LAB SESSION - 07 BINARY SEARCH TREE

PreLab:-

1.
```
package PreLab1;
public class Node {
        public int data;
        public Node left, right;
        public Node(int data) {
            this.data = data;
            right = null;
            left = null;
        }
}
```

```
package prelab1;
public class BinarySearchTree {
        Node root;
        public BinarySearchTree() {
            root = null;
        }
        public BinarySearchTree(int data) {
            root = new Node(data);
        }
```

```
public void insert(int data) {
        root = insertNode(data, root);
}
public Node insertNode(int data, Node root) {
        if (root == null) {
            root = new Node(data);
            return root;
        }
        else if (data < root.data)
            root.left = insertNode(data, root.left);
        else if (data > root.dat)
            root.right = insertNode(data, root.right);
        return root;
}
void inorder() {
        inorderNode(root);
}
void inorderNode(Node root) {
        if (root != null) {
            inorderNode(root.left);
            System.out.println(root.data);
            inorderNode(root.right);
        }
}
```

```
Package BreObj;
Public class Demo {
    Public static void main (String args[]){
    BinarySearchTree tree = new BinarySearch
                            Tree ();
        tree.insert (10);
        tree.insert (20); tree.insert (30);
        tree.insert (40);
        tree.insert (100);
        tree.inOrder();
    }
}
```

OUTPUT..

10
20
30
40
100
500

7, The program is same as previous program but we add these methods in Binary search tree.

```
void preOrder() {
        preOrderNode (root);
}
```

```
void PreorderNode(Node root) {
        if (root != null) {
            System.out.print(root.data + " -> ");
            preOrderNode(root.left);
            preOrderNode(root.right);
        }
}

void preorder() {
        preorderNode(root);
    }
void preorderNode(Node root) {
        if (root != null) {
            System.out.print(root.data + " -> ");
            preOrderNode(root.left);
            preOrderNode(root.right);
        }
}
void postorder() {
        postorderNode(root);
    }
void postorderNode(Node root) {
        if (root != null) {
            postorderNode(root.left);
            postorderNode(root.right);
            System.out.println(root.data + " -> ");
        }
}
```

OUTPUT:

INORDER

10 -> 20 -> 30 -> 40 -> 100 -> 500

POSTORDER:

10 -> 40 -> 30 -> 20 -> 500 -> 100

PREORDER:

100 -> 20 -> 10 -> 30 -> 40 -> 500.

PROGRAM:

```
package inLab1;
Public class Solution {

    Public TreeNode trim(TreeNode root,
int low, int high) {
        if(root == null)
            return null;
        if(root.data < low)
            return trim(root.right, low, high);
    root.left = trim(root.left, low, high);
     root.left = trim(root.right, low, high);
     return root;
    }
}
```

```
package InLab2;
public class Demo {
    public static void main(String args[]) {
        Solution s = new Solution();
        TreeNode tree = new TreeNode();
        tree.insert(3);
        tree.insert(0);
        tree.insert(4);
        tree.insert(2);
        tree.insert(1);
        s.trim(tree, 1, 3);
        tree.inorder();
    }
}
```

output:

3 2 1

```
2. package InLab2;
public class Node {
    public int data;
    public Node left, right;
    public Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}
package InLab2;
public class Solution {
    static int count = 0;
```

```
Public static Node insert(Node root, int data){
    if (root == null)
        return new Node(data);
    if ( data< root.data)
    root.left= insert(root.left, data);
    else if ( data > root.data)
        root.right = insert(root.right, data);
    return root;
}
Public static Node kthSmallest(Node root,
 int k) {
    if (root == null)
        return null;
    Node left = kthSmallest(root.left, k);
    if (left != null)
        return left;
    count++;
    if (count == k)
        return root;
    return kthSmallest(root.right, k);
}
Public static void printKth Smallest(Node
root, int k) {
    Node res = kth Smallest(root, k);
    if (res == null)
```

```
        System.out.println("There are less than k
            nodes in the BST");
    }
    public static void main(String args[]) {
        Node root=null;
        int keys[] = {20,8,22,4,12,10,14};
        for (int k: keys)
            root = insert(root, k);
        int k=3;
        Print kth smallest(root, k);
    }
}
```

OUTPUT:

k-th smallest element is 10

3.
```
public int count() {
    return countRightNode(root) + countLeftNode
                                  (root) - 1;
}
public int countLeftNode(Node root) {
    int count=0;
    while (root != null) {
        root = root.left;
    } count++;
    return count;
}
```

```
public int countRightNode(Node root) {
    int count=0;
    while (root!=null) {
        root= root.right;
        count++;
    }
    return count;
}
```

```
package inlab3;
public class Demo {
    public static void main(String args[]) {
        BinarySearchTree tree=new BinarySearch
                              Tree();
        tree.insert(3);
        tree.insert(9);
        tree.insert(20);
        tree.insert(15);
        tree.insert(7);
        System.out.println("Target Sum: "+tree.
                           count());
    }
}
```

OUTPUT:

```
Target Sum: 3
```

Post lab

```
package Post1Ob1;
public class Solution {
    static int S,P;
    public void SP(Node root, int val){
        if (root!=null) {
            if(root.data==val){
                if( root.left!=null){
                    Node t=root.left;
                    while(t.right!=null){
                        t=t.right;
                    }
                P=t.data;
                }
            if(root.right!=null){
                    Node t=root.right;
                    while(t.left!=null){
                        t=t.left;
                    }
                S=t.data;
            }
        else if(root.data>val){
                S=root.data;
            SP(root.left, val);
        }
```

```
        } else if (val > root.data) {
            p = root.data;
            sp = (root.right, val);
        }
    }
}

public static void main(String args[]) {
    Node root = new Node(20);
    root.left = new Node(10);
    root.right = new Node(30);
    root.left.left = new Node(5);
    root.right.right = new Node(35);
    Solution se = new Solution();
    se.sp(root, 6);
    System.out.println("Successor of
    6 is: " + s + " and predecessor is: " + p);
    se.sp(root, 30);
    System.out.println("Successor of
    30 is: " + s + " and predecessor is: " + p);
}
}
```

OUTPUT:

Inorder successor of 10 is : 13 and predessor is: 2

Inorder successor of 30 is: 35 and predessor is: 25

2. Package PostLab2;

Public class Solution {

Public TreeNode trim(TreeNode root, int low,
int high) {

if (root == null)
return null;

if (root.data < low)
return trim (root.right, low, high);

if (root.data > high)
return trim (root.left, low, high);

root.left = trim (root.left, low, high);

root.right = trim (root.right, low, high);

return root;

}

}

Package PostLab2;

Public class Demo {

```
public static void main(String args[]) {
    Solution s = new Solution();
    TreeNode tree = new TreeNode();
    tree.insert(4);
    tree.insert(1);
    tree.insert(6);
    tree.insert(0);
    tree.insert(2);
    tree.insert(5);
    tree.insert(7);
    tree.insert(3);
    tree.insert(8);
    TreeNode b = new TreeNode();
    s.trim(tree, 1, 3);
    tree.inorder();
}
}
```

OUTPUT.

3 0 1 3 6, 2 1, 3 6, 3 5, 2 6, 4 5, 3 3, 8