IARE

IARE
INSTITUTE OF AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : Racherla Santhosh

Class : IT-B Semester : 03

Course Code : ACSD11 Course Name : DS laboratory

Name of the Course Faculty : Ms. K. Laxminarayanamma Faculty ID : IARE10033

Exercise Number : 10 Week Number : 10 Date : 12/11/2024

Roll Number									
2	3	9	5	1	A	1	2	C	3

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure / Performance in the Lab		Source Code / Calculations and Graphs	Program Execution / Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	10.1	Searching in BST	4	4	-	4	4	4	20
2	10.2	Find the Node with Min. value in a BST							
3	10.3	check if a Binary Tree is a BST or Not							
4	10.4	Second Largest Element in BST.							
5	10.5	Insertion in BST							
6									
7									
8									
9									
10									
11									
12									

Santhosh

Signature of the Student

Signature of the Faculty

10. Binary Search Tree (BST).

10.1 Searching In Binary Search Tree :-

AIM :- Write a program on the task is to delete a node in this BST. For Searching a value in BST, Consider it as a Sorted array. Perform Search operations in BST using Binary Search Algorithm.

PROGRAM :-

```
import java.util.Scanner;
class Node {
  int key;
  Node left, right;
  Public Node (int item) {
      key = item;
  } left = right = null;
}
```

```
class BST {
  Node root;
  Public BST() {
    root = null;
  }
  void insert (int key) {
    root = insertRec (root, key);
  }
  Node insertRec (Node root, int key) {
    if (root == null) {
      root = new Node (key);
      return root;
    }
    if (key < root.key) {
      root.left = insertRec (root.left, key);
    } else if (key > root.key) {
      root.right = insertRec (root.right, key);
    }
    return root;
  }
```

```
Node Search (int Key) {
    return SearchRec (root, Key);
}

Node SearchRec (Node root, int Key) {
    if (root == null || root.Key == Key) {
        return root;
    }
    if (Key > root.Key) {
        return SearchRec (root.right, Key);
    }
    return SearchRec (root.left, Key);
}

void delete (int Key) {
    root = deleteRec (root, Key);
}

Node deleteRec (Node root, int Key) {
    if (root == null) {
        return root;
    }
```

```
if (key < root.key) {
    root.left = deleteRec(root.left, key);
} else if (key > root.key) {
    root.right = deleteRec(root.right, key);
} else {
    if (root.left == null) {
        return root.right;
    } else if (root.right == null) {
        return root.left;
    }
    root.key = minValue(root.right);
    root.right = deleteRec(root.right, root.key);
}
return root;
}

Public static void main(String[] args) {
    BST tree = new BST();
    Scanner scanner = new Scanner(System.in);
```

```
while (true) {
    System.out.println("\n Binary Search Tree Operations:");
    System.out.println("1. Insert a node");
    System.out.println("2. Search for a node");
    System.out.println("3. Delete a node");
    System.out.println("4. print inorder traversal");
    System.out.println("5. Exit");
    System.out.print("choose an option (1-5): ");
    int choice = Scanner.nextInt();
    Switch (choice) {
        case 1:
            System.out.print("Enter the value to Insert: ");
            int insertValue = Scanner.nextInt();
            tree.insert(insertValue);
            System.out.println("Inserted" + insertValue + "into
                                the BST");
            break;
```

```
case 2:
    System.out.print("Enter the value to Search:");
    int SearchValue = Scanner.nextInt();
    Node result = tree.Search(SearchValue);
    if (result != null) {
    System.out.println("Node with key" + SearchValue +
                       "found in the BST");
} else {
    System.out.println("Node with key" + SearchValue +
                       "not found in the BST.");
};
break;
case 3:
System.out.print("Enter the value to delete:");
int deleteValue = Scanner.nextInt();
tree.delete(deleteValue);
System.out.println("Deleted" + deleteValue +
                   "from the BST");
break;
```

```
case 4:
    System.out.println("Inorder traversal of the BST:");
    tree.inorder();
    break;
case 5:
    System.out.println("Exiting Program");
    scanner.close();
    return;
default:
    System.out.println("Invalid choice. Please choose
                        a valid option");
    }
}
}
```

OUTPUT :-

Binary Search Tree Operations :

1. Insert a node
2. Search for a node
3. Delete a node

4. Print inorder traversal

5. Exit

Choose an option (1-5) : 1

Enter the value to insert : 10

Inserted 10 into the BST.

Choose an option (1-5) : 1

Enter the value to insert : 20

Inserted 20 into the BST.

Choose an option (1-5) : 1

Enter the value to insert : 30

Inserted 30 into the BST.

Choose an option (1-5) : 2

Enter the value to Search: 30

Node with Key 30 in the BST.

Choose an option (1-5) : 4

Inorder traversal of the BST :

10 20 30 40

choose an option (1-5) : 3

Enter the value to delete : 20

Delete 20 from the BST.

choose an option (1-5) : 5

Exiting Program.

10.2 Find the Node with Minimum Value in a BST :

AIM :- Write a function to find the node with minimum value in a Binary Search Tree.

PROGRAM :-

```
import java.util.Scanner;
class Node {
  int data;
  Node left, right;
  Public Node (int item) {
      data = item;
      left = right = null; } }
```

```
class BinarySearchTree {
    Node root;
    Public BinarySearchTree() {
        root = null;
    }
    void insert(int data) {
        root = insertRec(root, data);
    }
    Node insertRec(Node root, int data) {
        if (root == null) {
            root = new Node(data);
            return root;
        }
        if (data < root.data) {
            root.left = insertRec(root.left, data);
        } else if (data > root.data) {
            root.right = insertRec(root.right, data);
        }
        return root;
    }
```

```
Public static void main (String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    Scanner scanner = new Scanner (System.in);
    System.out.println (" Enter the number of nodes you
                        want to insert : ");
    int n = scanner.nextInt();
    System.out.println (" Enter the values to insert into
                        the BST : ");
    for (int i=0; i<n; i++) {
        int value = scanner.nextInt();
        tree.insert (value);
    }
    try {
        System.out.println (" The minimum value in the BST
                            is : "+ tree.findMinValue());
    } catch (IllegalArgumentException e) {
        System.out.println (e.getMessage());
```

```
    }
    Scanner.close();
  }
}
```

OUTPUT :-

```
Enter the number of nodes you want to insert : 5
Enter the values to insert into the BST :
10 20 30 40 50
The minimum value in the BST is : 10 .
```

10.3 Check if a Binary Tree is BST or not :-

AIM :- Write a Program on a Binary Search Tree which is a node-based binary tree data structure with some properties.

PROGRAM :-

```
import java.util.Scanner;
class Node {
  int data;
  Node left, right;
```

```
Public Node (int item) {
    data = item;
    left = right = null;
  }
}

class BinaryTree {
  boolean isBST(Node node) {
    return isBSTUtil(node, Integer.MIN_VALUE,
  }                        Integer.MAX_VALUE);
boolean isBSTUtil(Node node, int min, int max) {
  if (node == null) {
    return true;
  }
if (node.data <= min || node.data >= max) {
    return false;
  }
return isBSTUtil(node.left, min, node.data) &&
       isBSTUtil(node.right, node.data, max);
}
```

```
Public Static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
    Scanner Scanner = new Scanner (System.in);
    Node root = null;
    System.out.print(" Enter the number of nodes:");
    int n = Scanner.nextInt();
    System.out.println("Enter the values of the nodes:");
    for (int i=0; i<n; i++) {
        int value = Scanner.nextInt();
        root = tree.insert(root, value);
    }
    if (tree.isBST(root)) {
        System.out.println("Is BST");
    } else {
        System.out.println("Not a BST");
    }
    Scanner.close();
} }
```

OUTPUT :-

Enter the number of nodes : 5

Enter the values of the nodes :

20 8 22 4 12

IS BST.

10.4 Second Largest Element in BST :-

AIM :- Write a Program on given a Binary Search Tree (BST), find the Second largest element.

PROGRAM :-

```
import java.util.Scanner;
class Node {
    int Key;
    Node left, right;
    Public Node (int item) {
        Key = item;
        left = right = null;
    }
}
```

```
class BinarySearchTree {
 int count = 0;
 void SecondLargestUtil (Node node) {
  if (node == null) {
   return;
  }
  SecondLargestUtil (node.right);
  count++;
  if (count == 2) {
   System.out.println ("The second largest element is " +
   return;                    node.key);
  }
  SecondLargestUtil Node (node.left);
}
Node insert (Node root, int key) {
 if (root == null) {
  return new Node (key);
 }
 if (key < root.key) {
  root.left = insert (root.left, key);
```

```
} else if (key > root.key) {
    root.right = insert(root.right, key);
}
return root;
}
Public Static void main (String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    Scanner Scanner = new Scanner(System.in);
    Node root = null;
    System.out.print("Enter the number of nodes: ");
    int n = Scanner.nextInt();
    System.out.println("Enter the values of the nodes: ");
    for (int i=0; i<n; i++) {
        int value = Scanner.nextInt();
        root = tree.insert(root, value);
    }
    tree.SecondLargest(root);
    Scanner.close();    }    }
```

OUTPUT :-

Enter the number of nodes : 5

Enter the value of the nodes : 10 20 25 35 7

The Second largest element is 25.

10.5 Insertion in Binary Search Tree (BST) :-

AIM :- Write a program on a given Binary Search Tree, the task is to insert a new node in this BST.

PROGRAM :-

```
import java.util.Scanner;
class Node {
    int val;
    Node left, right;
    Public Node (int item) {
        val = item;
        left = right = null;
    }
}
class Binary Search Tree {
    Node root;
```

```
Public BinarySearchTree() {
    root = null;
}

void insert(int val) {
    root = insertRec(root, val);
}

Node insertRec(Node root, int val) {
    if (root == null) {
        root = new Node(val);
        return root;
    }
    if (val < root.val) {
        root.left = insertRec(root.left, val);
    } else if (val > root.val) {
        root.right = insertRec(root.right, val);
    }
    return root;
}

void inorder() {
```

```
    inorderRec(root);
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    BinarySearchTree tree = new BinarySearchTree();
    System.out.println("Enter the num of nodes to insert:");
    int n = scanner.nextInt();
    System.out.println("Enter the values of the nodes:");
    for (int i=0; i<n; i++) {
        int value = scanner.nextInt();
        tree.insert(value);
    }
    System.out.println("Inorder traversal of the BST:");
    tree.inorder();
    scanner.close(); } }
```

OUTPUT :-

Enter the num of nodes you want to insert : 5

Enter the values of the nodes : 10 20 5 30 25

Inorder Traversal of the BST : 5 10 20 25 30.

VIVA VOCE :-

1) What is the time complexity of Searching for a value in BST?

A) The Time complexity is O(log n) for a balanced BST, where n is the number of nodes. In the worst case (unbalanced), it can be O(n).

2) How do you find the minimum value node in BST?

A) It is the leftmost node in the BST. Start at the root and keep moving to the left child until you reach a node with no left child.

3) How do you insert a node in a BST?

A) Start at the root. If the new value is less than the current node, move to the left child; if greater, move to the right. Continue until you find a null position to insert the new node.

4) What is the Second Largest Element in a BST?

A) The second largest element is the node just before the largest element in an inorder traversal (reverse).