




DATA STRUCTURES

WENYE LI
CUHK-SZ

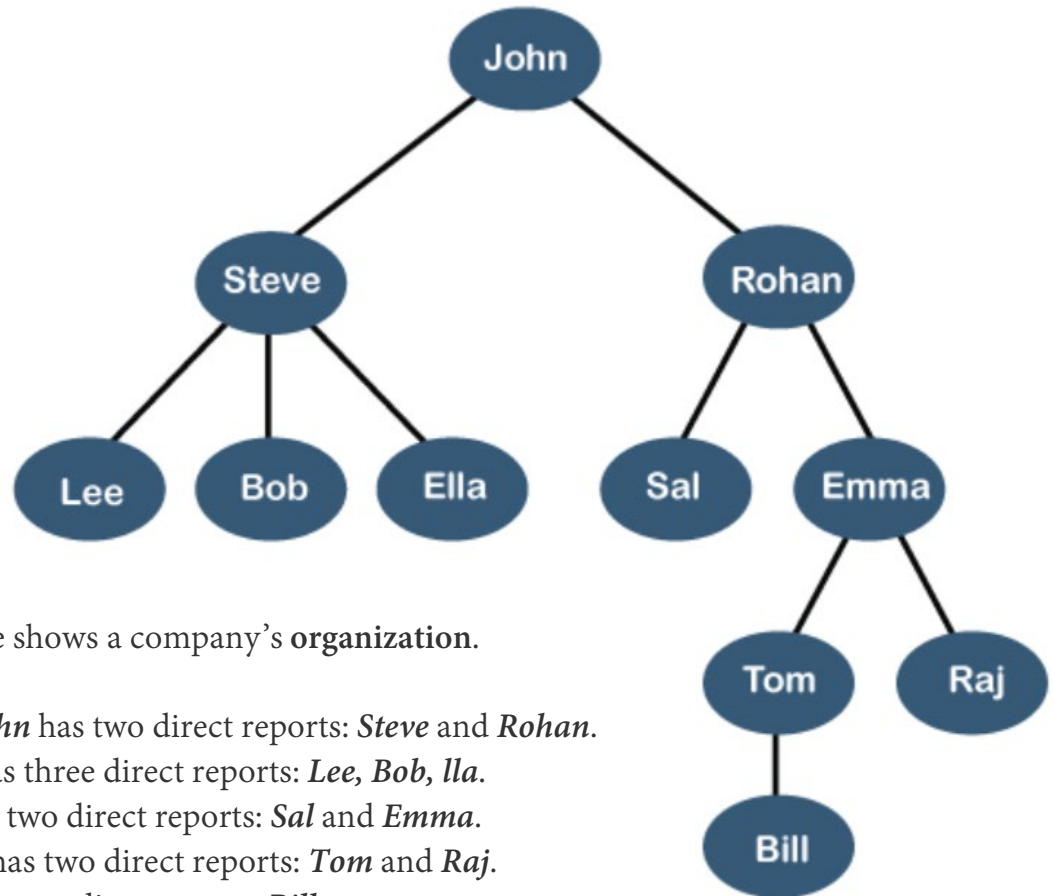
OUTLINE

- Trees
- Types of Trees
- Implementation
- Examples

- 
- Linear data structures like an array, linked list, stack and queue
 - all elements are arranged in a sequential manner.
 - Factors considered for choosing data structure
 - **What type of data needs to be stored?**
 - It might be possible that a certain data structure can be the best fit for some kind of data.
 - **Cost of operations**
 - For example, we have a simple list on which we have to perform the search operation; then, we can create an array in which elements are stored in sorted order to perform the binary search. The binary search works very fast for the simple list as it divides the search space into half.
 - **Memory usage**
 - Sometimes, we want a data structure that utilizes less memory.

TREES

- A *tree* is a data structure that represent hierarchical data.
- Suppose we want to show the employees and their positions in the hierarchical form then it can be represented as shown.



The tree shows a company's **organization**.

CEO *John* has two direct reports: *Steve* and *Rohan*.

Steve has three direct reports: *Lee*, *Bob*, *lla*.

Bob has two direct reports: *Sal* and *Emma*.

Emma has two direct reports: *Tom* and *Raj*.

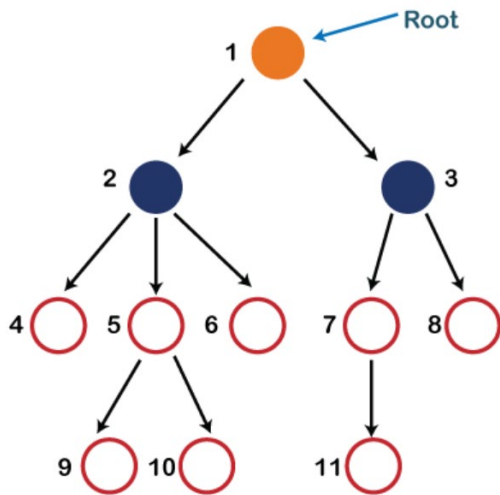
Tom has one direct report: *Bill*.

In this structure, the **root** is at the top, and its branches are moving in a downward direction. Therefore, we can say that the Tree data structure is an efficient way of storing the data in a hierarchical way.

TREES

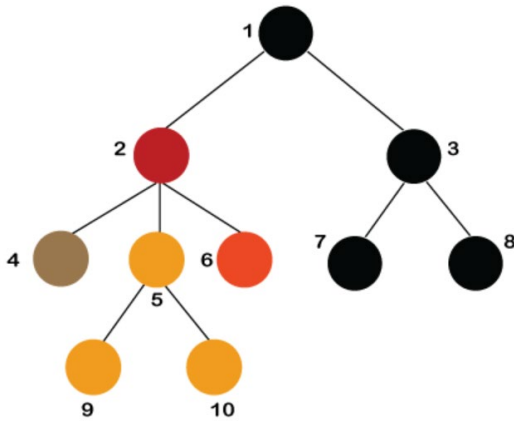
- A tree data structure is defined as a collection of objects or entities known as **nodes** that are linked together to represent or simulate hierarchy.
- A tree is a **non-linear** data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a tree are arranged in multiple levels.
- In a tree, the topmost node is known as a **root** node.
- Each node contains some **data**, which can be of any type.
- Each node contains the link or reference of other nodes that can be called **children**.

BASIC TERMS



- **Root:** The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent.
- **Child node:** If the node is a descendant of any node, then the node is known as a child node.
- **Parent:** If the node contains any sub-node, then that node is said to be the parent of that sub-node.
- **Sibling:** The nodes that have the same parent are known as siblings.
- **Leaf Node:** The node of the tree, which doesn't have any child node, is called a leaf node. A leaf node is the bottom-most node of the tree. Leaf nodes can also be called external nodes.
- **Internal nodes:** A node has at least one child node known as an internal node.
- **Ancestor node:** An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors, e.g., nodes 1, 2, and 5 are the ancestors of node 10.
- **Descendant:** The immediate successor of the given node is known as a descendant of a node, e.g., 10 is the descendant of node 5.

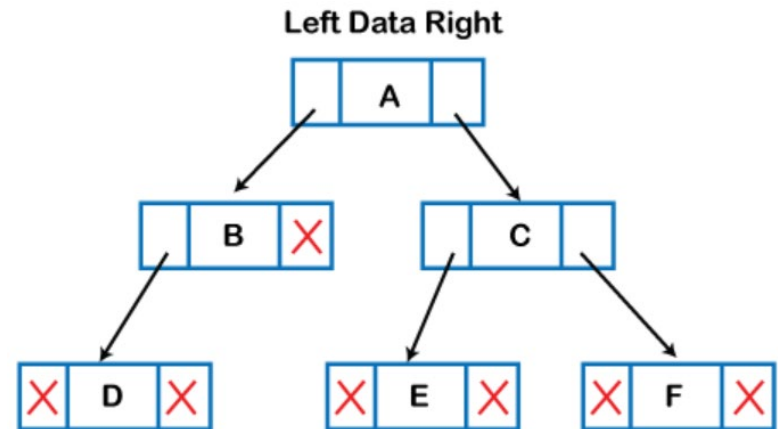
PROPERTIES



- **Recursive data structure:** A tree can be defined as recursively. The root node contains a link to all the roots of its subtrees. The left subtree is shown in yellow, and the right subtree is shown in red. The left subtree can be further split into subtrees shown in three different colors. Recursion means reducing something in a self-similar manner.
- **Number of edges:** If there are n nodes, then there would $n-1$ edges. Each arrow in the structure represents the link or path. Each node, except the root node, will have at least one incoming link known as an edge. There would be one link for the parent-child relationship.
- **Depth of node x :** the length of the path from the root to node x , i.e., the number of edges between the root node and node x . The root node has 0 depth.
- **Height of node x :** the longest path from the node x to the leaf node.

IMPLEMENTATION

- A tree can be created by creating the nodes dynamically with the help of the pointers.
- A node contains three fields. The second field stores the data; the first field stores the address of the left child, and the third field stores the address of the right child.
- Note: This structure can only be defined for **binary trees** because a binary tree have at most two children, and generic trees can have more than two children. The node structure for generic trees would be different as compared to the binary tree.



APPLICATIONS

- **Storing naturally hierarchical data:** The file system stored on the disc drive, the file and folder are in the form of the naturally hierarchical data and stored in the form of trees.
- **Organize data:** It is used to organize data for efficient insertion, deletion and searching. For example, a binary tree has a $\log N$ time for searching an element.
- **Trie:** It is a special kind of tree that is used to store the dictionary, fast and efficient for dynamic spell checking.
- **Heap:** It is a tree data structure implemented using arrays. It is used to implement priority queues.
- **B-Tree and B+Tree:** B-Tree and B+Tree are the tree data structures used to implement indexing in databases.
- **Routing table:** The tree data structure is used to store the data in routing tables in the routers.

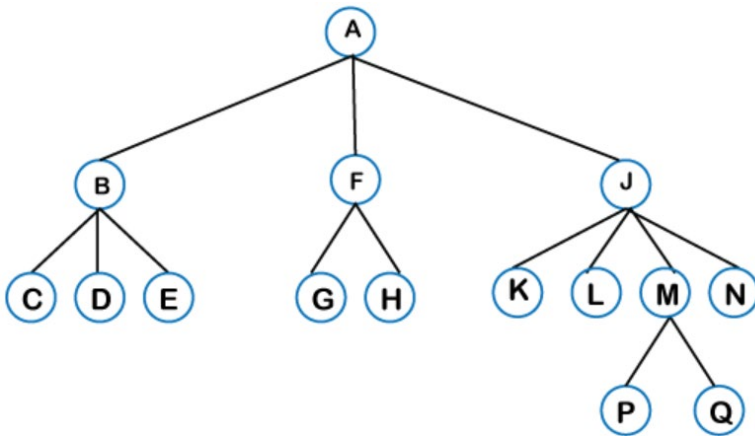
OUTLINE

- Trees
- Types of Trees
- Implementation
- Examples

TYPES OF TREES

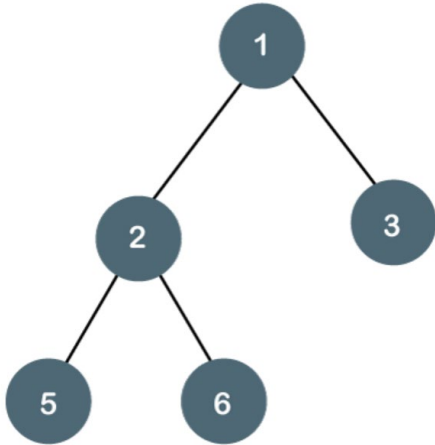
- General Tree
- Binary Tree
- Binary Search Tree
- AVL Tree
- Red-Black Tree
- <https://www.javatpoint.com/tree>

GENERAL TREE

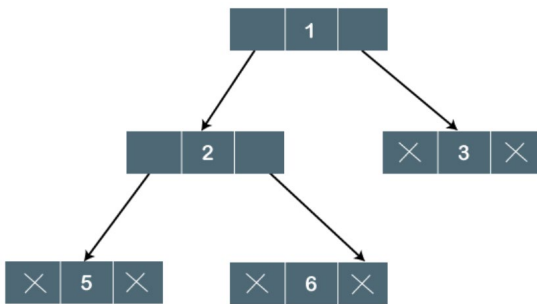


- A node can have either 0 or maximum n number of nodes. There is no restriction imposed on the degree of the node (the number of nodes that a node can contain). The topmost node is known as a **root** node. The children of the parent node are known as **subtrees**.
- There can be n number of subtrees in a general tree. In the general tree, the subtrees are unordered as the nodes in the subtree cannot be ordered.
- Every non-empty tree has a downward edge, and these edges are connected to the nodes known as child nodes. The root node is labeled with level 0 . The nodes that have the same parent are known as siblings.

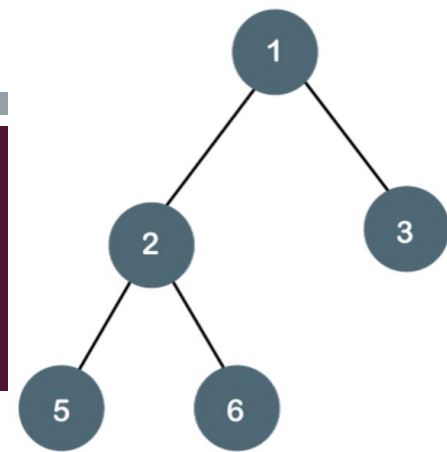
BINARY TREE



- Binary tree means that the node can have maximum two children. Each node can have either 0, 1 or 2 children.
- In the example,
 - Node 1 contains left and right pointers pointing to left and right nodes respectively.
 - Node 2 contains both left and right nodes.
 - Nodes 3, 5 and 6 are leaf nodes; all these nodes contain NULL pointer on both left and right parts.



PROPERTIES OF BINARY TREE



- At each level of i , the maximum number of nodes is 2^i .
- The **height of a tree** is defined as the longest path from the root node to the leaf node.
 - The maximum number of nodes at height 3 is $(1+2+4+8) = 15$.
 - In general, the maximum number of nodes possible at height h is $(2^0 + 2^1 + 2^2 + \dots + 2^h) = 2^{h+1} - 1$.
- The minimum number of nodes possible at height h is equal to $h+1$.
- If the number of nodes is minimum, then the height of the tree would be maximum.
- If the number of nodes is maximum, then the height of the tree would be minimum.

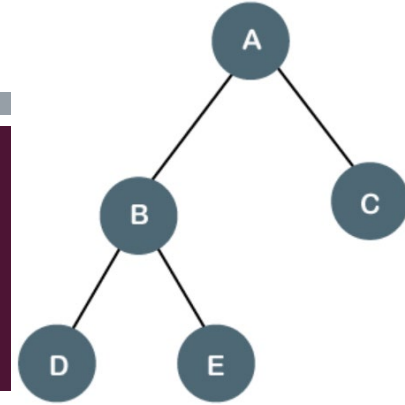
PROPERTIES OF BINARY TREE

- Suppose a binary tree has ' n ' nodes. The minimum height can be computed as:
 - As we know that, $n = 2^{h+1} - 1$ and $n+1 = 2^{h+1}$
 - Taking log on both the sides,
 - $\log_2(n+1) = \log_2(2^{h+1})$
 - $\log_2(n+1) = h+1$
 - $h = \log_2(n+1) - 1$
- The maximum height can be computed as:
 - As we know that, $n = h+1$
 - $h = n-1$

TYPES OF BINARY TREE

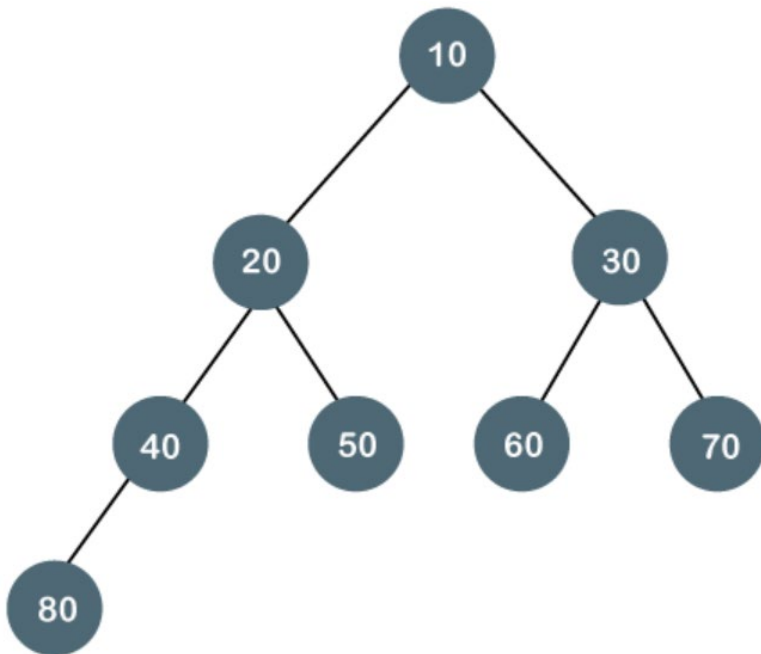
- Full/ proper/ strict Binary tree
- Complete Binary tree
- Perfect Binary tree
- Balanced Binary tree

FULL BINARY TREE



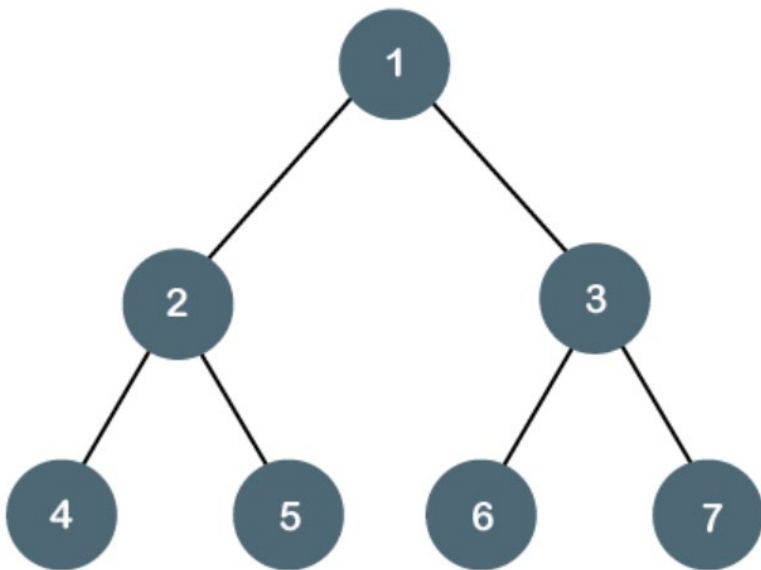
- Each node contains either zero or two children.
- The number of leaf nodes is equal to the number of internal nodes plus 1.
- The maximum number of nodes is $2^{h+1} - 1$.
- The minimum number of nodes is $2^*h + 1$.
- The minimum height of the full binary tree is $\log_2(n+1) - 1$.
- The maximum height of the full binary tree can be computed as:
 - $n = 2^*h + 1$
 - $n-1 = 2^*h$
 - $h = (n-1)/2$

COMPLETE BINARY TREE



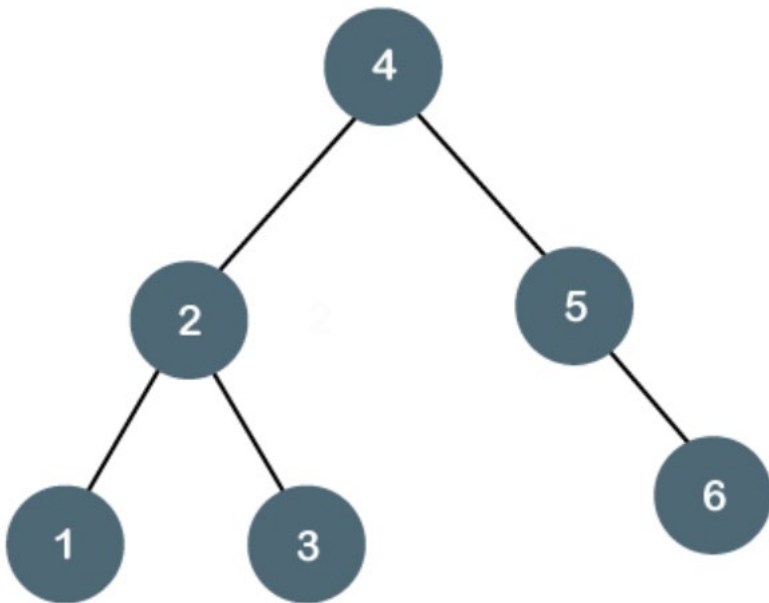
- All nodes are completely filled except the last level. In the last level, all nodes must be as left as possible.
- The maximum number of nodes in complete binary tree is $2^{h+1} - 1$.
- The minimum number of nodes in complete binary tree is 2^h .
- The minimum height of a complete binary tree is $\log_2(n+1) - 1$.
- The maximum height of a complete binary tree is $\log_2(n)$.

PERFECT BINARY TREE



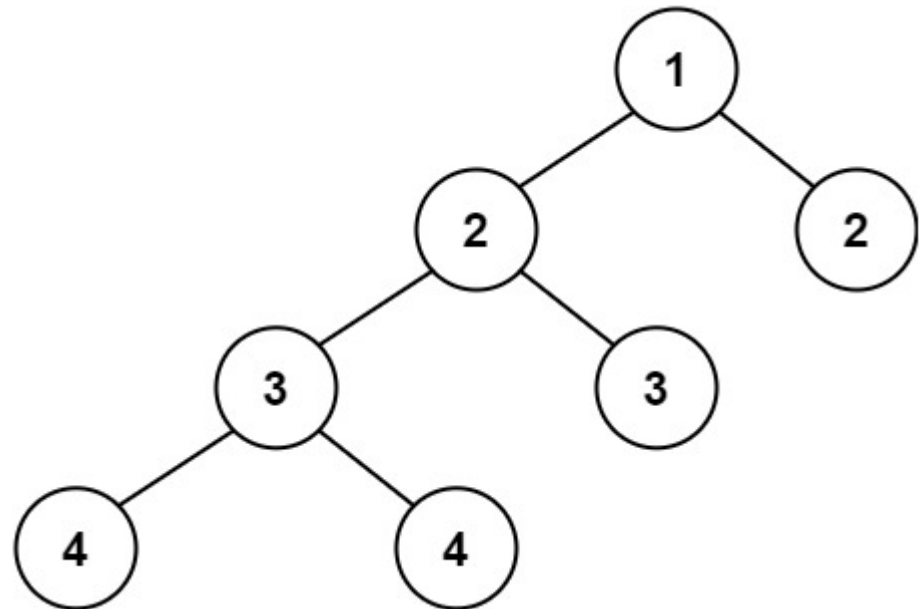
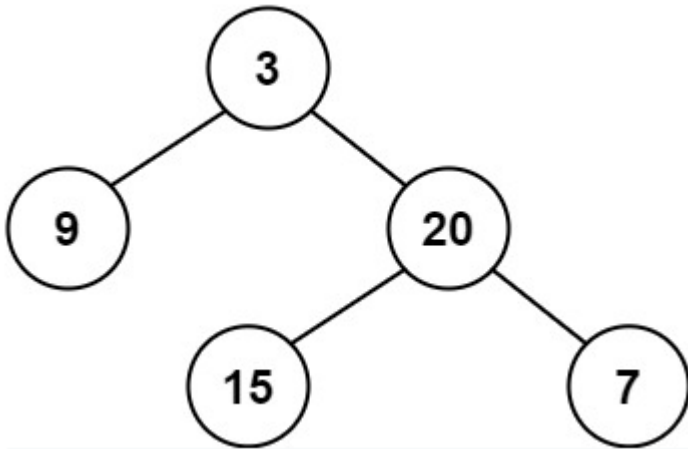
- All internal nodes have 2 children, and all leaf nodes are at the same level.
- All perfect binary trees are complete binary trees as well as full binary trees. But vice versa is not true, i.e., all complete binary trees and full binary trees are the perfect binary trees.

BALANCED BINARY TREE

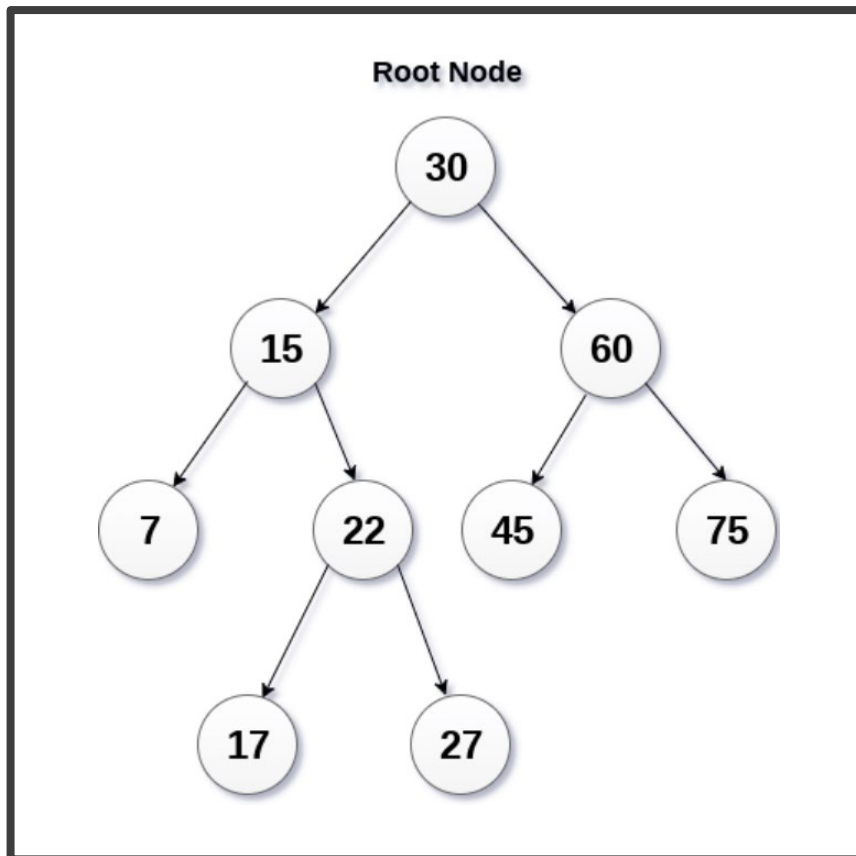


- A binary tree in which the left and right subtrees of every node differ in height by no more than 1.
- E.g., *AVL* and *Red-Black trees* are balanced binary tree.

BALANCED OR NOT?



BINARY SEARCH TREE



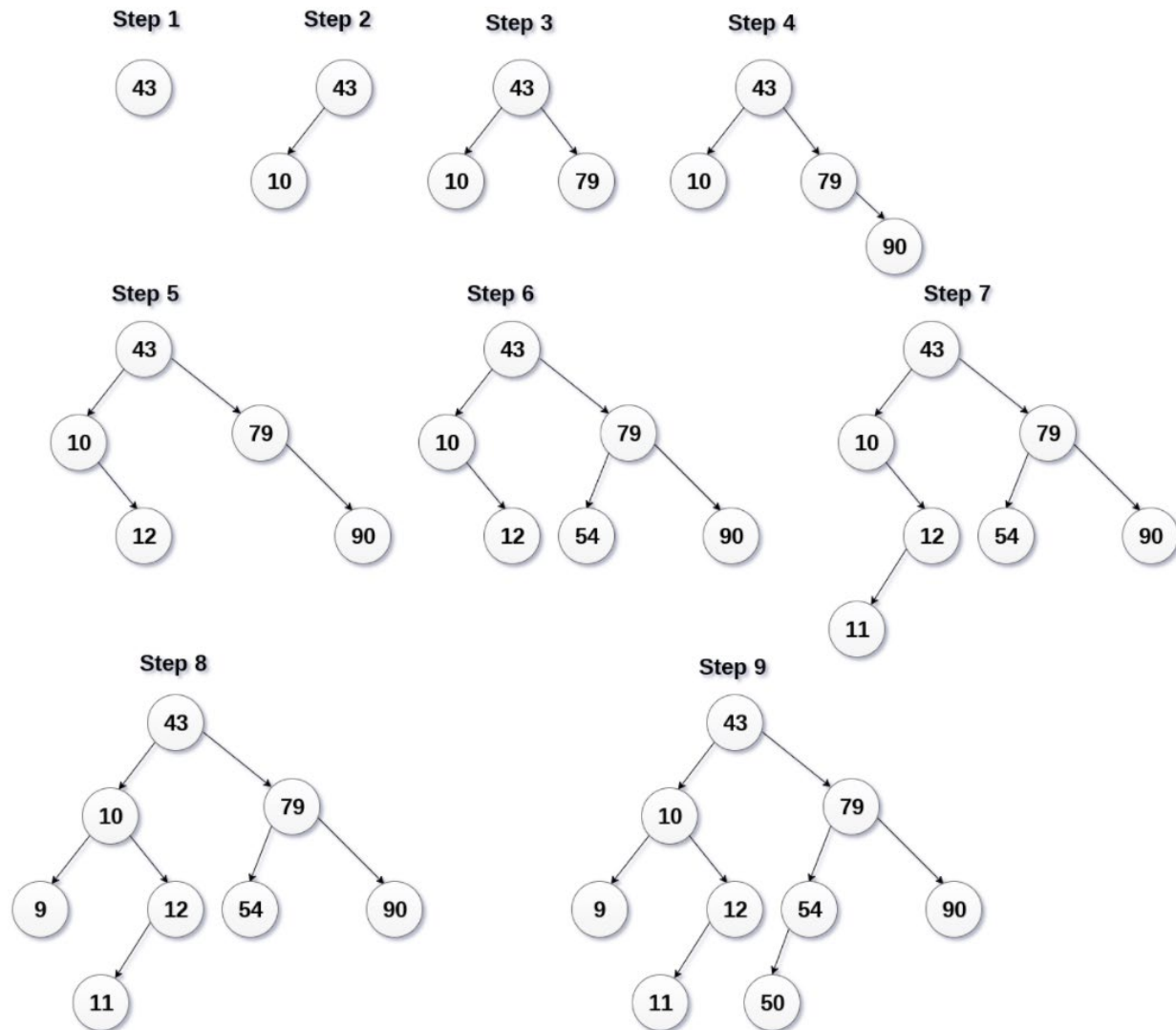
- A class of binary trees, in which the nodes are arranged in a specific order, also called ordered binary tree.
- The value of all nodes in the left sub-tree is less than the value of the root.
- The value of all nodes in the right sub-tree is greater than or equal to the value of the root.
- The rule is recursively applied to all left and right sub-trees of the root.

ADVANTAGES OF BST

- Searching is very efficient in a BST since, we get a hint at each step, about which sub-tree contains the desired element.
- BST is considered as efficient data structure in comparison to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes $O(\log_2 n)$ time. In worst case, the time it takes to search an element is $O(n)$.
- It also speed up the insertion and deletion operations as comparison to that in array and linked list.

CREATE A BST

- 43, 10, 79, 90, 12, 54, 11, 9, 50
- Insert 43 into the tree as the root of the tree.
- Read next element. If it is less than the root node, insert it as the root of the left sub-tree.
- Otherwise, insert it as the root of the right sub-tree.





THANKS