Name: Shubham Patgavkar
Div: C Batch: C3
Roll No: 233074
PRN No: 22320100

Practical No: 2

Title: Create a binary search tree (BST) of and perform following operations: i) Insert ii) Display inorder iii) Search a node iv) Find height of the tree v) level wise display iv) Delete v) Mirror

Code:

```
class Node {
    int data;
    Node left, right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class binarysearchtreep {
    Node root;

    binarysearchtreep() {
        root = null;
    }

    public Node insert(int data, Node root) {
        if (root == null) {
            Node newNode = new Node(data);
            root = newNode;
            return root;
        } else if (root.data > data) {
            root.left = insert(data, root.left);
        } else {
            root.right = insert(data, root.right);
        }
        return root;
```

```
    }

    public boolean search(int data, Node root) {
        if (root == null) {
            return false;
        }
        if (root.data > data) {
            return search(data, root.left);
        } else if (root.data == data) {
            return true;
        } else {
            return search(data, root.right);
        }
    }

    public void inorder(Node root) {
        if (root == null)
            return;
        inorder(root.left);
        System.out.print(root.data + " ");
        inorder(root.right);
    }

    public Node inorderSuccessor(Node root) {
        while (root.left != null) {
            root = root.left;
        }
        return root;
    }

    public Node delete(Node root, int data) {
        if (root == null) {
            return root;
        }
        if (root.data > data) {
            root.left = delete(root.left, data);
        } else if (root.data < data) {
            root.right = delete(root.right, data);
        } else {
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
```

```
                return root.left;
            }
            Node successor = inorderSuccessor(root.right);
            root.data = successor.data;
            root.right = delete(root.right, successor.data);
        }
        return root;
    }

    public int height(Node root) {
        if (root == null) {
            return 0;
        }
        int heightLeft = height(root.left);
        int heightRight = height(root.right);
        return Math.max(heightLeft, heightRight) + 1;
    }

    public void printLevelOrder(Node root) {
        int h = height(root);
        for (int i = 1; i <= h; i++)
            printCurrentLevel(root, i);
    }

    public void printCurrentLevel(Node root, int level) {
        if (root == null)
            return;
        if (level == 1)
            System.out.print(root.data + " ");
        else if (level > 1) {
            printCurrentLevel(root.left, level - 1);
            printCurrentLevel(root.right, level - 1);
        }
    }

    public void printMirror() {
        root = mirror(root);
    }

    public Node mirror(Node root) {
        if (root == null)
            return root;
```

```
        Node left = mirror(root.left);
        Node right = mirror(root.right);

        root.left = right;
        root.right = left;

        return root;
    }

    public int getCol(int h) {
        if (h == 1)
            return 1;
        return getCol(h - 1) + getCol(h - 1) + 1;
    }

    public void printTree(int[][] M, Node root, int col, int row, int
height) {
        if (root == null)
            return;
        M[row][col] = root.data;
        printTree(M, root.left, col - (int) Math.pow(2, height - 2),
row + 1, height - 1);
        printTree(M, root.right, col + (int) Math.pow(2, height -
2), row + 1, height - 1);
    }

    public void treePrinter(binarysearchtreep tree) {
        int h = height(tree.root);
        int col = getCol(h);
        int[][] M = new int[h][col];
        printTree(M, tree.root, col / 2, 0, h);
        for (int i = 0; i < h; i++) {
            for (int j = 0; j < col; j++) {
                if (M[i][j] == 0)
                    System.out.print("  ");
                else
                    System.out.print(M[i][j] + " ");
            }
            System.out.println();
        }
    }
```

```
    public static void main(String[] args) {
        int treedata[] = { 500, 300, 200, 400, 700, 600, 800 };
        binarysearchtreep tree = new binarysearchtreep();
        for (int i = 0; i < treedata.length; i++) {
            tree.root = tree.insert(treedata[i], tree.root);
        }
        System.out.println("Printing tree:");
        tree.treePrinter(tree);
        System.out.println("\nInorder traversal:");
        tree.inorder(tree.root);
        if (tree.search(200, tree.root)) {
            System.out.println("\nFound");
        } else {
            System.out.println("\nNot found");
        }
        tree.delete(tree.root, 200);
        System.out.println("\nInorder traversal after deletion:");
        tree.inorder(tree.root);
        System.out.println("\nHeight of tree: " +
tree.height(tree.root));
        System.out.println("Level order traversal:");
        tree.printLevelOrder(tree.root);
        tree.printMirror();
        System.out.println("\nMirror:");
        tree.inorder(tree.root);
        System.out.println("\nPrinting tree after mirror
operation:");
        tree.treePrinter(tree);
    }
}
```

Output:

Printing tree:

```
          500
    300         700
200  400    600  800
```

Inorder traversal:

200 300 400 500 600 700 800

Found

Inorder traversal after deletion:
300 400 500 600 700 800
Height of tree: 3
Level order traversal:
500 300 700 400 600 800
Mirror:
800 700 600 500 400 300
Printing tree after mirror operation:
```
        500
  700       300
800   600   400
```