LAB-3

Q) Binary Search Tree

Ans)

```
import java.util.*;
import java.io.*;

class BinaryTree {

    private Node deleteNodeData(Node root, int value) {
        if (root == null) {
            return root;
        }

        if (value < root.value) {
            root.left = deleteNodeData(root.left, value);
        } else if (value > root.value) {
            root.right = deleteNodeData(root.right, value);
        } else {
            if ((root.left == null) || (root.right == null)) {
                Node temp;
                if (root.left != null) {
                    temp = root.left;
                } else {
                    temp = root.right;
                }

                if (temp == null) {
                    temp = root;
                    root = null;
                } else {
                    root = temp;
                }
                temp = null;
            } else {
                Node temp = minValueNode(root.right);
                root.value = temp.value;
                root.right = deleteNodeData(root.right, temp.value);
            }
        }

        if (root == null) {
            return root;
        }

        root.height = Math.max(calcheight(root.left), calcheight(root.right)) + 1;
        return root;
    }

    private Node insert(Node node, int value) {
        if (node == null) {
            // Using constructor class
            return (new Node(value));
        }

        if (value < node.value) {
            node.left = insert(node.left, value);
        } else {
            node.right = insert(node.right, value);
        }

        node.height = Math.max(calcheight(node.left), calcheight(node.right)) + 1;

        return node;
    }

    // Search node
    private Node search(Node node, int key)
    {
        if (node==null) {
            System.out.println();
            System.out.println("Node not found");
            System.out.println();
        }

        if (node.left == null && node.right == null) {
            System.out.println();
            System.out.println("Item not found");
            System.out.println();
            return node;
        }

        if (node.left.value > key) {
            return search(node.left, key);
        } else if (node.left.value < key) {
            return search(node.right, key);
        } else if (node.left.value == key) {
            System.out.println();
            System.out.println("Item found in the left node");
            System.out.println();
            return node;
        } else if (node.right.value == key) {
            System.out.println();
```

```
                System.out.println("Item found in the right node");
                System.out.println();
                return node;
            }
            return node;
        }


        public class Node { // Our constructor class
            private Node left, right;
            private int height = 1;
            private int value;

            private Node(int val) {
                this.value = val;
            }
        }

        private Node minValueNode(Node node) {
            Node current = node;
            while (current.left != null) {
                current = current.left;
            }
            return current;
        }


        private int calcheight(Node N) {
            if (N == null) {
                return 0;
            }
            return N.height;
        }

        public void traverseInOrder(Node root) {
            if (root != null) {
                traverseInOrder(root.left);
                System.out.printf("%d ", root.value);
                traverseInOrder(root.right);
            }
        }

        public void print(Node root) {

            if (root == null) {
                System.out.println("(XXXXXX)");
                return;
            }

            int height = root.height,
                width = (int) Math.pow(2, height - 1);


            List < Node > current = new ArrayList < Node > (1),
                next = new ArrayList < Node > (2);
            current.add(root);

            final int maxHalfLength = 4;
            int elements = 1;

            StringBuilder sb = new StringBuilder(maxHalfLength * width);
            for (int i = 0; i < maxHalfLength * width; i++) {
                sb.append(' ');
            }

            String textBuffer;

            // Iterating through height levels.
            for (int i = 0; i < height; i++) {

                sb.setLength(maxHalfLength * ((int) Math.pow(2, height - 1 - i) - 1));

                // Creating spacer space indicator.
                textBuffer = sb.toString();

                // Print tree node elements
                for (Node n: current) {

                    System.out.print(textBuffer);

                    if (n == null) {
                        System.out.print("        ");
                        next.add(null);
                        next.add(null);
                    } else {

                        System.out.printf("(%6d)", n.value);
                        next.add(n.left);
                        next.add(n.right);

                    }
```

```
                System.out.print(textBuffer);

            }

            System.out.println();

            // Print tree node extensions for next level.
            if (i < height - 1) {

                for (Node n: current) {
                    System.out.print(textBuffer);

                    if (n == null) {
                        System.out.print("        ");
                    } else {
                        System.out.printf("%s      %s",
                            n.left == null ? " " : "/", n.right == null ? " " : "\\");
                    }

                    System.out.print(textBuffer);

                }

                System.out.println();

            }

            elements *= 2;
            current = next;
            next = new ArrayList < Node > (elements);
        }
    }

    public static void main(String args[]) {
        BinaryTree t = new BinaryTree();
        Scanner in = new Scanner(System.in);
        int se;
        Node root = null;
        while (true) {
            System.out.println("(1) Insert");
            System.out.println("(2) Delete");
            System.out.println("(3) Search");

            try {
                BufferedReader bufferRead = new BufferedReader(new
InputStreamReader(System.in));
```

```
                    String s = bufferRead.readLine();

                    if (Integer.parseInt(s) == 1) {
                        System.out.print("Value to be inserted: ");
                        root = t.insert(root, Integer.parseInt(bufferRead.readLine()));
                        t.print(root);
                    } else if (Integer.parseInt(s) == 2) {
                        System.out.print("Value to be deleted: ");
                        root = t.deleteNodeData(root,
Integer.parseInt(bufferRead.readLine()));
                        t.print(root);
                    } else if (Integer.parseInt(s) == 3) {
                        System.out.print("Value to be searched: ");
                        se = in.nextInt();
                        t.search(root, se);
                    } else {
                        System.out.println("Invalid choice, try again!");
                        continue;
                    }

                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
}
```