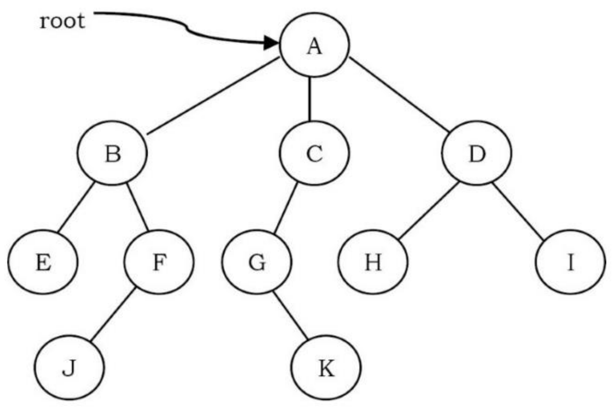
Name	**Hatim Yusuf Sawai**
UID no.	**2021300108**
Experiment No.	**5**

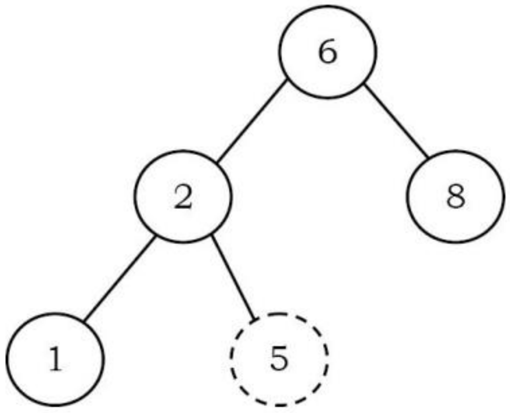
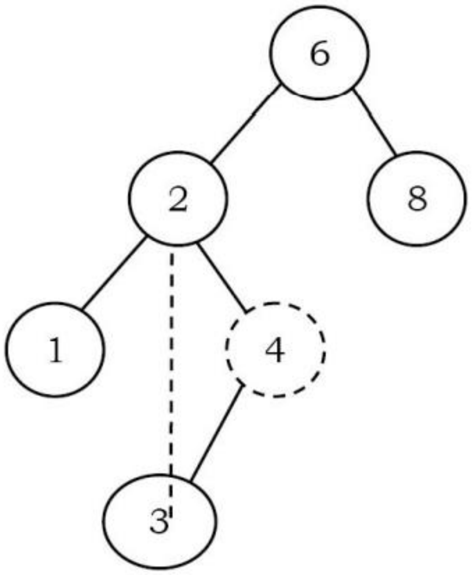
AIM:	**Program on BST (Binary Search Tree)**
Program 1	
PROBLEM STATEMENT:	Write A Menu-Driven program to show insertion, traversal & deletion operations on a Binary Search Tree.
THEORY:	**What is a tree?** A tree is a data structure similar to a linked list but instead of each node pointing simply to the next node in a linear fashion, each node points to a number of nodes. A tree is an example of a nonlinear data structure. A tree structure is a way of representing the hierarchical nature of a structure in a graphical form. **Structure:**  **Terminology:** • The **root** of a tree is the node with no parents. There is only 1 root node in a tree (node A in the above example). • An **edge** refers to the link from parent to child (all links in the figure).

• A node with no children is called a **leaf** node (E, J, K, H and I).
• Children of the same parent are called **siblings** (B, C & D are siblings of A, and E & F are the siblings of B).
• A node p is an **ancestor** of node q if there exists a path from the root to q and p appears on the path. The node q is called a **descendant** of p. For example, A, C and G are the ancestors of K.
• The set of all nodes at a given depth is called the **level** of the tree (B, C and D are the same level). The root node is at **level zero.**

Binary Tree:
A tree is called a binary tree if each node has zero children, one child or two children. An empty tree is also a valid binary tree. We can visualize a binary tree as consisting of a root and two disjoint binary trees called the left and right subtrees of the root.

Binary Search Tree (BST):
In binary search trees, all the left subtree elements should be less than the root data and all the right subtree elements should be greater than the root data. This is called the binary search tree property. Note that, this property should be satisfied at every node in the tree.
• The left subtree of a node contains only nodes with keys less than the node key.
• The right subtree of a node contains only nodes with keys greater than the node's key.
• Both the left and right subtrees must also be binary search trees.

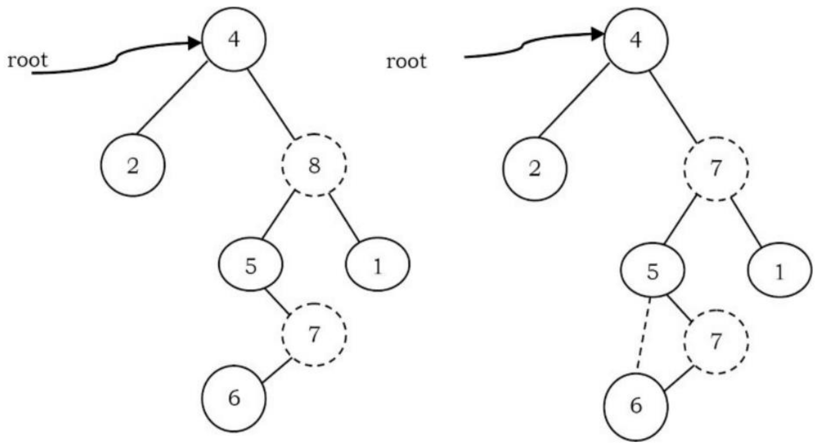
Operations of BST:
The 3 types of BST operations:
• **Insertion of an element**
To insert an element into a bst we need to find the location first by a recursive mechanism: if the element is greater than the node go right, if the element is less than the node go left and if its equal then come out of the operation.

	• **Deletion of elements in 3 diff cases** **Case 1: Leaf Node** If the element to be deleted is a leaf node: return NULL to its parent. That means to make the corresponding child pointer NULL. Example: to delete **5**  **Case 2: One Child:** If the element to be deleted has one child: In this case we just need to send the current node's child to its parent. Example: to delete **4** 

Case 3: Two Children

If the element to be deleted has both children: The general strategy is to replace the key of this node with the largest element of the left subtree and recursively delete that node (which is now empty).

Example: to delete **8**



• **Traversing using inorder, preorder & postorder**

PreOrder (VLR) Traversal:

Preorder traversal is defined as follows:

• Visit the root.

• Traverse the left subtree in Preorder.

• Traverse the right subtree in Preorder.

InOrder (LVR Traversal:

In Inorder Traversal the root is visited between the subtrees. Inorder traversal is defined as follows:

• Traverse the left subtree in Inorder.

• Visit the root.

• Traverse the right subtree in Inorder.

PostOrder (LRV) Traversal:

In postorder traversal, the root is visited after both subtrees. Postorder traversal is defined as follows:

• Traverse the left subtree in Postorder.

• Traverse the right subtree in Postorder.

• Visit the root.

ALGORITHM:

```
1. Create BinTree Class with inner Node class
2. Create 2 ref vars: right, left & int var data
3. Initialize root node to null
4. Create methods for diff operations of  BinTree
```

insert Method:

1. This method helps to make the recursive call to **inserter** Method

inserter Method:

```
1. if root is null:
2. root = new node(data)
3. return root
4. if data is less than root.data
5. root.left = inserter(root.left, data)
6. else if data is more than root.data
7. root.right = inserter(root.right, data)
8. Return the root
```

delete Method:

1. This method deletes the given node by making recursive call to deleter method

deleter Method:

```
1. if root is null then return the root
2. if data is less than root.data
3. root.left = deleter(root.left, data)
4. else if data is more than root.data
5. root.right = deleter(root.right, data)
6. else
a. if root.left is null then return root.right
b. else if root.right is null then return root.left
c. root.data = minValue(root.right);
d. root.right = deleter(root.right, root.data);
7. return the root
```

minValue Method:

```
1. initialize minv equal to root.data
2. loop till root.left is not null:
a. minv = root.left.data
b. root = root.left
3. return minv
```

PreOrder Method:

```
1. if root is not null
2. print root.data
3. recurr: PreOrder(root.left)
4. recurr: PreOrder(root.right)
```

InOrder Method:

```
1. if root is not null
2. recurr: PreOrder(root.left)
3. print root.data
4. recurr: PreOrder(root.right)
```

PostOrder Method:

```
1. if root is not null
2. recurr: PreOrder(root.left)
3. recurr: PreOrder(root.right)
4. print root.data
```

PROBLEM SOLVING:

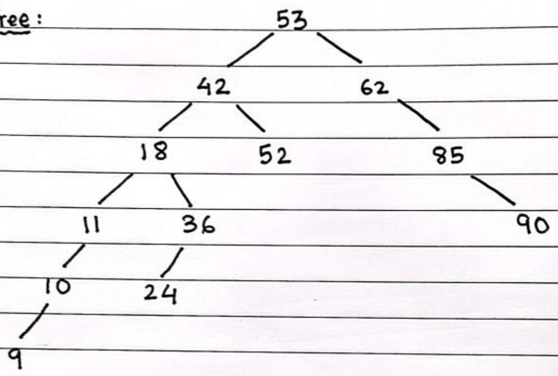
HATIM SAWAI

BST Operations:

1. Insertion 2. Deletion 3. Traversal

Tree input: 53 42 18 11 36 24 10 62 85 9 90 52

Tree:



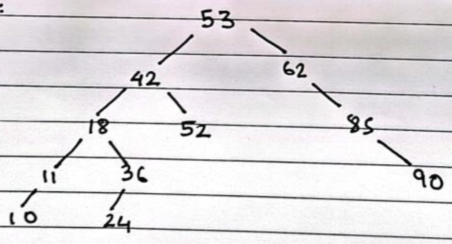
→ InOrder: 9,10,11,18,24,36,42,52,53,62,85,90

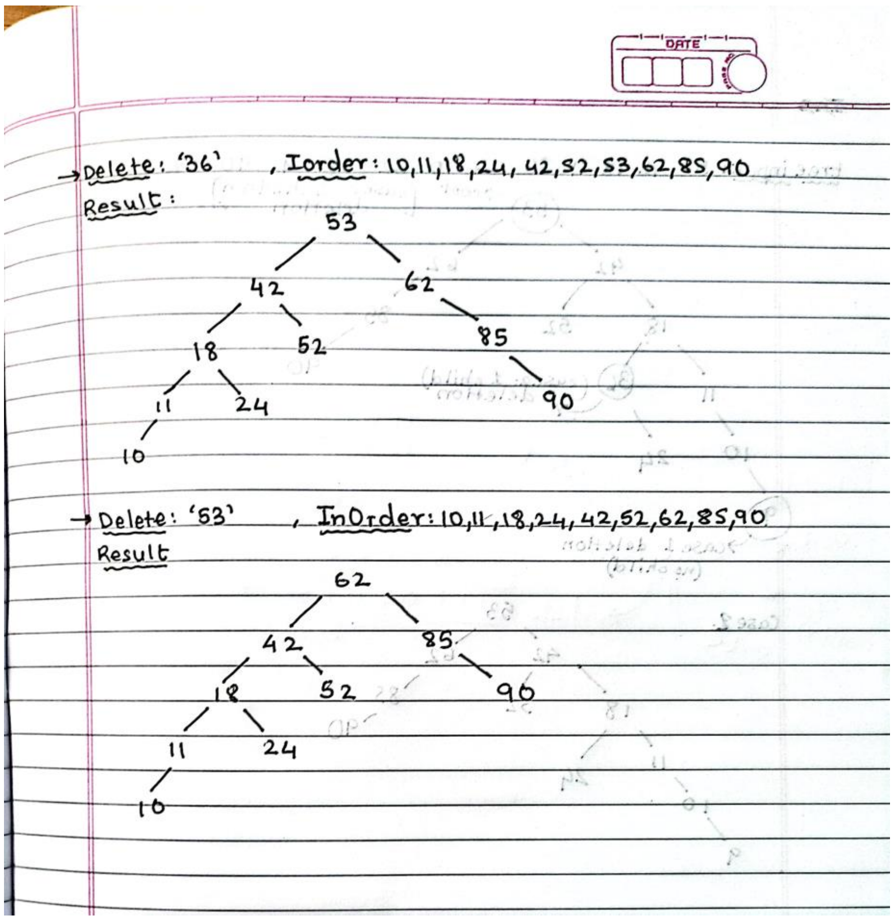
→ PostOrder: 9,10,11,24,36,18,52,42,90,85,62,53

→ PreOrder: 53,42,18,11,10,9,36,24,52,62,85,90

→ Delete: '9' , InOrder: 10,11,18,24,36,42,52,53,62,85,90

Result:




→ Delete: '36' , Iorder: 10,11,18,24,42,52,53,62,85,90

Result:

→ Delete: '53' , InOrder: 10,11,18,24,42,52,62,85,90

Result

PROGRAM:

BTCheck.java:

```
import java.util.Scanner;
import bintreeds.BinTree;
public class BTCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BinTree bt = new BinTree();
        int choice,flag;
        int n,d;
        while(true) {
            System.out.println("Select 1 operation:\n1. Insert\t2. Delete\n3. PreOrder\t4. InOrder\t5. PostOrder");
            choice = sc.nextInt();
            switch(choice) {
                case 1:
```

```
                System.out.print("Enter no. of elements to insert: ");
                n = sc.nextInt();
                System.out.print("Enter the elements: ");
                for (int i = 0; i < n; i++) {
                    bt.insert(sc.nextInt());
                }
                System.out.print("InOrder: ");
                bt.InOrder(bt.root);
                System.out.println();
                break;
            case 2:
                System.out.print("Enter the element to delete: ");
                d = sc.nextInt();
                bt.delete(bt.root,d);
                System.out.print("InOrder: ");
                bt.InOrder(bt.root);
                System.out.println();
                break;
            case 3:
                System.out.print("PreOrder: ");
                bt.PreOrder(bt.root);
                System.out.println();
                break;
            case 4:
                System.out.print("InOrder: ");
                bt.InOrder(bt.root);
                System.out.println();
                break;
            case 5:
                System.out.print("PostOrder: ");
                bt.PostOrder(bt.root);
                System.out.println();
                break;
            case 6:
                System.out.println("Size of the tree is " +
bt.size(bt.root));
```

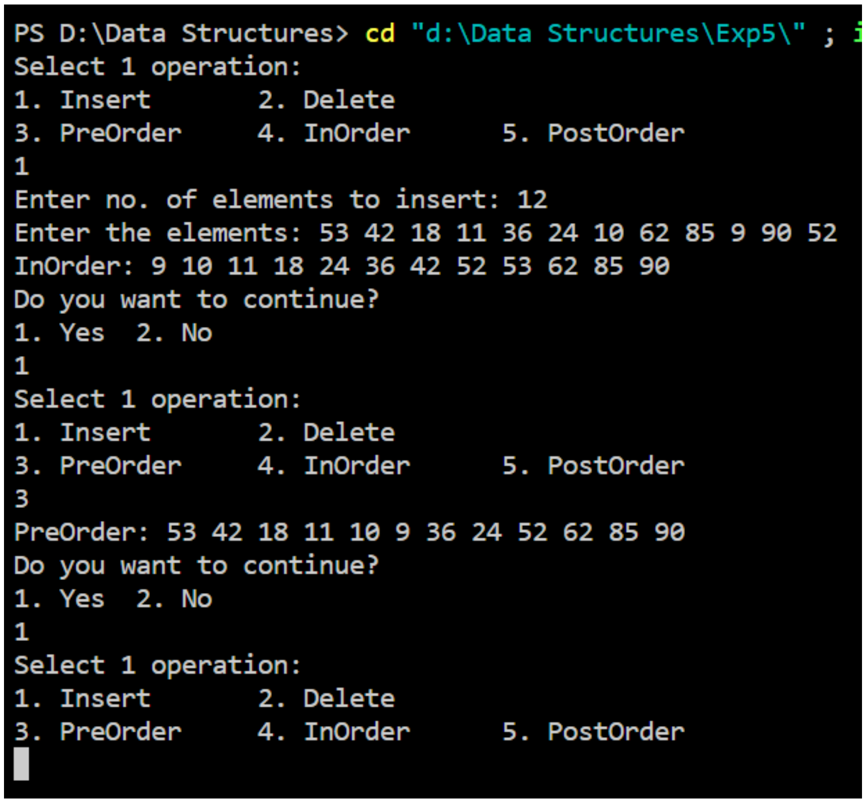
```
                    break;
                default:
                    System.out.println("Invalid choice!");
            }
            System.out.println("Do you want to continue?\n1. Yes\t2. No");
            flag = sc.nextInt();
            if (flag == 2) {
                break;
            }
        }
        sc.close();
    }
}
```

BinTree.java:

```
package bintreeds;
public class BinTree {
    class Node {
        int data;
        Node left, right;
        Node(int data) {
            this.data = data;
            left = right = null;
        }
    }
    public Node root=null;
    public void insert(int data) {
        root = inserter(root, data);
    }
    private Node inserter(Node root, int data) {
        if (root == null) {
            root = new Node(data);
            return root;
        }
        if (data < root.data) {
            root.left = inserter(root.left, data);
        } else if (data > root.data) {
            root.right = inserter(root.right, data);
        }
        return root;
    }
    public void delete(Node root,int data) {
        root = deleter(root, data);
    }
    private Node deleter(Node root, int data) {
        if (root == null) {
            return root;
        }
        if (data < root.data) {
            root.left = deleter(root.left, data);
        } else if (data > root.data) {
            root.right = deleter(root.right, data);
        } else {
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
                return root.left;
            }
            root.data = minValue(root.right);
            root.right = deleter(root.right, root.data);
        }
        return root;
    }
    private int minValue(Node root) {
        int minv = root.data;
        while (root.left != null) {
            minv = root.left.data;
            root = root.left;
        }
        return minv;
    }
    public int size(Node root) {
        if (root == null)
            return 0;
        else
            return (size(root.left) + 1 + size(root.right));
    }
    public void PreOrder(Node root) {
        if (root!=null) {
            System.out.print(root.data + " ");
            PreOrder(root.left);
            PreOrder(root.right);
        }
    }
    public void InOrder(Node root) {
        if (root!=null) {
            InOrder(root.left);
            System.out.print(root.data + " ");
            InOrder(root.right);
        }
    }
    public void PostOrder(Node root) {
        if (root!=null) {
            PostOrder(root.left);
            PostOrder(root.right);
            System.out.print(root.data + " ");
        }
    }
}
```

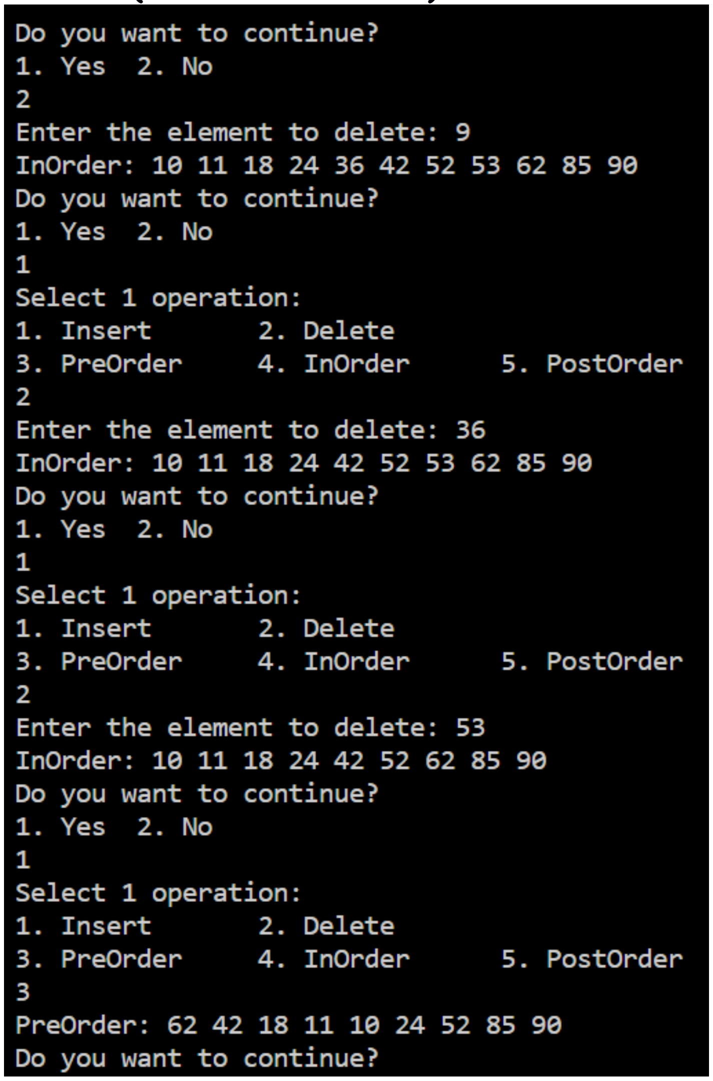
OUTPUT:

Insertion & Traversal:



```
PS D:\Data Structures> cd "d:\Data Structures\Exp5\" ; i
Select 1 operation:
1. Insert       2. Delete
3. PreOrder     4. InOrder      5. PostOrder
1
Enter no. of elements to insert: 12
Enter the elements: 53 42 18 11 36 24 10 62 85 9 90 52
InOrder: 9 10 11 18 24 36 42 52 53 62 85 90
Do you want to continue?
1. Yes  2. No
1
Select 1 operation:
1. Insert       2. Delete
3. PreOrder     4. InOrder      5. PostOrder
3
PreOrder: 53 42 18 11 10 9 36 24 52 62 85 90
Do you want to continue?
1. Yes  2. No
1
Select 1 operation:
1. Insert       2. Delete
3. PreOrder     4. InOrder      5. PostOrder
```

Deletion (All 3 cases in order):



```
Do you want to continue?
1. Yes  2. No
2
Enter the element to delete: 9
InOrder: 10 11 18 24 36 42 52 53 62 85 90
Do you want to continue?
1. Yes  2. No
1
Select 1 operation:
1. Insert       2. Delete
3. PreOrder     4. InOrder      5. PostOrder
2
Enter the element to delete: 36
InOrder: 10 11 18 24 42 52 53 62 85 90
Do you want to continue?
1. Yes  2. No
1
Select 1 operation:
1. Insert       2. Delete
3. PreOrder     4. InOrder      5. PostOrder
2
Enter the element to delete: 53
InOrder: 10 11 18 24 42 52 62 85 90
Do you want to continue?
1. Yes  2. No
1
Select 1 operation:
1. Insert       2. Delete
3. PreOrder     4. InOrder      5. PostOrder
3
PreOrder: 62 42 18 11 10 24 52 85 90
Do you want to continue?
```

CONCLUSION:	In this experiment, we learned how to perform insertion & deletion operations on a binary search tree. We also learned how to perform 3 types of traversals (preorder, inorder & postorder) on a BST.