**Experiment 3**

**Experiment Name:** Binary Search Tree Operations (Recursive) a. Insert a node in BST

b. Delete a node from BST

c. Inorder traversal

**Objectives:**

• To understand the structure and properties of a Binary Search Tree (BST).

• To implement recursive functions for insertion and deletion of nodes in a BST.

• To perform inorder traversal using recursion and verify that it produces elements in sorted order.

• To develop problem-solving skills through recursive implementation of tree-based operations.

• To analyze the role of recursion in managing hierarchical data structures efficiently.

**Prerequisites:**

• Basic understanding of C programming language.

• Familiarity with recursion and function calls.

• Knowledge of control structures (if, else, loops).

• Understanding of pointers and dynamic memory allocation. • Fundamental concepts of tree data structures and Binary Search Tree (BST) properties.

**Key Terms:**

• Binary Search Tree (BST): A binary tree in which the left child contains values smaller than the root, and the right child contains values greater than the root.

• Node: The basic unit of a tree containing data and pointers to its left and right children.

• Root Node: The topmost node of the BST from which all other nodes branch out.

• Leaf Node: A node with no left or right child.

• Insertion: Adding a new node into the BST while maintaining its ordering property.

• Deletion: Removing a node from the BST by handling three cases – leaf node, node with one child, and node with two children.

• Traversal: Visiting all the nodes of a tree in a specific order (in this experiment, inorder traversal which gives sorted output).

• Recursion: A programming technique where a function calls itself to solve smaller instances of a problem.

• Pointer: A variable in C that stores the address of another variable, used for dynamic memory allocation in trees.

**Apparatus/Tools Required:**

• Computer system with minimum 2 GB RAM.

• Operating System: Windows / Linux.

• C Compiler (GCC / Turbo C / MinGW).

• Integrated Development Environment (IDE) such as Code::Blocks / Dev-C++ / Turbo C / VS Code.

• Text editor (Notepad / gedit / Sublime Text) for writing code. • Printer/Notebook for recording observations**.**

**Theory and Application:**

A Binary Search Tree (BST) is a special type of binary tree where each node contains a key, and the tree is organized in such a way that:

• All keys in the left subtree are smaller than the root key. • All keys in the right subtree are greater than the root key. • Both subtrees are themselves BSTs.

This ordering property allows efficient searching, insertion, and deletion of elements. Since the structure of a tree is naturally recursive, operations on BST can be implemented using recursive functions.

• In Insertion, the key is compared with the root and placed in the left or right subtree accordingly.

• In Deletion, we must handle three cases: deleting a leaf node, deleting a node with one child, and deleting a node with two children (replaced by inorder successor or predecessor).

• In Inorder Traversal, the tree is traversed in the order Left → Root → Right, which produces the keys in sorted order.

Thus, recursion simplifies the implementation of BST operations and demonstrates the divide-and-conquer approach in programming.

**Experimental Procedure:**

1. Define a node structure containing an integer data field and two pointers (left and right).

2. Implement a recursive insertion function to add new nodes while maintaining BST properties.

3. Implement a recursive deletion function to remove nodes by handling three cases:

• Leaf node

• Node with one child

• Node with two children (using inorder successor).

4. Implement a recursive inorder traversal function to display elements in ascending order.

5. Create a menu-driven program using printf() and scanf() with options for: Insert, Delete, Display (Inorder), and Exit.

6. Compile and run the program in a C compiler (e.g., GCC/Code::Blocks).

7.Test the program with sample data (insert, delete, and display) and verify that inorder traversal always produces a sorted sequence

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*left, \*right;

};

struct Node\* newNode(int value) {

struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node)); temp->data = value;

temp->left = temp->right = NULL;

return temp;

}

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

if (root == NULL) {

return newNode(value);

}

if (value < root->data)

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

else if (value > root->data)

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

}

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

root = root->left;

return root;

}

struct Node\* deleteNode(struct Node\* root, int key) { if (root == NULL) return root;

if (key < root->data)

root->left = deleteNode(root->left, key); else if (key > root->data)

root->right = deleteNode(root->right, key); else {

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

struct Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data); }

return root;

}

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

printf("%d ", root->data);

inorder(root->right);

}

}

int main() {

struct Node\* root = NULL;

int choice, value;

do {

printf("\n--- Binary Search Tree Operations (Recursive) ---\n");

printf("1. Insert a node\n");

printf("2. Delete a node\n");

printf("3. Inorder Traversal\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

root = insert(root, value);

printf("Node inserted successfully.\n"); break;

case 2:

printf("Enter value to delete: ");

scanf("%d", &value);

root = deleteNode(root, value);

printf("Node deleted successfully (if present).\n"); break;

case 3:

printf("Inorder Traversal: ");

inorder(root);

printf("\n");

break;

case 4:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice! Try again.\n");

}

} while (choice != 4);

return 0;

}

**Errors:**

1.If <stdio.h> or <stdlib.h> is not included, functions like printf, scanf, malloc, and free will cause compilation errors.

2. To declare return type properly (e.g., writing insert(root, value) instead of struct Node\* insert(struct Node\* root, int value)), compilation will fail.

3.Writing struct node instead of struct Node (case mismatch) will cause errors.

**Output:**

--- Binary Search Tree Operations (Recursive) ---

1. Insert a node

2. Delete a node

3. Inorder Traversal

4. Exit

Enter your choice: 1

Enter value to insert: 50

Node inserted successfully.

--- Binary Search Tree Operations (Recursive) --- 1. Insert a node

2. Delete a node

3. Inorder Traversal

4. Exit

Enter your choice: 1

Enter value to insert: 30

Node inserted successfully.

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 1

Enter value to insert: 70

Node inserted successfully.

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 1

Enter value to insert: 20

Node inserted successfully.

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 1

Enter value to insert: 40

Node inserted successfully.

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 1

Enter value to insert: 60

Node inserted successfully.

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 1

Enter value to insert: 80

Node inserted successfully.

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 3

Inorder Traversal: 20 30 40 50 60 70 80

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 2

Enter value to delete: 20

Node deleted successfully (if present).

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 3

Inorder Traversal: 30 40 50 60 70 80

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 2

Enter value to delete: 30

Node deleted successfully (if present).

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 3

Inorder Traversal: 40 50 60 70 80

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 2

Enter value to delete: 50

Node deleted successfully (if present).

--- Binary Search Tree Operations (Recursive) --- Enter your choice: 3

Inorder Traversal: 40 60 70 80

--- Binary Search Tree Operations (Recursive) ---

Enter your choice: 4

Exiting program.

**Related Short Questions:**

**Q1. What is a Binary Search Tree (BST)?**

A BST is a binary tree in which the left child of a node contains values less than the node, and the right child contains values greater than the node.

**Q2. What property distinguishes a BST from a normal binary tree?** In a BST, nodes are arranged in **sorted order**: left subtree < root < right subtree. A normal binary tree has no such ordering rule.

**Q3. What are the three cases of node deletion in a BST?** 1. Node is a **leaf** → simply delete it.

2. Node has **one child** → replace node with its child.

3. Node has **two children** → replace with inorder successor (or predecessor) and delete it.

**Q4. Why is recursion commonly used in tree operations?** Because a tree is a **recursive structure** (each subtree is itself a tree), recursion simplifies traversal, insertion, and deletion.

**Q5. What is the time complexity of insertion, deletion, and search in a BST?**

• Average case: **O(log n)**

• Worst case (skewed tree): **O(n)**

**Q6. What is the output of an inorder traversal of a BST?** Inorder traversal always produces the elements in **ascending sorted order**.

**Q7. How do you find the inorder successor of a node in a BST?** The inorder successor is the **minimum node in the right subtree** of the given node.

**Q8. What is the difference between preorder, inorder, and postorder traversal?**

• Preorder: Root → Left → Right

• Inorder: Left → Root → Right

• Postorder: Left → Right → Root

**Q9. What happens if you try to insert a duplicate element into a BST?**

In most implementations, duplicate elements are **ignored** or stored in a special way (e.g., counted or stored in a linked list).

**Q10. What is the worst-case time complexity of BST operations, and when does it occur?**

Worst-case is **O(n)**, which occurs when the tree becomes **skewed** (like a linked list).