Leaning Objective:

In this Module you will be learning the following:

- Binary Search Tree and its Operations

Introduction:

Binary Search Tree is a binary tree in which every node contains only smaller values in its left subtree and only larger values in its right subtree.

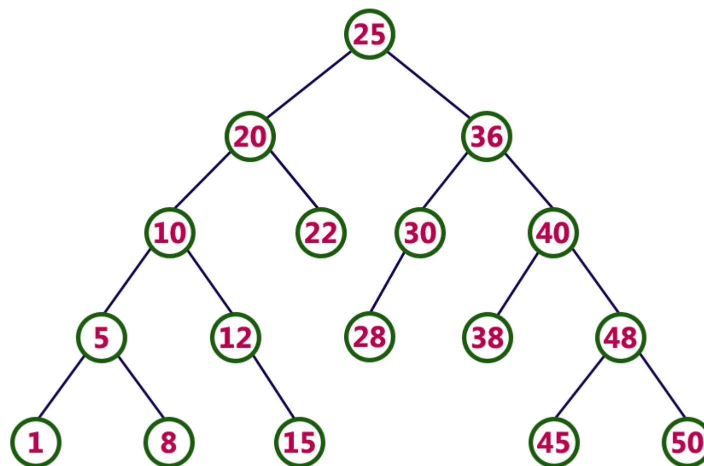
Material:

A BST is a Binary Tree in symmetric order, here symmetric order means

- The left subtree of a node contains only nodes with keys less 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 each must also be a binary search tree.
- There must be no duplicate nodes.
- The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

Example

The following tree is a Binary Search Tree. In this tree, left subtree of every node contains nodes with smaller values and right subtree of every node contains larger values.



Every Binary Search Tree is a binary tree but all the Binary Trees need not to be binary search trees.

Binary Search Tree Representations

A binary search tree data structure is represented using two methods. Those methods are as follows...

1. **Array Representation**
2. **Linked List Representation**

Construct a Binary Search Tree by inserting the following sequence of numbers using Array and Linked List Reprsentaion... **10,12,5,4,20,8,7,15,13,11,22**

1. Array Representation

In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree. Consider the above example of binary tree and it is represented as follows...

10	5	12	4	8	11	20			7				15	22																	13

To represent a binary tree of depth **'n'** using array representation, we need one dimensional array with a maximum size of $\mathbf{2^{n+1} - 1}$.

Each node v is stored at index i defined as follows:

- If v is the root, i = 1
- The left child of v is in position 2i
- The right child of v is in position 2i + 1
- The parent of v is in position???

2. Linked List Representation

We use double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address. In this linked list representation, a node has the following structure...

Left Child Address	Data	Right Child Address

The above example of binary tree represented using Linked list representation is shown as follows...

Operations on a Binary Search Tree

1. **Search**
2. **Insertion**
3. **Deletion**

Search Operation in BST

In a binary search tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed as follows...

- **Step 1:** Read the search element from the user
- **Step 2:** Compare, the search element with the value of root node in the tree.
- **Step 3:** If both are matching, then display "Given node found!!!" and terminate the function
- **Step 4:** If both are not matching, then check whether search element is smaller or larger than that node value.
- **Step 5:** If search element is smaller, then continue the search process in left subtree.
- **Step 6:** If search element is larger, then continue the search process in right subtree.
- **Step 7:** Repeat the same until we found exact element or we completed with a leaf node
- **Step 8:** If we reach to the node with search value, then display "Element is found" and terminate the function.
- **Step 9:** If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

Insertion Operation in BST

In a binary search tree, the insertion operation is performed with **O(log n)** time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

- **Step 1:** Create a newNode with given value and set its **left** and **right** to **NULL**.
- **Step 2:** Check whether tree is Empty.
- **Step 3:** If the tree is **Empty**, then set set **root** to **newNode**.
- **Step 4:** If the tree is **Not Empty**, then check whether value of newNode is **smaller** or **larger** than the node (here it is root node).
- **Step 5:** If newNode is **smaller** than **or equal** to the node, then move to its **left** child. If newNode is **larger** than the node, then move to its **right** child.
- **Step 6:** Repeat the above step until we reach to a **leaf** node (e.i., reach to NULL).
- **Step 7:** After reaching a leaf node, then isert the newNode as **left child** if newNode is **smaller or equal** to that leaf else insert it as **right child**.

Deletion Operation in BST

In a binary search tree, the deletion operation is performed with **O(log n)** time complexity. Deleting a node from Binary search tree has follwing three cases...

- **Case 1: Deleting a Leaf node (A node with no children)**
- **Case 2: Deleting a node with one child**
- **Case 3: Deleting a node with two children**

Case 1: Deleting a leaf node

- **Step 1: Find** the node to be deleted using **search operation**
- **Step 2:** Delete the node using **free** function (If it is a leaf) and terminate the function.

Case 2: Deleting a node with one child

We use the following steps to delete a node with one child from BST...

- **Step 1: Find** the node to be deleted using **search operation**
- **Step 2:** If it has only one child, then create a link between its parent and child nodes.
- **Step 3:** Delete the node using **free** function and terminate the function.

Case 3: Deleting a node with two children

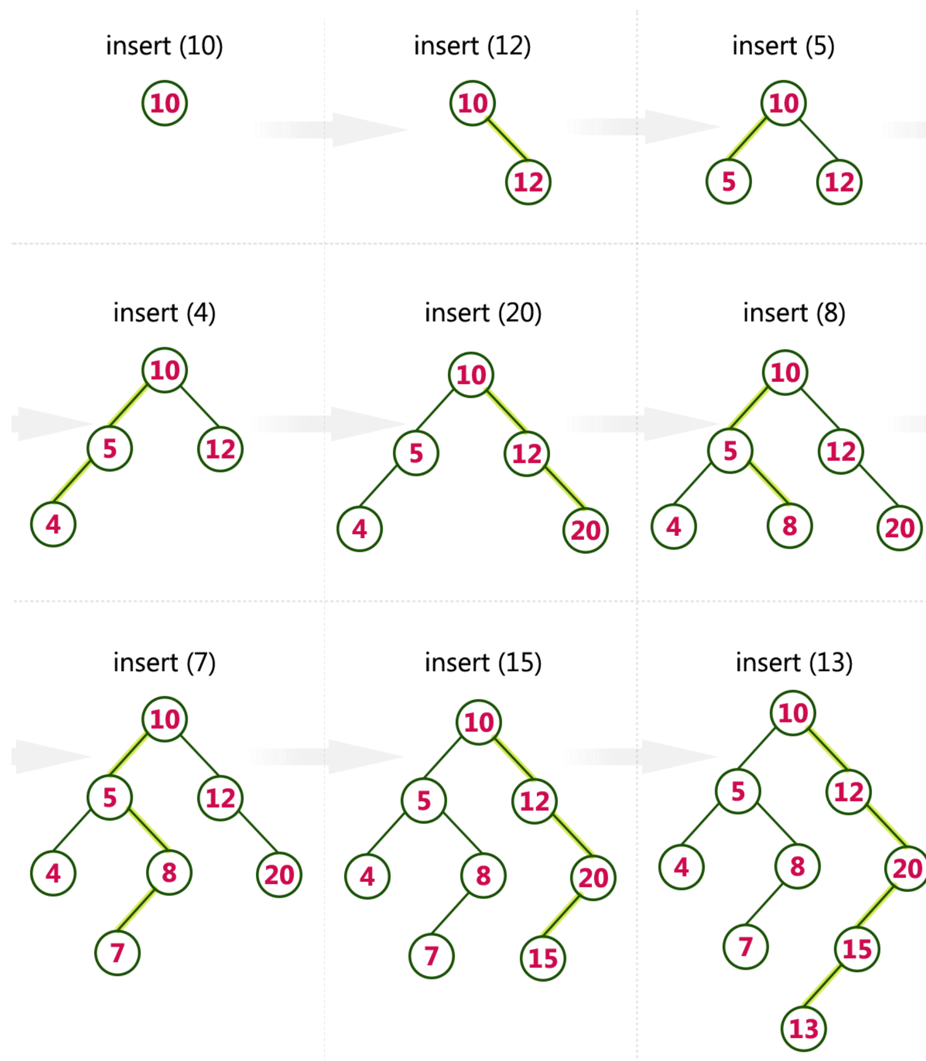
We use the following steps to delete a node with two children from BST...

- **Step 1: Find** the node to be deleted using **search operation**
- **Step 2:** If it has two children, then find the **largest** node in its **left subtree** (OR) the **smallest** node in its **right subtree**.
- **Step 3: Swap** both **deleting node** and node which found in above step.
- **Step 4:** Then, check whether deleting node came to **case 1** or **case 2** else goto steps 2
- **Step 5:** If it comes to **case 1**, then delete using case 1 logic.
- **Step 6:** If it comes to **case 2**, then delete using case 2 logic.
- **Step 7:** Repeat the same process until node is deleted from the tree.

Example

Construct a Binary Search Tree by inserting the following sequence of numbers...

10, 12, 5, 4, 20, 8, 7, 15 and 13



Problem Sets:

1. Write a C Program to implement the following operations on BST using Arrays and Likened List?

   Basic Operations

   - Insert an element into a tree
   - Delete an element into a tree
   - Searching an element
   - Traversing the Tree

   Auxiliary Operations

   - Finding Size of the Tree
   - Finding height of the Tree
   - Finding the level which has maximum sum