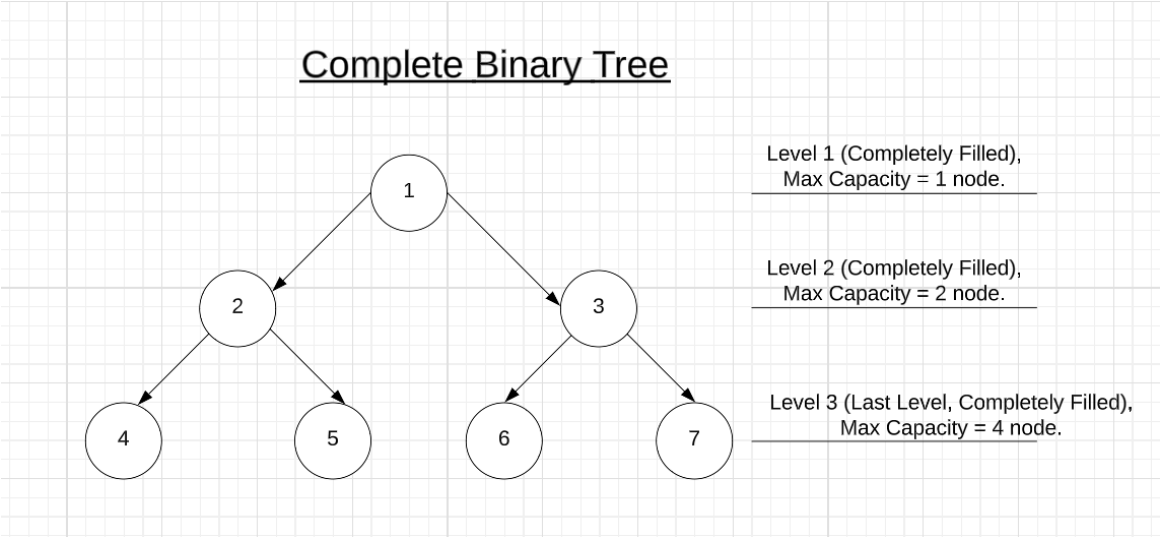
The node structure of Binary Tree contains three parts,

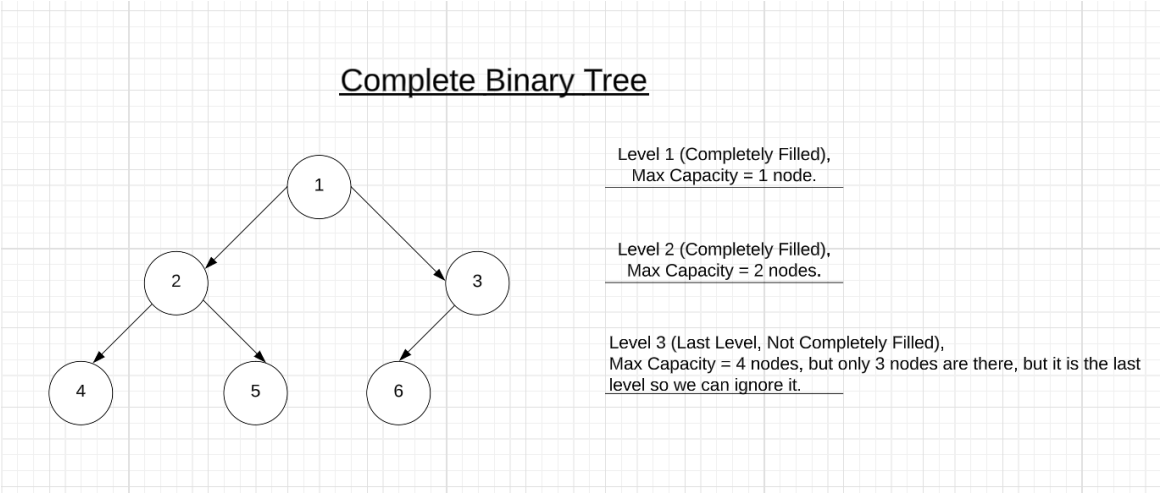
1. **Data Part**,
    a. This part of the node stores the actual data or value associated with the node.
2. **Pointer to left child**
    a. This pointer points to the left child of the current node, which is another node in the binary tree.
3. **Pointer to right child**
    a. This pointer points to the right child of the current node, which is another node in the binary tree.

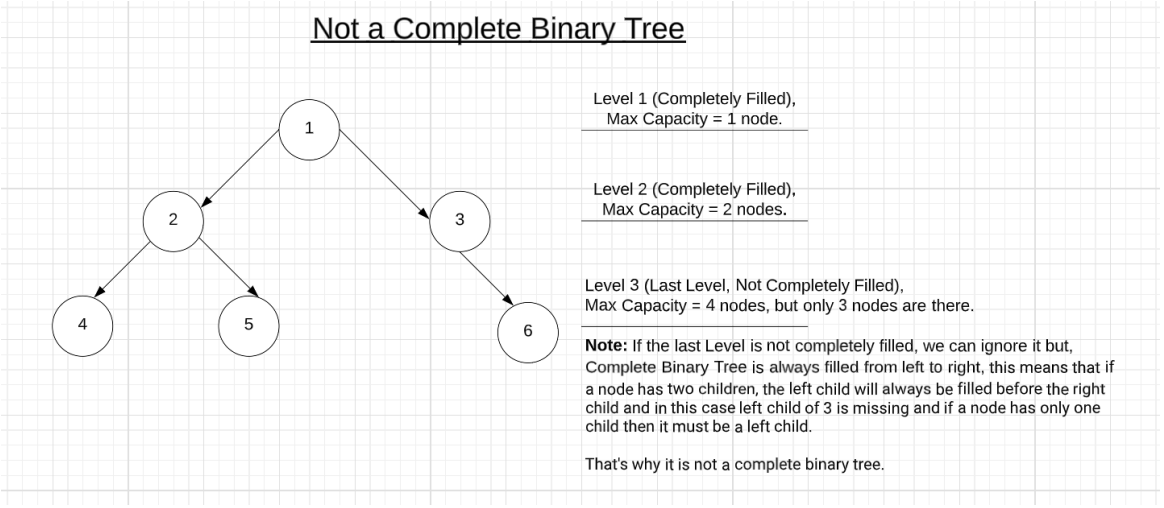
Types of Binary Tree:

Complete Binary Tree

1. A Complete Binary Tree is a binary tree in which all the levels are completely filled except the last level.
2. A Complete Binary Tree is always filled from left to right. This means that if a node has two children, the left child will always be filled before the right child.
3. And if a node has only one child, then that child must be left child because of the order that we have just discussed in point 2.

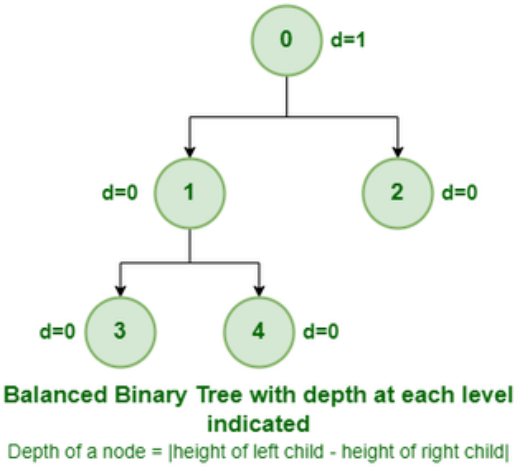






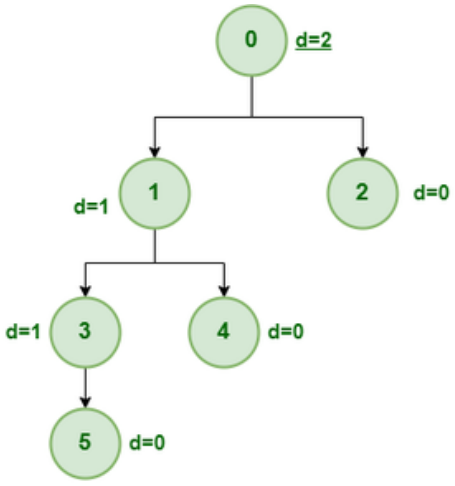
Balanced Binary Tree or Height Balanced Binary Tree

1. A binary tree is considered balanced if, for all nodes in the tree, the difference in height between its left subtree and right subtree (known as the balance factor) is not more than 1.
2. This ensures that the tree is approximately balanced, and no subtree is significantly taller than the other. Balanced binary trees provide efficient operations with a time complexity of O(log n) for search, insertion, and deletion.
3. The height of a balanced binary tree is approximately $\log_2(n)$, where "n" is the number of nodes in the tree.



Balanced Binary Tree with depth at each level indicated

Depth of a node = |height of left child - height of right child|



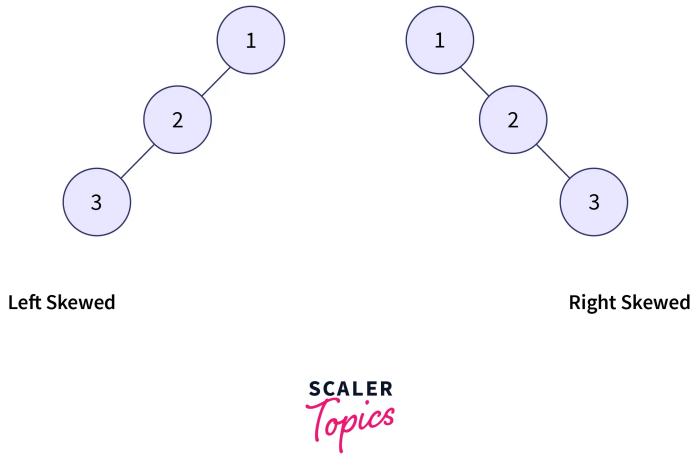
Unbalanced Binary Tree with depth at each level indicated

Depth of a node = |height of left child - height of right child|

Skewed Binary Tree

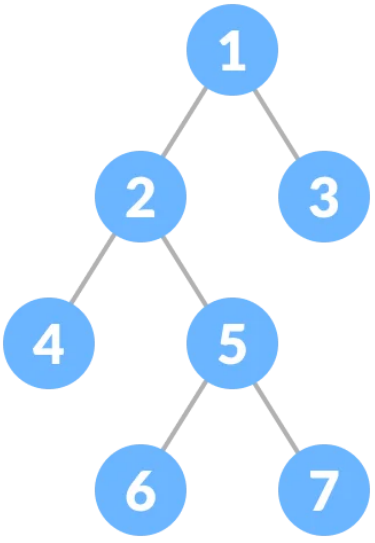
1. A skewed binary tree is a type of binary tree in which all the nodes have only either one child or no child.

2. Each node has only 1 child node or no child node.
3. **Types of Skewed Binary trees,**
    a. There are 2 types of Skewed Binary Tree,
        i. **Left Skewed Binary Tree:** Each node only have left child or no child.
        ii. **Right Skewed Binary Tree:** Each node only have right child or no child.



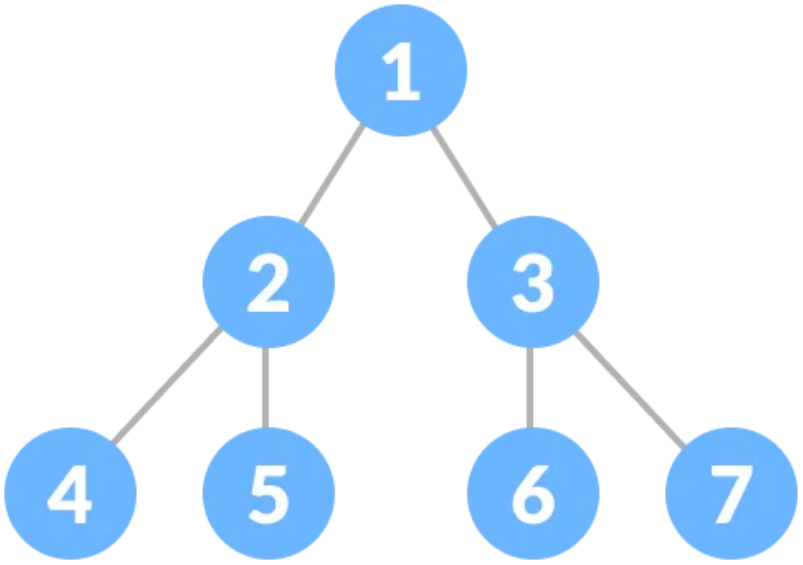
Full or Strict Binary Tree

a. In Full or Strict Binary Tree, every node has either two child or no child.



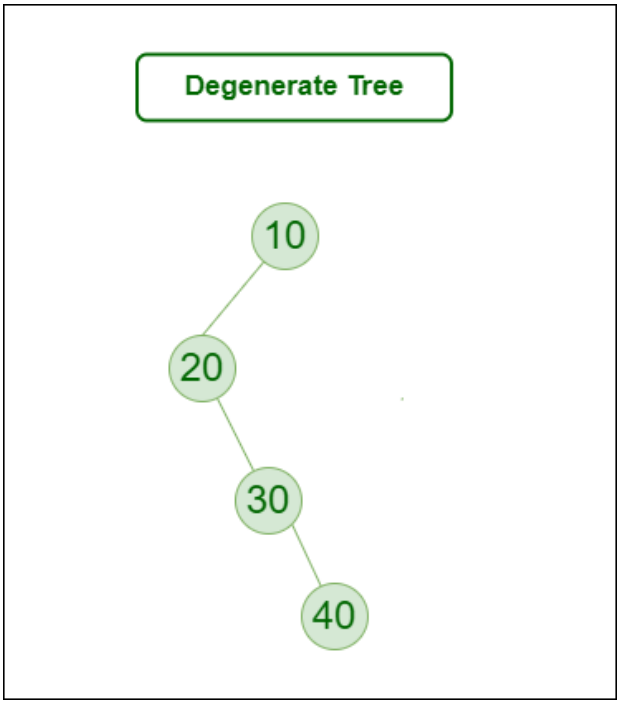
Perfect Binary Tree

1. In Perfect Binary Tree, Each node has exactly two child and all the leaf nodes are at same level.
2. In Perfect Binary Tree, All levels are completely filled.



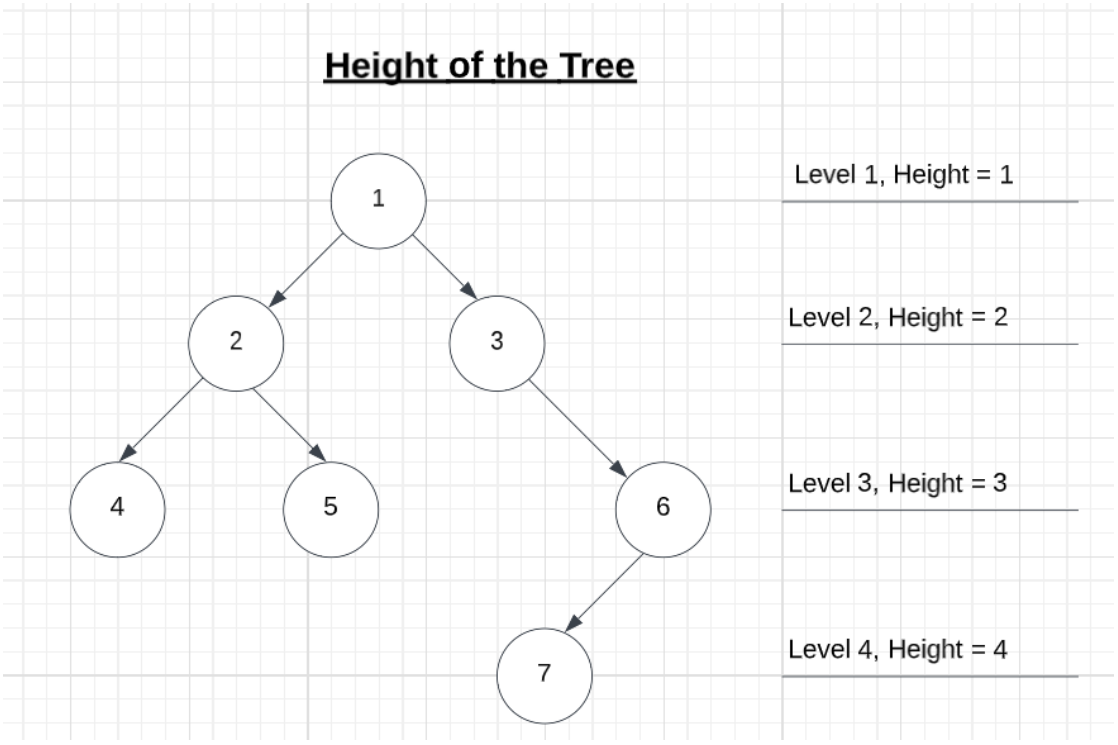
Degenerate or Pathological Tree

1. A degenerate or pathological tree is the tree having a single child either left or right.
2. Each node has single child node either left or right.

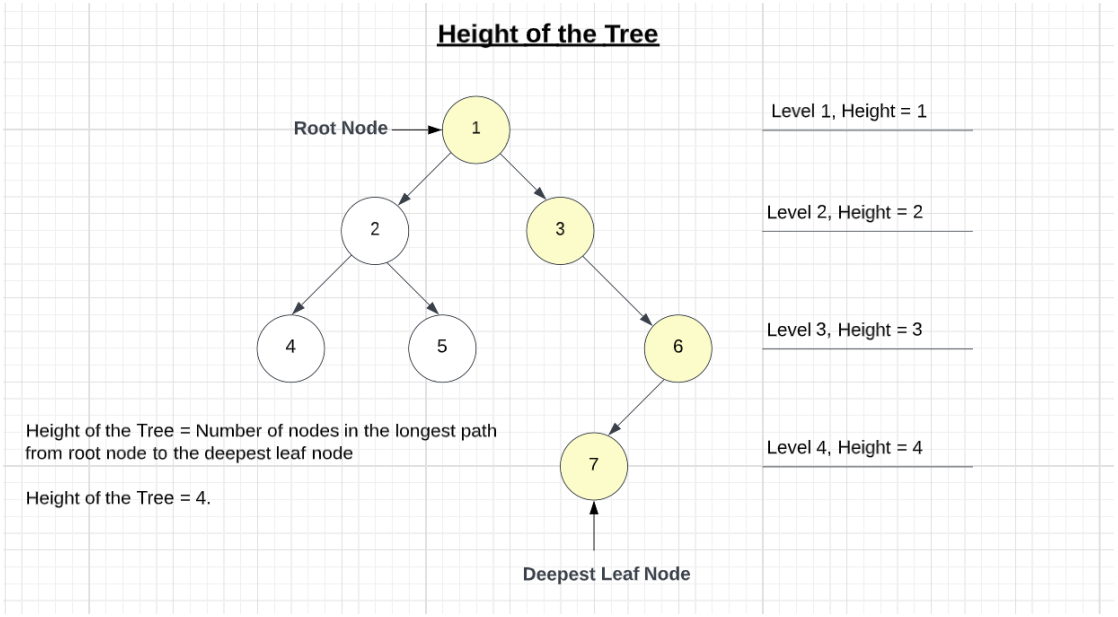


Properties of Tree:

1. **Height of the Tree**
    a. Height of the tree is the maximum levels we have in a tree.

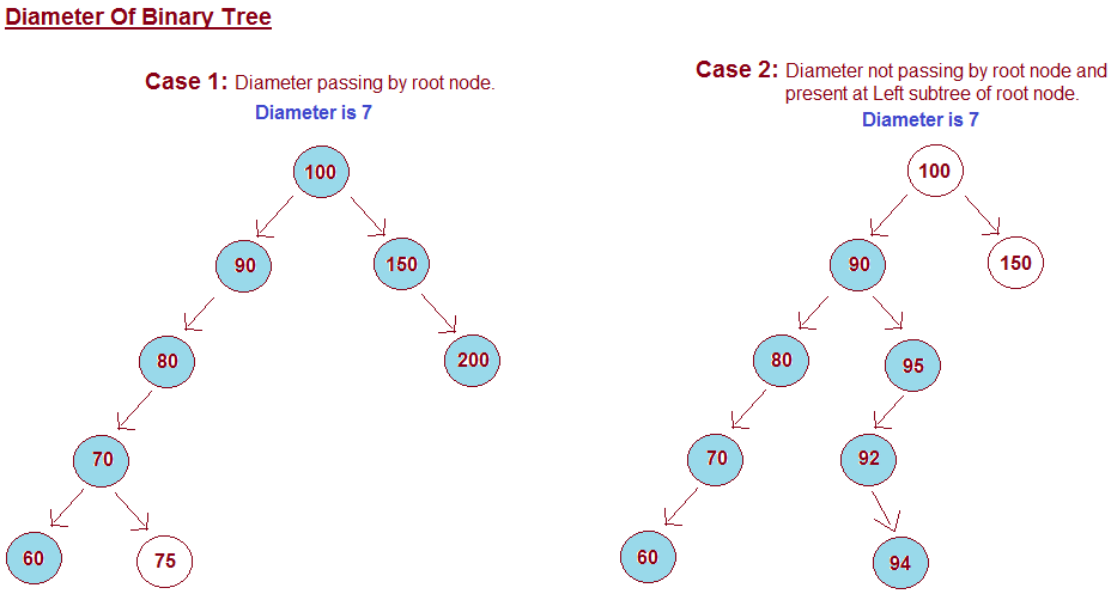


    b. or, The height of a tree is the length of the longest path from the root node of the tree to a leaf node of the tree. So, ***the number of nodes in this longest path from root to the deepest leaf node is the height of the tree.***



c. ***Note that the Height and Depth of the tree are same but the Height and Depth of the node may be different.***

2. **Diameter of a Tree**
   a. Diameter of Tree = The diameter of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the root node.
   b. *The length of a path between two nodes is represented by the number of edges between them.* ***Length = number of edges in the path.***
   c. Or, *The length of a path between two nodes is represented by the number of nodes between them.* ***Length = number of nodes in the path.***
   d. Whether to take **length = the number of edges in the path** or **length = the number of nodes in the path**. It is given in the question.



Properties that will help you in Questions:

1. The total number of nodes in a Perfect Binary Tree of height “h” is equal to **2^(h+1) - 1**, where “h” is the height of the Perfect Binary Tree.
2. The total number of leaf nodes in a Perfect Binary Tree of height “h” is equal to 2^h, where “h” is the height of the Perfect Binary Tree.
3. Internal Node (Nodes that has atleast one child node) in Perfect Binary Tree = Total number of nodes in Perfect Binary Tree - total number of leaf nodes in Perfect Binary Tree.
4. A binary tree with “n” leaf nodes, then a binary tree can have at least $\log_2(n+1)$ levels.
5. A binary tree with “n” nodes, then a binary tree can have a minimum height (levels) is $\log_2(n+1)$.
6. In Strict Binary Tree with “n” leaf nodes, then the total number of internal nodes is equal to **n-1**.

What is Binary Search Tree ?

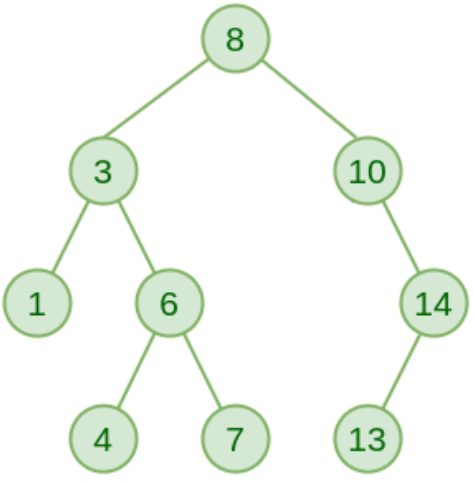
A Binary Search Tree (BST) is a binary tree data structure. But BST has one property that separate a binary search tree from a regular binary tree and that property is:

For Every Node,

1. All values in the left subtree of a node are less than or equal to the value of that node.
2. All values in the right subtree of a node are greater than the value of that node.

```
BST = node values at Root -> left <= Root value and node values at Root -> right >= Root value.
So , for every node if this property is statisfied then this tree is considered as Binary Search Tree (BST).

Property:- InOrder Traversal of Binary Search Tree (BST) comes out in sorted order.
```

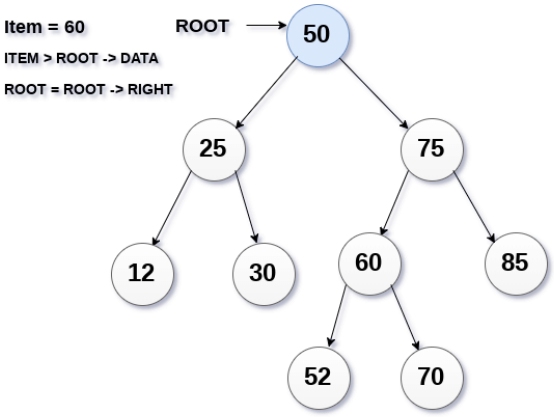


Binary Search Tree

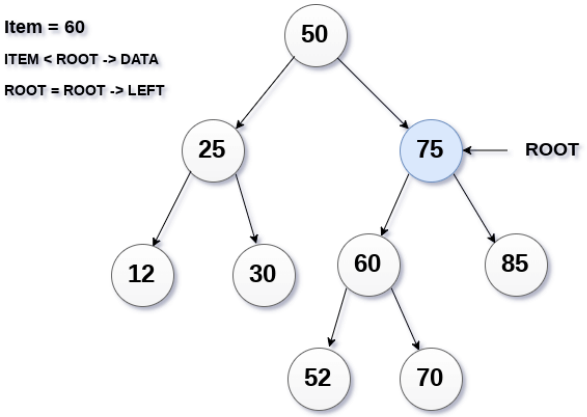
Advantages of using Binary Search Tree:

You can easily apply Binary Search on BST to search an element. So, searching in BST having time complexity o(log n) and space complexity is o(h), h is the height of the BST.

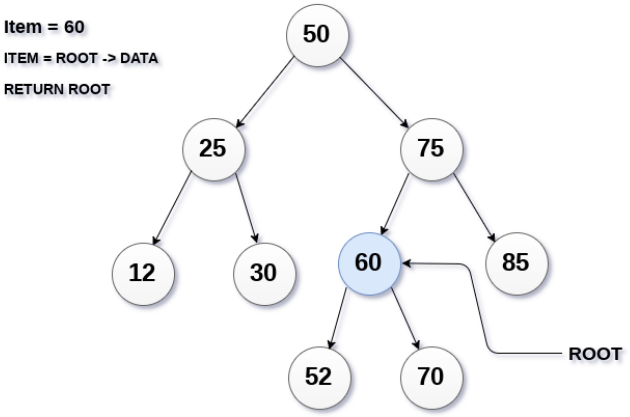
Similarly, Insertion and Deletion in BST can also be done in O(log n) time using Binary Search.



STEP 1



STEP 2



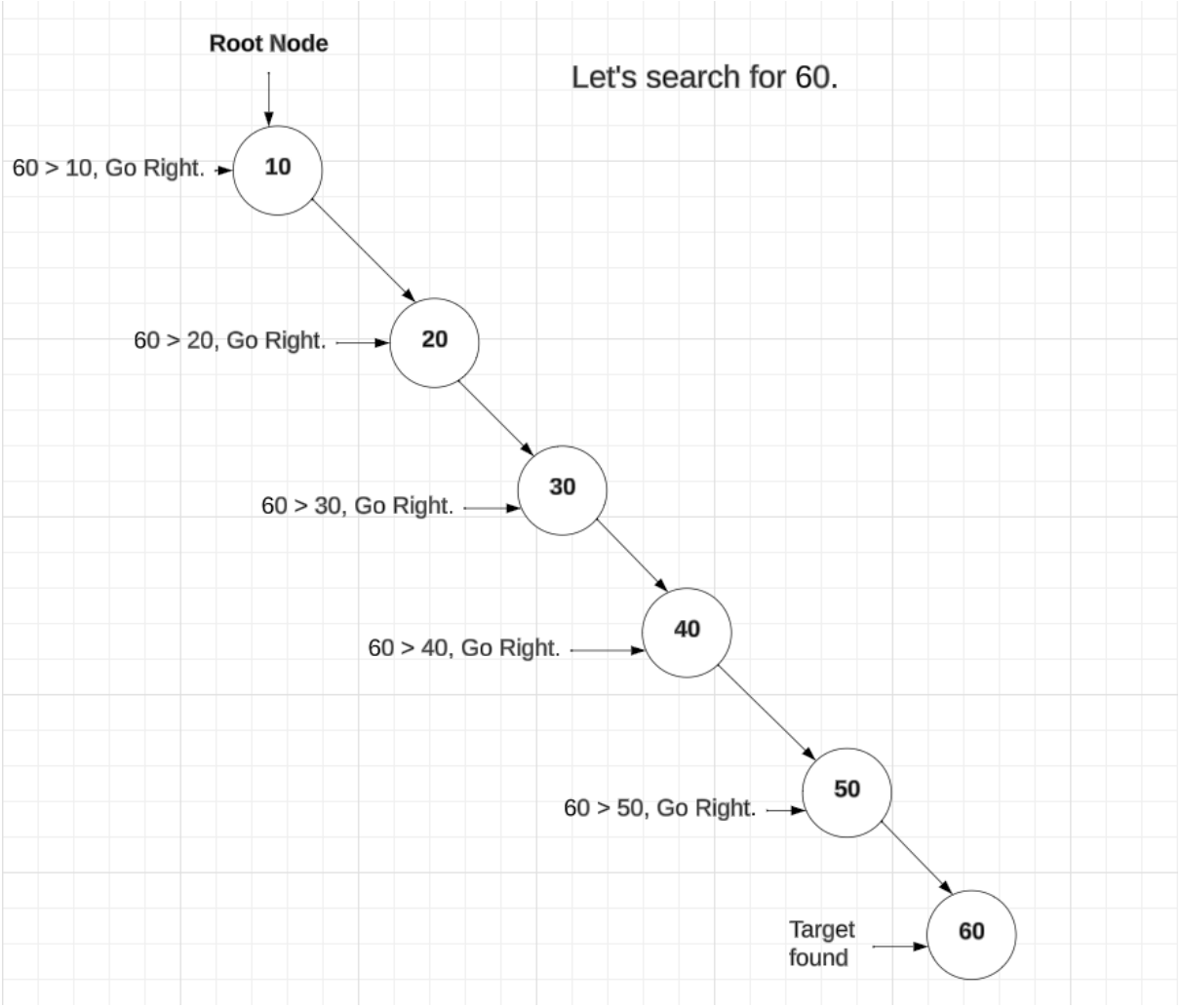
STEP 3

Note: The In-Order Traversal of Binary Search Tree (BST) comes out in sorted order.

Limitation of Binary Search Tree:

In the worst-case scenario, when the binary search tree (BST) is completely unbalanced and resembles a linked list (e.g., a skewed tree), the time complexity for searching becomes O(n), where "n" is the number of nodes in the tree.

Similarly, Insertion and Deletion in BST can also take O(n) time in words case, where "n" is the number of nodes in the tree.



How do we solve this Issue of Searching, Insertion, and Deletion in Unbalanced BST in O(n) Time Complexity?

To address the issue of unbalanced Binary Search Trees (BSTs), we have to use something which is called as Self-Balancing Binary Search Trees.

These specialized BST data structures automatically maintain balance during insertion and deletion operations, ensuring that the height of the tree remains logarithmic (O(log n)) in relation to the number of nodes. Here are two common types of self-balancing BSTs:

1. AVL Trees
2. Red-Black Trees