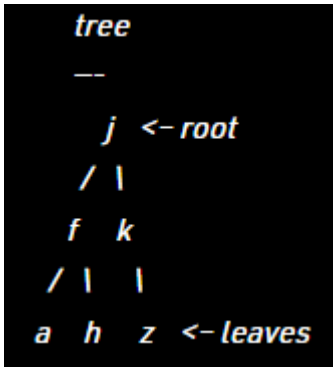
Name	Virinchi Sadashiv Shettigar
UID no.	2021300118
Experiment No.	5

AIM:	Implement a given problem statement using Binary Search Trees.
PROBLEM STATEMENT:	Insertion and Deletion operation where deletion is of leaf node.
THEORY:	**Tree** A tree is a popular data structure that is non-linear in nature. Unlike other data structures like an array, stack, queue, and linked list which are linear in nature, a tree represents a hierarchical structure. The ordering information of a tree is not important. A tree contains nodes and 2 pointers. These two pointers are the left child and the right child of the parent node. Let us understand the terms of tree in detail. **Root:** The root of a tree is the topmost node of the tree that has no parent node. There is only one root node in every tree. **Edge:** Edge acts as a link between the parent node and the child node. **Leaf:** A node that has no child is known as the leaf node. It is the last node of the tree. There can be multiple leaf nodes in a tree. Subtree: The subtree of a node is the tree considering that particular node as the root node. Depth: The depth of the node is the distance from the

root node to that particular node.
Height: The height of the node is the distance from that node to the deepest node of that subtree.
Height of tree: The Height of the tree is the maximum height of any node. This is same as the height of root node.



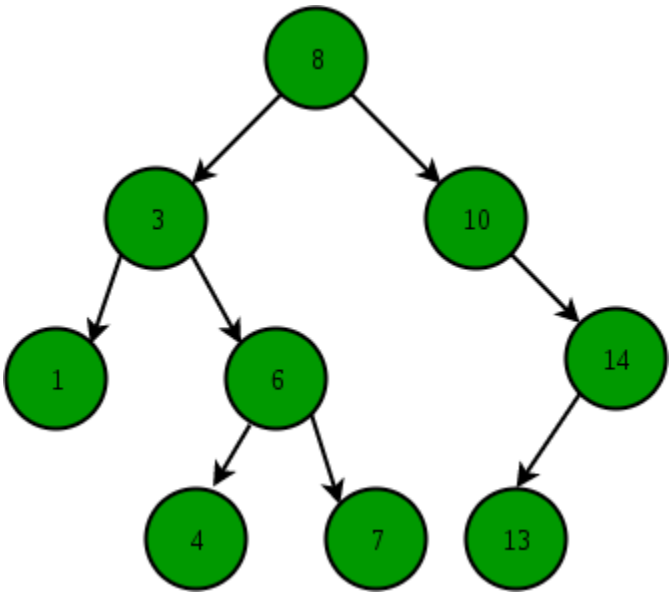
The types of Trees in the Data Structure.

1. Binary tree.
2. Binary Search Tree.
3. AVL Tree.
4. B-Tree.

BINARY SEARCH TREE

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- 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.



- There must be no duplicate nodes(BST may have duplicate values with different handling approaches)

HANDLING APPROACH FOR DUPLICATE VALUES

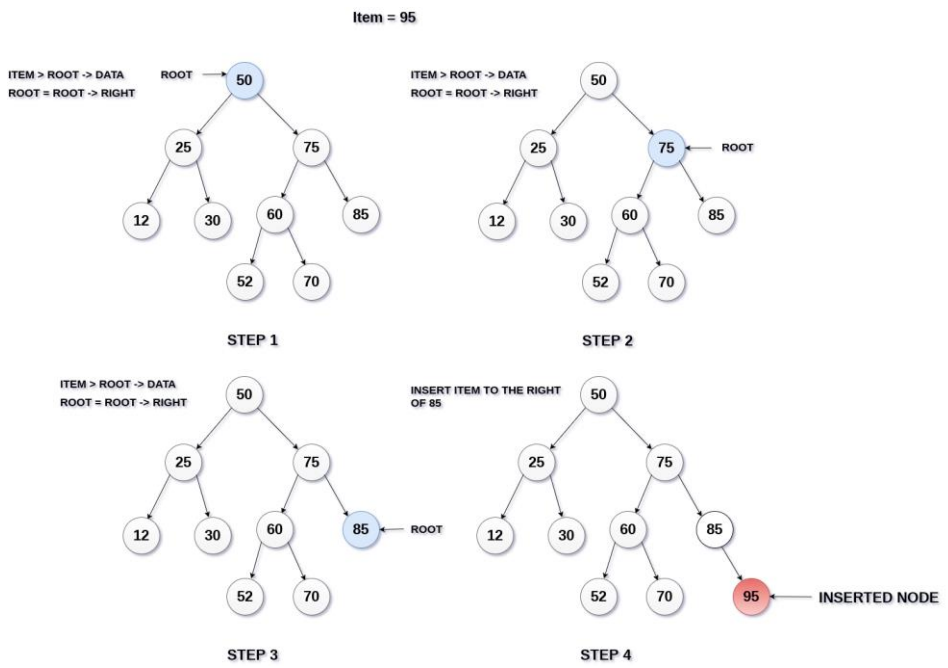
- You can not allow the duplicated values at all.
- We must follow a consistent process throughout i.e either store duplicate value at the left or store the duplicate value at the right of the root, but be consistent with your approach.
- We can keep the counter with the node and if we found the duplicate value, then we can increment the counter

OPERATIONS ON BINARY SEARCH TREES

- Insertion
- Deletion
- Traversal

INSERTION

Insertion of element in a Binary Search Tree is done using a top-to-bottom approach. The element that is to be inserted is first checked with root node. If it is greater than or equal to the root element, it is passed on to the right subtree to check again for the same condition. If the given element reaches at a point where it has no child, the given element becomes the leaf node/ child.
Example:



DELETION

When we delete a node, three possibilities arise.

1) Node to be deleted is the leaf: Simply remove from the tree.

```
          50                            50
        /    \        delete(20)      /   \
      30     70      --------->     30     70
     /  \    /  \                     \    / \
    20 40 60  80                     40  60   80
```

2) Node to be deleted has only one child: Copy the child to the node and delete the child

```
          50                            50
        /    \        delete(30)      /   \
      30     70      --------->     40    70
        \    / \                           /  \
        40 60 80                          60 80
```

3) Node to be deleted has two children: Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also

be used.

```
          50                            60
        /   \          delete(50)     /   \
      40    70        --------->     40    70
           /  \                            \
         60   80                           80
```

The important thing to note is, inorder successor is needed only when the right child is not empty. In this particular case, inorder successor can be obtained by finding the minimum value in the right child of the node.

TRAVERSAL

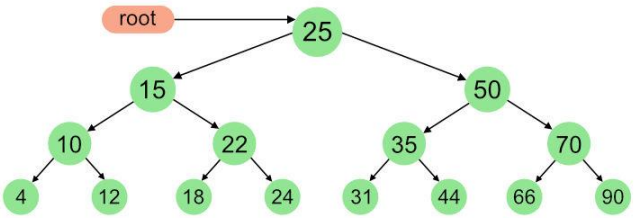
Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used methods for traversing trees:

- Inorder
- Preorder
- Postorder

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



INORDER:

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)

Uses:

In the case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-

increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.

PREORDER:

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

Uses:

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expressions on an expression tree.

POSTORDER:

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

Uses:

Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree

ADVANTAGES OF BINARY SEARCH TREES

1. BST is fast in insertion and deletion when balanced.
2. BST is efficient.
3. We can also do range queries – find keys between N and M (N <= M).
4. BST code is simple as compared to other data structures.

DISADVANTAGES OF BINARY SEARCH TREES

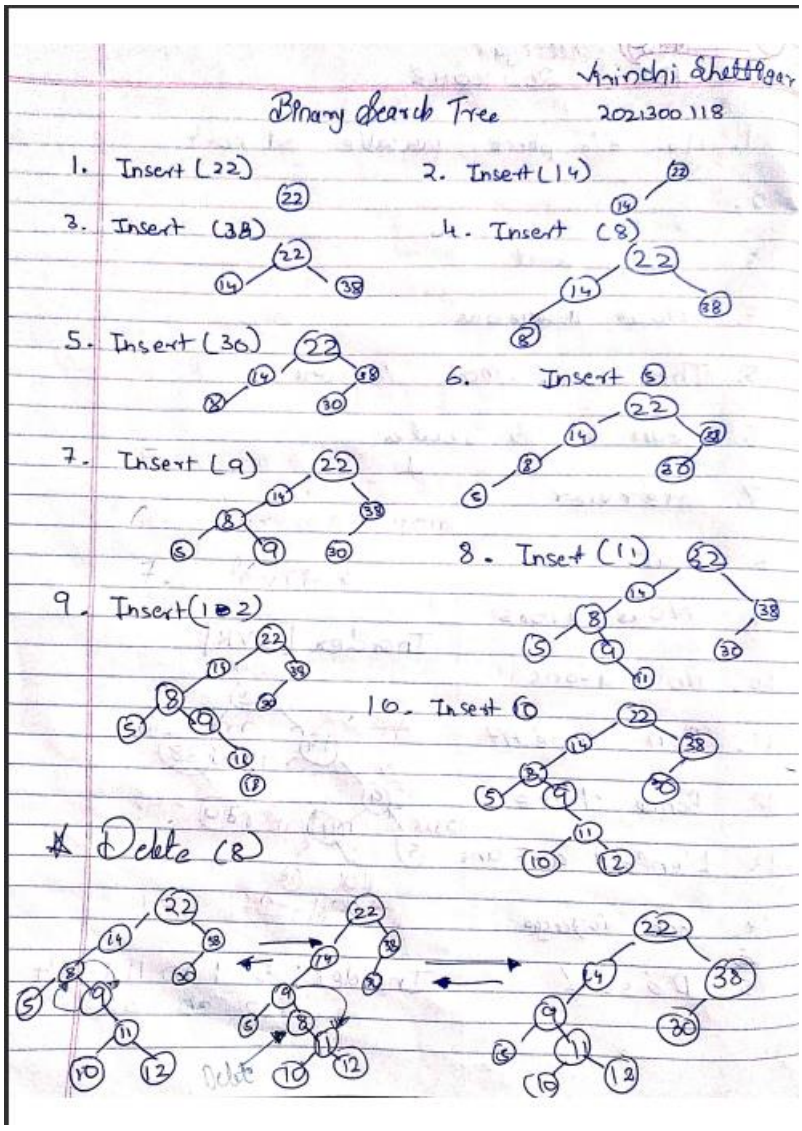
5. The main disadvantage is that we should always implement a balanced binary search tree. Otherwise the cost of operations may not be logarithmic and degenerate into a linear search on an array.
6. Accessing the element in BST is slightly slower than array.
7. A BST can be imbalanced or degenerated which can increase the complexity.

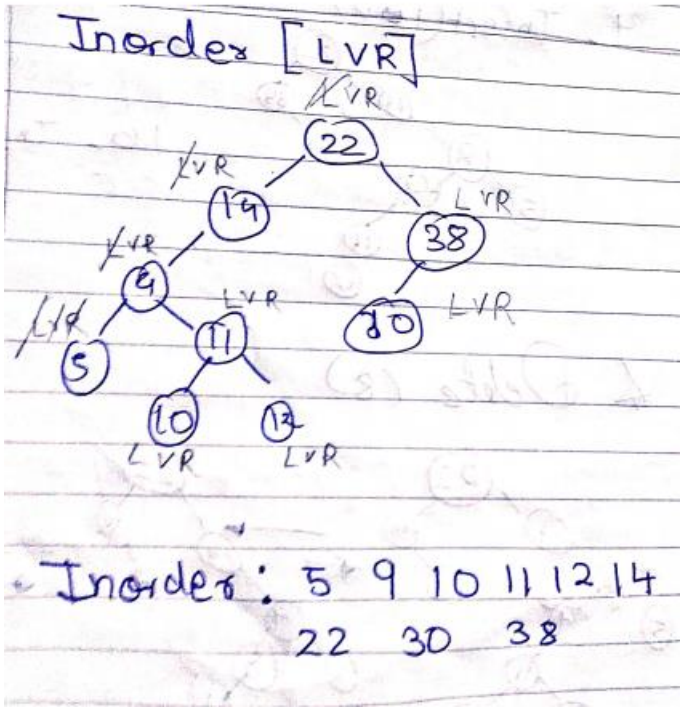
APPLICATIONS OF BINARY SEARCH TREES

1. BSTs are used for indexing in databases.
2. It is used to implement search algorithms.
3. BSTs are used to implement the Huffman coding algorithm.

	4. It is also used to implement dictionaries.
ALGORITHM:	Exp5 Class Main function 1. Create a BST object obj 2. Display menu for menu driven program 3. Input choice from user 4. If choice is 1, input data from user and call insert(data) 5. If choice is 2, input data from user and call delete(data) 6. If choice is 3, call inorder() 7. If choice is 4, call preoder() 8. If choice is 5, call postorder() 9. Repeat steps from 2 to 8 until user chose to exit the program BST class Node class Data Members: int data,Node left, Node right Constructor Node(int data) set data to data and left and right to null Data Members Node root Void insert(int data) 1. Set root by calling insert(root, data) Node insert(Node root,int data) 1. If root equals to null, then initialize root using new Node(data) 2. Else if data > data of a root then insert then set by recursively calling insert(root.right, data). 3. Else, left of root is set by recursively calling insert(root.left, data) 4. Return root. void delete(int data) 1. Set root by calling delete(root,data) Node delete(Node root,int data) 1. If root equals to null, return root

	2. Else if data < data of a root, left of toot is et by recursively calling delete(root.left,data) 3. Else if data > data of a root, right of toot is et by recursively calling delete(root.right,data) 4. Else check if left and right of root Node is null if yes then set root equals to null 5. Else if left of root Node is null if yes then assign root node to its right of it. 6. Else if right of root Node is null if yes then assign root node to its left of it. 7. Else create a Node temp by calling the function findMin(root.right) 8. Set root.data to temp.data 9. Set root.right by calling the delete(root.right,temp.data) Node findMin(Node root) 1. Until left of root is not equal to null,set root to left of root 2. Return root Void Inorder() 1. Call inoder(root) Void Inoder(Node root) 1. Call Inorder(root.left) 2. Print data of root 3. Call Inoder(root.right) Void Preorder() 1. Call Preorder(root) Void Preorder(Node root) 1. Print data of root 2. Call Preorder(root.left) 3. Call Preorder(root.right) Void Postorder() 1. Call Postorder(root) Void Postorder(Node root) 1. Call Postorder(root.left) 2. Call Postorder(root.right) 3. Print data of root

PROBLEM-SOLVING:	



PROGRAM:

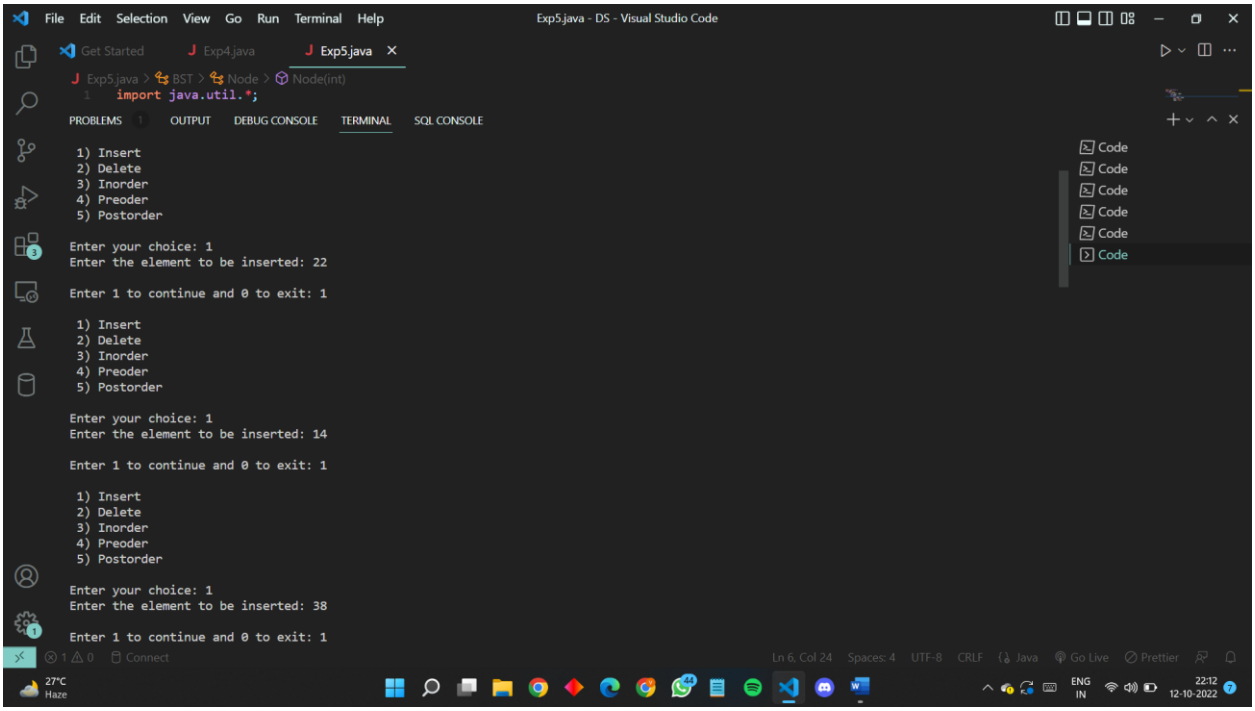
```
import java.util.*;
class BST{
    class Node{
        Node left, right;
        int data;
        Node(int data){
            this.data = data;
            left = null;
            right = null;
        }
    }
    Node root;
    void insert(int data){
        root = insert(root, data);
    }
    Node insert(Node root, int data){
        if(root == null){
            root = new Node(data);
        }
        else if(data >= root.data){
            root.right = insert(root.right, data);
        }
        else{
            root.left = insert(root.left, data);
```

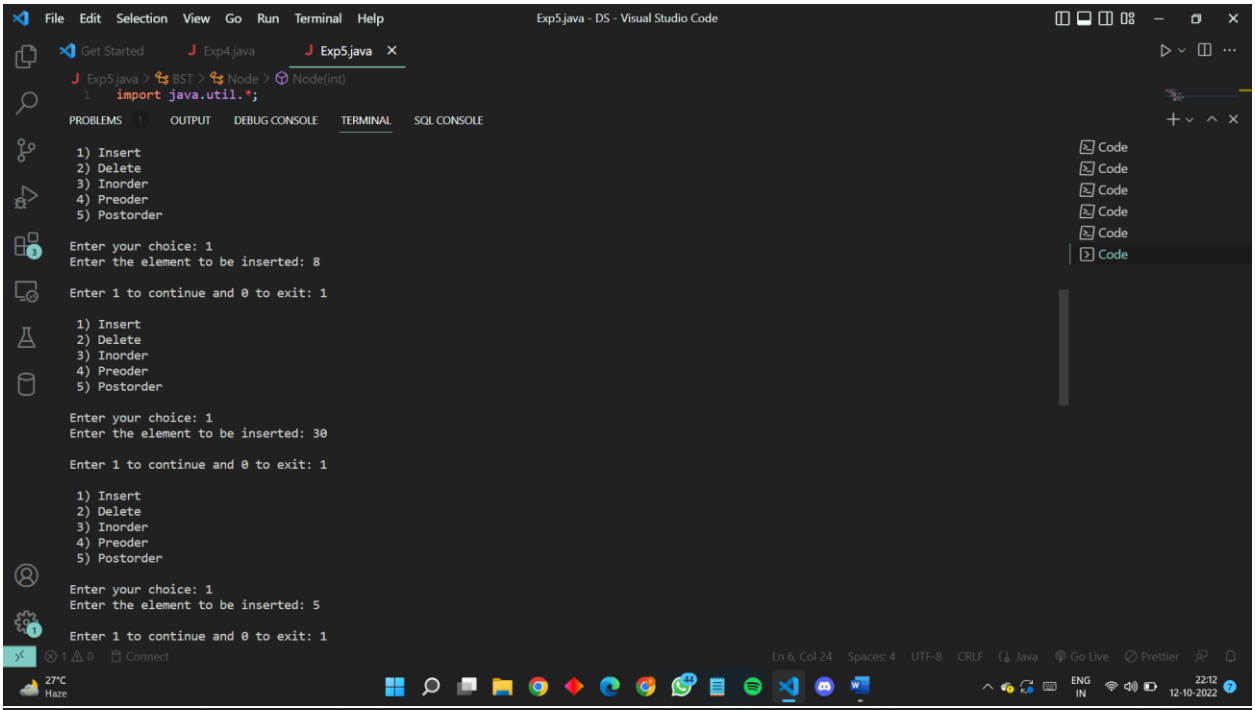
```
    }
    return root;
}
void delete(int data){
    root = delete(root,data);
}
Node delete(Node root, int data){
    if(root == null){
        return root;
    }
    else if(data < root.data){
        root.left = delete(root.left, data);
    }
    else if(data > root.data){
        root.right = delete(root.right, data);
    }
    else{
        if(root.left == null && root.right == null){
            root = null;
        }
        else if(root.left == null){
            root = root.right;
        }
        else if(root.right == null){
            root = root.left;
        }
        else{
            Node temp = findMin(root.right);
            root.data = temp.data;
            root.right = delete(root.right, temp.data);
        }
    }
    return root;
}
Node findMin(Node root){
    while(root.left != null){
        root = root.left;
    }
    return root;
}
void inorder(){
    inorder(root);
}
void inorder(Node root){
    if(root != null){
        inorder(root.left);
```

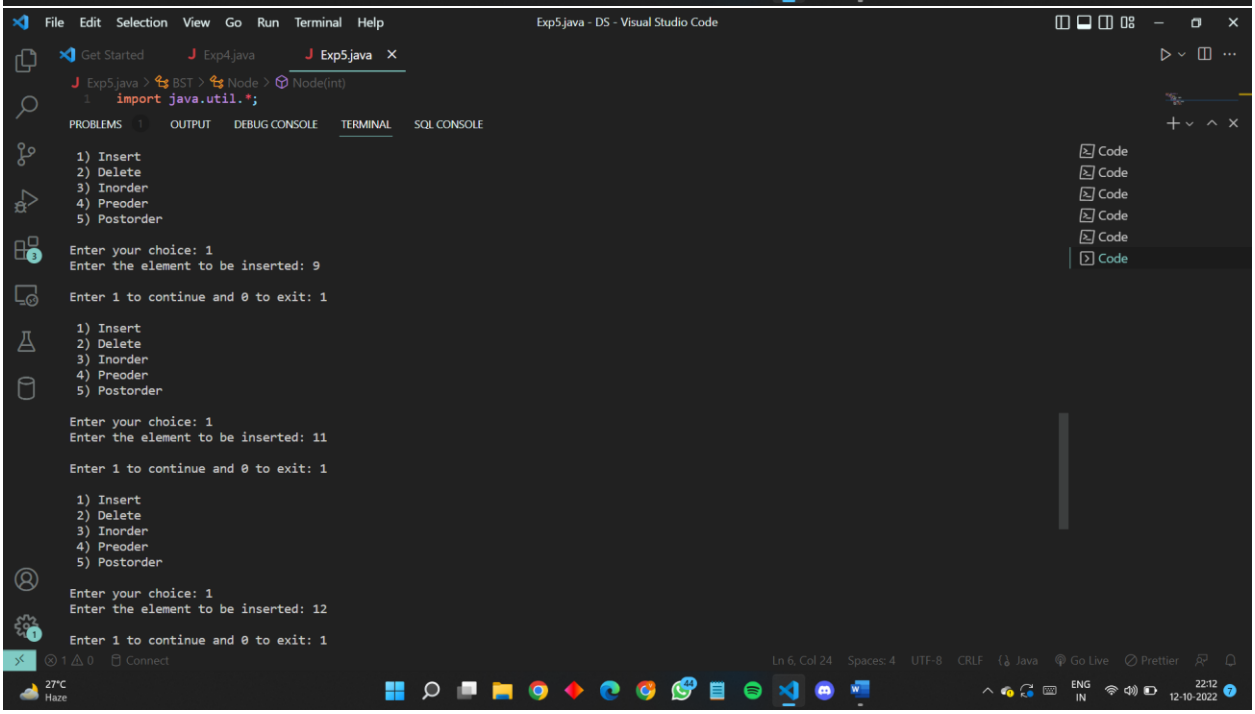
```
            System.out.print(root.data + " ");
            inorder(root.right);
        }
    }
    void preorder(){
        preorder(root);
    }
    void preorder(Node root){
        if(root != null){
            System.out.print(root.data + " ");
            preorder(root.left);
            preorder(root.right);
        }
    }
    void postorder(){
        postorder(root);
    }
    void postorder(Node root){
        if(root != null){
            postorder(root.left);
            postorder(root.right);
            System.out.print(root.data + " ");
        }
    }
}
class Exp5{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST obj = new BST();
        int choice, flag=1;
        while (flag == 1) {
            System.out.println("\n 1) Insert \n 2) Delete \n 3) Inorder \n 4)
Preoder \n 5) Postorder\n ");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter the element to be inserted: ");
                    obj.insert(sc.nextInt());
                    break;
                case 2:
                    System.out.print("Enter the element to be deleted: ");
                    obj.delete(sc.nextInt());
                    break;
                case 3:
                    System.out.print("Inorder: ");
```

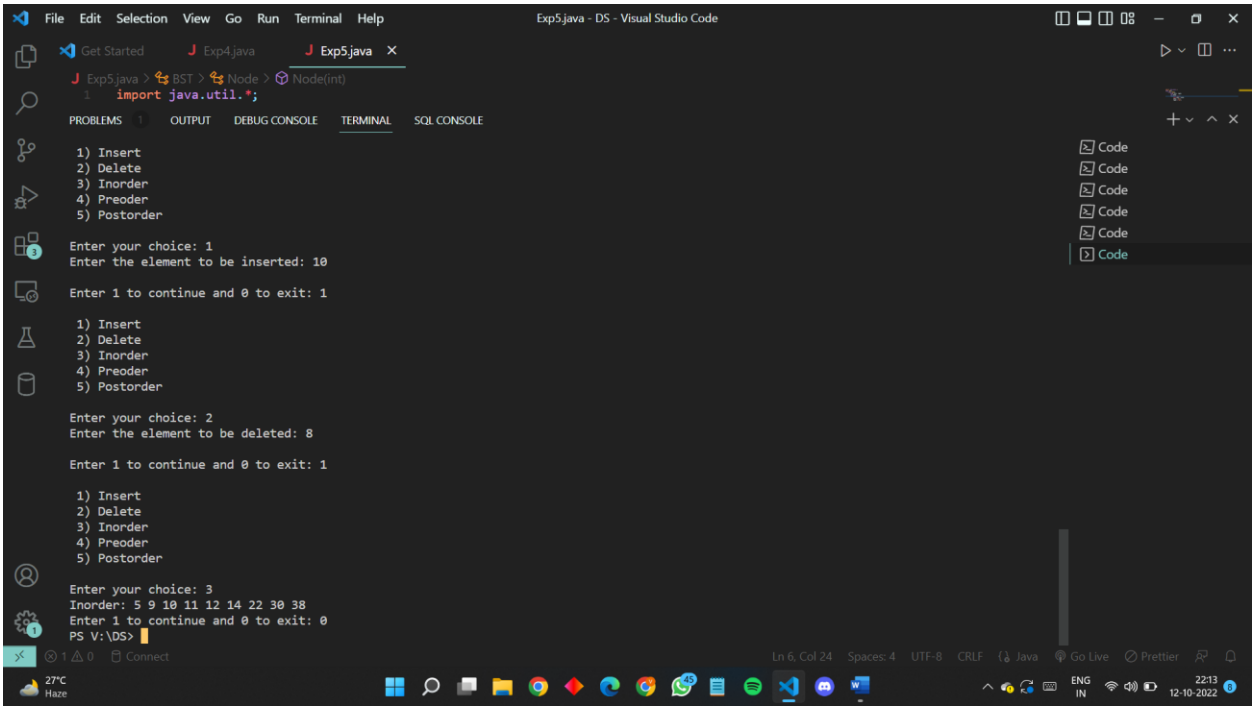
```
                obj.inorder();
                break;
            case 4:
                System.out.print("Preorder: ");
                obj.preorder();
                break;
            case 5:
                System.out.print("Postorder: ");
                obj.postorder();
                break;
            default:
                System.out.println("Invalid choice");
            }
            System.out.print("\nEnter 1 to continue and 0 to exit: ");
            flag=sc.nextInt();
            if (flag==0) {
                break;
            }

        }
    }
}
```

OUTPUT:




File Edit Selection View Go Run Terminal Help
Exp5.java - DS - Visual Studio Code
Get Started
Exp4.java
Exp5.java
Exp5.java > BST > Node > Node(int)
import java.util.*;
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
SQL CONSOLE
1) Insert
2) Delete
3) Inorder
4) Preoder
5) Postorder
Enter your choice: 1
Enter the element to be inserted: 8
Enter 1 to continue and 0 to exit: 1
1) Insert
2) Delete
3) Inorder
4) Preoder
5) Postorder
Enter your choice: 1
Enter the element to be inserted: 30
Enter 1 to continue and 0 to exit: 1
1) Insert
2) Delete
3) Inorder
4) Preoder
5) Postorder
Enter your choice: 1
Enter the element to be inserted: 5
Enter 1 to continue and 0 to exit: 1
Code
Code
Code
Code
Code
Code
Ln 6, Col 24
Spaces: 4
UTF-8
CRLF
Java
Go Live
Prettier
27°C
Haze
ENG
IN
22:12
12-10-2022


File Edit Selection View Go Run Terminal Help
Exp5.java - DS - Visual Studio Code
Get Started
Exp4.java
Exp5.java
Exp5.java > BST > Node > Node(int)
import java.util.*;
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
SQL CONSOLE
1) Insert
2) Delete
3) Inorder
4) Preoder
5) Postorder
Enter your choice: 1
Enter the element to be inserted: 9
Enter 1 to continue and 0 to exit: 1
1) Insert
2) Delete
3) Inorder
4) Preoder
5) Postorder
Enter your choice: 1
Enter the element to be inserted: 11
Enter 1 to continue and 0 to exit: 1
1) Insert
2) Delete
3) Inorder
4) Preoder
5) Postorder
Enter your choice: 1
Enter the element to be inserted: 12
Enter 1 to continue and 0 to exit: 1
Code
Code
Code
Code
Code
Code
Ln 6, Col 24
Spaces: 4
UTF-8
CRLF
Java
Go Live
Prettier
27°C
Haze
ENG
IN
22:12
12-10-2022



CONCLUSION:	In this experiment, I learned about binary search trees and learned about their advantages and disadvantages. Then I implemented a menu driven binary search tree program for insertion, deletion, inOrder, preOrder and postOrder as per needs of user.