Course Name:	**Data Structures Laboratory**	**Semester:**	**III**
Date of Performance:	**22 / 09 / 2025**	**Batch No:**	**A1**
Faculty Name:	**Prof. Sushma Kadge**	**Roll No:**	**20**
Faculty Sign & Date:		**Grade/Marks:**	**/25**

Experiment No: 5
Title: Binary Search Tree Traversal

Aim and Objective of the Experiment:

Write a program for binary search tree (BST) traversal.

COs to be achieved:

CO1 : Understand and implement the different data structures used in problem solving
CO2: Apply linear and non-linear data structure in application development

Books/Journals/Websites referred:

1. GeeksForGeeks

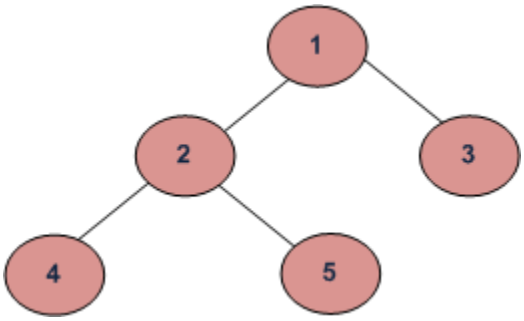
Tools required:

DEV C/C++ compiler/ Code blocks C compiler

Theory:

Binary search tree (BST) is a special kind of binary tree where each node contains-
1. Only larger values in its right subtree.
2. Only smaller values in its left subtree.

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Stepwise-Procedure:

Inorder(tree)
1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Preorder(tree)
1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Postorder(tree)
1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.

Implementation details:

- BST created using `struct node`.
- Recursive functions implemented for Inorder, Preorder, and Postorder traversal.
- Smallest and largest nodes found by traversing leftmost and rightmost nodes.

Importance of Approach:
Recursion provides a simple, natural approach for tree traversal, eliminating the need for complex iterative stack management.

C/C++ Code implemented:

```c
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int value)
{
    if (root == NULL)
        return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}

void inorder(struct Node* root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node* root)
{
    if (root != NULL)
    {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root)
{
```

```
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

struct Node* findMin(struct Node* root)
{
    while (root && root->left != NULL)
        root = root->left;
    return root;
}

struct Node* findMax(struct Node* root)
{
    while (root && root->right != NULL)
        root = root->right;
    return root;
}

int main()
{
    struct Node* root = NULL;
    int n, value;
    printf("Enter number of nodes: \n");
    scanf("%d", &n);
    printf("Enter node values: \n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &value);
        root = insert(root, value);
    }

    printf("Inorder traversal:\n");
    inorder(root);
    printf("\n");
    printf("Preorder traversal:\n");
    preorder(root);
    printf("\n");
    printf("Postorder traversal:\n");
    postorder(root);
    printf("\n");

    struct Node* minNode = findMin(root);
    struct Node* maxNode = findMax(root);
    printf("Minimum value:%d \n", minNode->data);
    printf("Maximum value:%d \n", maxNode->data);

    return 0;
}
```

Output:

```
D:\Programming\Java\New folder>gcc BST.c -o BST

D:\Programming\Java\New folder>BST
Enter number of nodes:
5
Enter node values:
12
32
16
9
27
Inorder traversal:
9 12 16 27 32
Preorder traversal:
12 9 32 16 27
Postorder traversal:
9 27 16 32 12
Minimum value:9
Maximum value:32

D:\Programming\Java\New folder>
```

Conclusion:

- Successfully implemented BST and demonstrated all three traversal methods.
- Inorder traversal confirms BST property as it outputs sorted values.
- Smallest and largest nodes identified efficiently.
- Recursion makes the traversal logic concise and clear.

Post lab:

For the binary search tree created above, find the smallest and largest node in BST.

- Included in the code

Signature of faculty in-charge with Date: