Batch: B2	Roll No.: 16010124107
Experiment / assignment / tutorial No. 6	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of BST & Binary tree traversal techniques.

Objective: To Understand and Implement Binary Search Tree along with Insertion, Deletion and Preorder, Postorder and Inorder Traversal Techniques.

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.geeksforgeeks.org/binary-tree-data-structure/
5. https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html

Abstract:

A tree is a non- linear data structure used to represent hierarchical relationship existing among several data items. It is a finite set of one or more data items such that, there is a special data item called the root of the tree. Its remaining data items are partitioned into

number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.

A binary tree is a finite set of nodes. It is either empty or It consists a node called root with two disjoint binary trees-Left subtree, Right subtree. The Maximum degree of any node is 2.

A Binary Search Tree is a node-based binary tree data structure in which 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 each must also be a binary search tree.

Related Theory: -

Algorithm: Preorder Traversal of BST

1. Start at the root node
2. Print this to console
3. Move to the left pointer of the node
4. Repeat steps 2 and 3 till a NULL ptr is achieved
5. Then, move to the right pointers of all till root node is visited
6. Then, move to the right subtree
7. Step 2,3,4 repeated
8. Repeat step 5 recursivey
9. stop

Algorithm:Postorder Traversal of BST

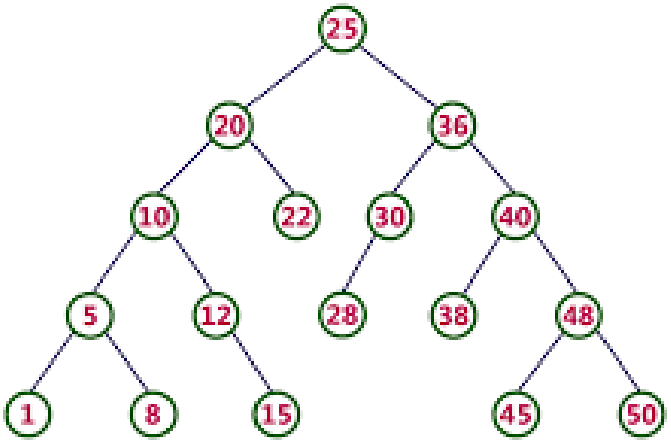
1. Start at the root node
2. Move to the left pointer of the node
3. Repeat step 2 till a NULL ptr is achieved
4. Move to the right pointer of the node
5. Repeat steps 2,3 on right subtree
6. Print this to console
7. Repeat whole process recursively
8. Stop

Algorithm: Inorder Traversal of BST

1. Start at the root node
2. Move to the left pointer of the node
3. Repeat step 2 till a NULL ptr is achieved

4. Print this to console
5. Then, move to the right pointer of the node
6. Repeat steps 2,3,4 on right subtree
7. Repeat whole process recursively
8. Stop

An example BST :



Preorder Traversal:
NLR
25,20,10,5,1,8,12,15,22,36,30,28,40,38,48,45,50

Postorder Traversal:
LRN
1,8,5,15,12,10,22,20,28,30,38,45,50,48,40,36,25

Inorder Traversal:
LNR
1, 5, 8, 10, 12, 15, 20, 22, 25, 28, 30, 36, 38, 40, 45, 48, 50

Algorithm for insertion into a BST:

- Create a New Node: Create a new node containing the value to be inserted.
- Handle Empty Tree: If the BST is empty (root is NULL), the new node becomes the root of the tree.
- Traverse the Tree:
    - Start at the root node of the BST.
    - Compare the value to be inserted with the current node's value.
    - If the value to be inserted is less than the current node's value, move to the left child.
    - If the value to be inserted is greater than the current node's value, move to the right child.
    - Repeat this comparison and movement until a NULL child pointer is encountered. This NULL pointer indicates the correct position for the new node.
- Insert the Node:
    - Once a NULL child pointer is reached, attach the new node as either the left or right child of the parent node, based on the final comparison. If the new node's value is less than the parent's value, it becomes the left child; otherwise, it becomes the right child.

Program source code for Implementation of BST insertion & Binary tree traversal techniques :

```
#include <iostream>

using namespace std;


struct Node {

    int data;

    Node *left, *right;


    Node(int value) {

        data = value;

        left = right = nullptr;

    }

};


Node* insert(Node* root, int value) {

    if (root == nullptr)

        return new Node(value);


    if (value < root->data)

        root->left = insert(root->left, value);

    else if (value > root->data)

        root->right = insert(root->right, value);
```

```
    return root;
}

void inorder(Node* root) {
    if (root != nullptr) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

void preorder(Node* root) {
    if (root != nullptr) {
        cout << root->data << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(Node* root) {
    if (root != nullptr) {
        postorder(root->left);
        postorder(root->right);
        cout << root->data << " ";
```

```
    }
}

int main() {
    Node* root = nullptr;
    int n, val;

    cout << "Enter number of nodes: ";
    cin >> n;

    cout << "Enter values: ";
    for (int i = 0; i < n; i++) {
        cin >> val;
        root = insert(root, val);
    }

    cout << "\nInorder Traversal: ";
    inorder(root);

    cout << "\nPreorder Traversal: ";
    preorder(root);

    cout << "\nPostorder Traversal: ";
    postorder(root);
```

```
    return 0;
}
```

Output Screenshots for Each Operation:

```
Enter number of nodes: 5
Enter values: 1
2
3
4
5

Inorder Traversal: 1 2 3 4 5
Preorder Traversal: 1 2 3 4 5
Postorder Traversal: 5 4 3 2 1
PS C:\Users\syeda\OneDrive\Desktop\personal>
```

Conclusion:-

A binary search tree is an implementation of a tree wherein nodes are arranged in order of their size. If the root node is x and the next element to be added is more than it, it goes to the right. If it is less than it, it goes to the left and so on. The in-order traversal of this type of tree is always sorted ascending order.

PostLab Questions:

1) Write an ADT for tree data structure

Tree ADT

1. Data Objects:
2. A finite set of elements (called nodes).
3. One node is the root.
4. Remaining nodes are divided into disjoint sets, each forming a subtree.

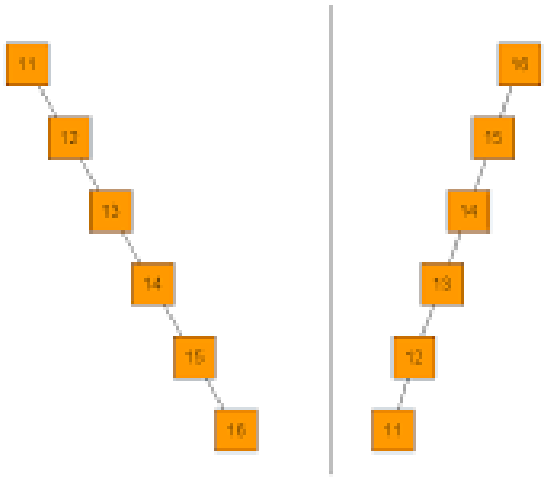
Operations:

1. createTree() : Create an empty tree.
2. isEmpty(T) : Check if tree T is empty.
3. insert(T, x, p) : Insert node x as child of parent node p in tree T.
4. deleteNode(T, x) : Delete node x and its subtree from T.
5. search(T, x) : Check if element x exists in tree T.
6. root(T) : Return the root of tree T.

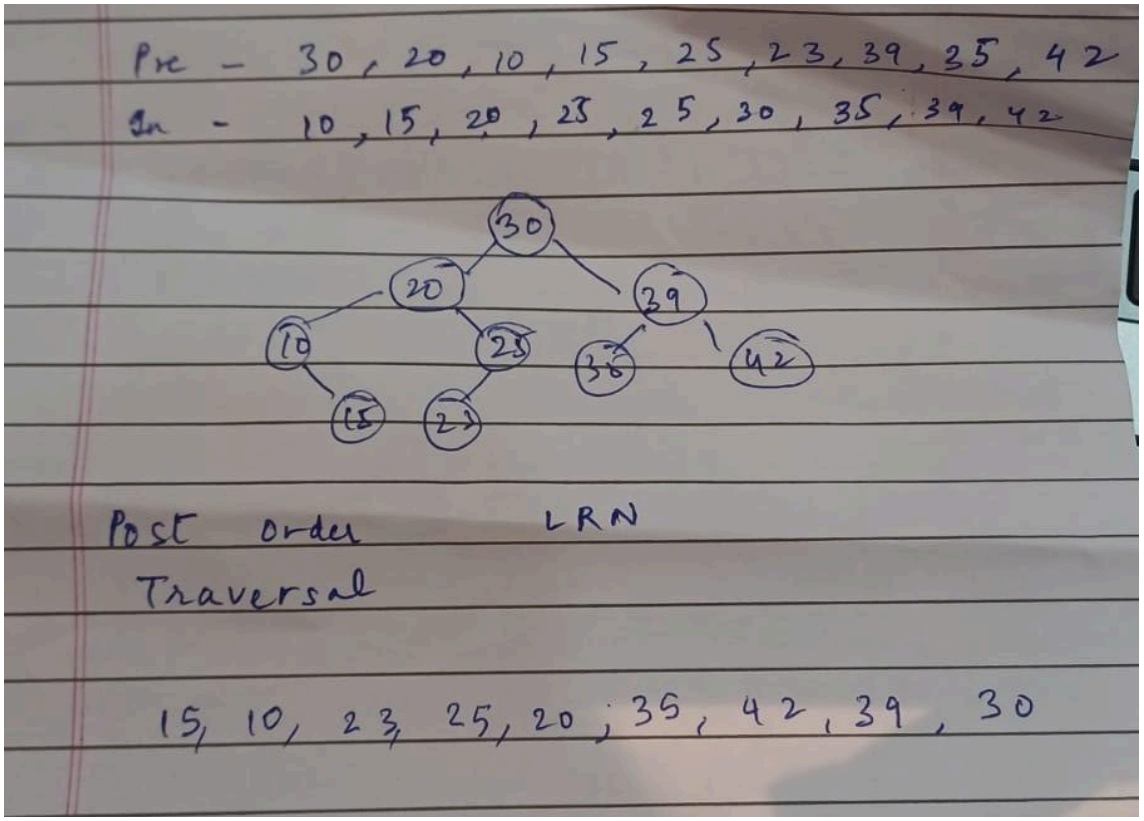
7. parent(T, x) : Return parent of node x.
8. children(T, x) : Return children of node x.
9. traverse(T, order) : Traverse tree T in a given order (Preorder, Inorder, Postorder, Level-order).
10. height(T) : Return height of tree T.
11. size(T) : Return total number of nodes in T.

2) What are the consequences of inserting elements in ascending or descending order into a BST?

In a binary search tree,
If you insert ascending order, tree becomes skewed right.
If you insert descending order, tree becomes skewed left.
Here, the left tree is ascending and right is inserted descending.



3) The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Construct a balanced Binary Search Tree and perform the Postorder Traversal for the same.


Pre - 30, 20, 10, 15, 25, 23, 39, 35, 42
In - 10, 15, 20, 23, 25, 30, 35, 39, 42
30
20
39
10
25
35
42
15
23
Post order Traversal
LRN
15, 10, 23, 25, 20, 35, 42, 39, 30