Institute Of Technology, Nirma university

BRANCH :- **Computer Science Engineering**

PRACTICAL SUBMISSION

|*|*STUDENT INFO*|*|

Name :- **Pratik Kansara**

Roll No. :- **20BCE510**

Division :- **E4**

|*|*SUBJECT INFO*|*|

Subject :- **Advanced Data Structures**

Practical No. :- **5**

Practical – 5

AIM :- Write a program to split a balance search tree at

i. Root
ii. ii. A given point of the split.

Code:

BstNode.java

```java
public class BstNode {

    public BstNode left;
    public BstNode right;
    public int key;
    public int val;

    public BstNode(int key, int val) {
        this.key = key;
        this.val = val;
    }
}
```

BST.java

```java
import java.util.Scanner;

public class BST {
    public BstNode root;
    private static int idxkey;
    private static int idxval;

    public void insert(int key, int val) {
        root = insertHelper(root, key, val);
    }

    private BstNode insertHelper(BstNode root, int key, int val) {
        if (root == null) {
            return new BstNode(key, val);
        }

        if (key < root.key) {
            root.left = insertHelper(root.left, key, val);
        } else {
            root.right = insertHelper(root.right, key, val);
        }

        return root;

    }
```

```
    public static BstNode createBST(int inOrderKey[], int inOrderVal[], int
start, int end) {

        if (start > end) {
            return null;
        }

        int mid = (start + end) / 2;
        BstNode t = new BstNode(inOrderKey[mid], inOrderVal[mid]);

        t.left = createBST(inOrderKey, inOrderVal, start, mid - 1);

        t.right = createBST(inOrderKey, inOrderVal, mid + 1, end);

        return t;
    }

    public static void storeInord(BstNode root, int inOrderKey[], int
inOrderVal[]) {
        if (root != null) {
            storeInord(root.left, inOrderKey, inOrderVal);
            inOrderKey[idxkey++] = root.key;
            inOrderVal[idxval++] = root.val;
            storeInord(root.right, inOrderKey, inOrderVal);
        }

    }

    public int sizeOfTree(BstNode root) {
        if (root == null) {
            return 0;
        }

        // Calculate left size recursively
        int left = sizeOfTree(root.left);

        // Calculate right size recursively
        int right = sizeOfTree(root.right);

        // Return total size recursively
        return (left + right + 1);
    }

    int getSplittingIndex(int inOrder[], int k) {
        for (int i = 0; i < idxkey; i++) {
            if (inOrder[i] >= k) {
                return i - 1;
            }
        }
        return idxkey - 1;
    }

    public void splitBST(BstNode root, int k) {

        // Print the original BST
```

```
        System.out.print("Original BST : ");
        if (root != null) {
            PrintInord(root);
        }
        System.out.println();

        // Store the size of BST1
        int numNode = sizeOfTree(root);

        int[] inOrderKey = new int[numNode + 1];
        int[] inOrderval = new int[numNode + 1];
        idxkey = 0;
        idxval = 0;

        storeInord(root, inOrderKey, inOrderval);

        // splitting index
        int splitIndex = getSplittingIndex(inOrderKey, k);

        BstNode root1 = null;
        BstNode root2 = null;

        // Creation of first Balanced BST
        if (splitIndex != -1)
            root1 = createBST(inOrderKey, inOrderval, 0, splitIndex);

        // Creation of Second Balanced BST
        if (splitIndex != (idxkey - 1))
            root2 = createBST(inOrderKey, inOrderval, splitIndex + 1, idxkey -
1);


        System.out.print("First balanced BST : ");
        if (root1 != null) {
            PrintInord(root1);
        }
        System.out.println();

        System.out.print("Second balanced BST : ");
        if (root2 != null) {
            PrintInord(root2);
        }
    }

    public void PrintInord(BstNode root) {
        if (root != null) {
            PrintInord(root.left);
            System.out.print(root.key + " ---> " + root.val + " , ");
            PrintInord(root.right);
        }
    }

    public static void main(String[] args) {
        BST b = new BST();

        Scanner sc = new Scanner(System.in);
```

```
        while (true) {
            System.out.println("1. For insert node in BST");
            System.out.println("2. For Split at Root");
            System.out.println("3. For Split at Particular Node");
            System.out.println("4. For Exit");
            int ch = sc.nextInt();

            switch (ch) {
                case 1:
                    System.out.println("Enter key : ");
                    int key = sc.nextInt();
                    System.out.println("Enter val : ");
                    int val = sc.nextInt();
                    b.insert(key, val);
                    break;
                case 2:
                    b.splitBST(b.root, b.root.key);
                    break;
                case 3:
                    System.out.println("Enter key to split at Particular Node :
");
                    int nkey = sc.nextInt();
                    b.splitBST(b.root, nkey);
                    break;
                case 4:
                    System.exit(0);
            }

        }
    }
}
```

OUTPUT

```
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
50
Enter val :
0
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
40
Enter val :
1
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
60
Enter val :
2
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
30
Enter val :
3
```

```
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
45
Enter val :
4
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
60
Enter val :
5
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
55
Enter val :
6
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
1
Enter key :
58
Enter val :
7
```

```
1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
2
Original BST : 30 ---> 3 , 40 ---> 1 , 45 ---> 4 , 50 ---> 0 , 55 ---> 6 , 58 ---> 7 , 60 ---> 2 , 60 ---> 5 ,
First balanced BST : 30 ---> 3 , 40 ---> 1 , 45 ---> 4 ,
Second balanced BST : 50 ---> 0 , 55 ---> 6 , 58 ---> 7 , 60 ---> 2 , 60 ---> 5 , 1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
3
Enter key to split at Particular Node :
40
Original BST : 30 ---> 3 , 40 ---> 1 , 45 ---> 4 , 50 ---> 0 , 55 ---> 6 , 58 ---> 7 , 60 ---> 2 , 60 ---> 5 ,
First balanced BST : 30 ---> 3 ,
Second balanced BST : 40 ---> 1 , 45 ---> 4 , 50 ---> 0 , 55 ---> 6 , 58 ---> 7 , 60 ---> 2 , 60 ---> 5 , 1. For insert node in BST
2. For Split at Root
3. For Split at Particular Node
4. For Exit
4
```