Student ID: 202112047

IT623 - Lab Assignment 7

1. **Write a program to find the maximum height of the tree.**

   Code:

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

	Node(int x) {
		key = x;
		left = right = null;
	}
}

public class Program1 {
	Node root;

	int maxHeight(Node node) {
		if (node == null)
			return -1;
		else {
			int lDepth = maxHeight(node.left);

			int rDepth = maxHeight(node.right);

			if (lDepth > rDepth) {

				return (lDepth + 1);
			} else
```

```
                return (rDepth + 1);
            }
        }

        public static void main(String[] args) {
            Program1 p = new Program1();

            // 1st level
            p.root = new Node(10);

            // 2nd level
            p.root.left = new Node(2);
            p.root.right = new Node(3);

            // 3rd level
            p.root.left.left = new Node(4);
            p.root.left.right = new Node(5);
            p.root.right.left = new Node(6);
            p.root.right.right = new Node(7);

            // 4th level
            p.root.left.left.right = new Node(8);
            p.root.left.right.right = new Node(9);
            p.root.right.right.left = new Node(10);
            p.root.right.right.right = new Node(11);

            // 5th level
            p.root.left.left.right.right = new Node(12);

            System.out.println("Maximum height of the tree is " +
p.maxHeight(p.root));
        }
}
```

Output Snapshot:

2. **Write a program to find whether the given number is present in the tree or not.**

Code:

```
public class Program2 {
	Node root;

	boolean isPresent(Node node, int x) {
		if (node == null)
			return false;

		if (node.key == x)
			return true;

		if (isPresent(node.left, x))
			return true;
		else if (isPresent(node.right, x))
			return true;
		else
			return false;
	}
```

```
public static void main(String[] args) {
        Program2 p = new Program2();

        // 1st level
        p.root = new Node(1);

        // 2nd level
        p.root.left = new Node(2);
        p.root.right = new Node(3);

        // 3rd level
        p.root.left.left = new Node(4);
        p.root.left.right = new Node(5);
        p.root.right.left = new Node(6);
        p.root.right.right = new Node(7);

        // 4th level
        p.root.left.left.right = new Node(8);
        p.root.left.right.right = new Node(9);
        p.root.right.right.left = new Node(10);
        p.root.right.right.right = new Node(11);

        // 5th level
        p.root.left.left.right.right = new Node(12);

        int num = 5;

        if (p.isPresent(p.root, num))
                System.out.println("Yes");
        else
                System.out.println("No");
}
```

```
}
```

Output Snapshot:

3. Write a program to insert and delete nodes in a Binary search tree.

Code:

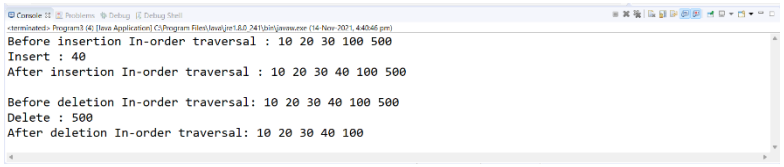
```
public class Program3 {

	Node root = null;

	void insert(int data) {
		root = insertdata(root, data);
	}

	Node insertdata(Node root, int data) {
		if (root == null) {
			root = new Node(data);
			return root;
		}

		if (data < root.key)
			root.left = insertdata(root.left, data);
		else if (data > root.key)
			root.right = insertdata(root.right, data);
```

```
        return root;
}

void inorder() {
        inorderRec(root);
}

void inorderRec(Node root) {
        if (root != null) {
                inorderRec(root.left);
                System.out.print(root.key + " ");
                inorderRec(root.right);
        }
}

void deletedata(int data) {
        root = deleteRec(root, data);
}

int minValue(Node root) {
        int minv = root.key;
        while (root.left != null) {
                minv = root.left.key;
                root = root.left;
        }
        return minv;
}

Node deleteRec(Node root, int data) {
        if (root == null)
                return root;

        if (data < root.key)
                root.left = deleteRec(root.left, data);
```

```
        else if (data > root.key)
                root.right = deleteRec(root.right, data);
        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[]) {
        Program3 p = new Program3();

        p.insert(100);
        p.insert(20);
        p.insert(500);
        p.insert(10);
        p.insert(30);

        System.out.print("Before insertion In-order traversal : ");
        p.inorder();

        System.out.println("\nInsert : 40 ");
        p.insert(40);

        System.out.print("After insertion In-order traversal : ");
        p.inorder();

        System.out.print("\n\nBefore deletion In-order traversal: ");
```

```
            p.inorder();

            System.out.println("\nDelete : 500");
            p.deletedata(500);

            System.out.print("After deletion In-order traversal: ");
            p.inorder();
        }
}
```

Output Snapshot:



4. Find median of BST.

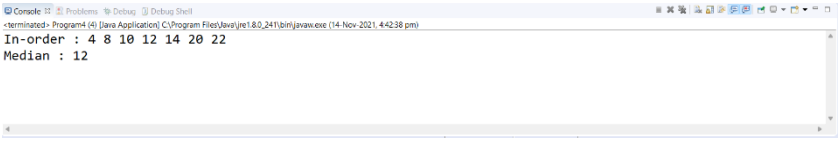
Code:

```
public class Program4 {
        Node root;
        int counter;

        void inOrder(Node node) {
            if (node == null)
                return;

            inOrder(node.left);
            System.out.print(node.key + " ");
```

```
        inOrder(node.right);
}

int counterNodes(Node head) {
        if (head != null) {
                return counterNodes(head.left) + counterNodes(head.right)
                + 1;
        }
        return 0;
}

public void getElements(Node head, int[] auxiliary) {
        if (head != null) {
                getElements(head.left, auxiliary);
                auxiliary[this.counter] += head.key;
                this.counter++;
                getElements(head.right, auxiliary);
        }
}

public void findMedian() {
        if (root != null) {

                int size = counterNodes(root);

                int[] auxiliary = new int[size];

                this.counter = 0;

                getElements(root, auxiliary);

                int result = 0;

                if (size % 2 != 0)
                        result = auxiliary[(size) / 2];
```

```
                else
                        result = (auxiliary[(size - 1) / 2] + auxiliary[(size) / 2]) /
2;
                System.out.print("\nMedian : " + result + " \n");
            }

        }

        public static void main(String[] args) {
            Program4 p = new Program4();

            // 1st level
            p.root = new Node(20);

            // 2nd level
            p.root.left = new Node(8);
            p.root.right = new Node(22);

            // 3rd level
            p.root.left.left = new Node(4);
            p.root.left.right = new Node(12);

            // 4th level
            p.root.left.right.left = new Node(10);
            p.root.left.right.right = new Node(14);

            System.out.print("In-order : ");
            p.inOrder(p.root);
            p.findMedian();
        }

}
```

Output Snapshot:



5. Find lowest common ancestor of 2 nodes.

Code:

```
public class Program5 {
	Node root;

	Node lca(Node node, int n1, int n2) {
		if (node == null)
			return null;

		if (node.key > n1 && node.key > n2)
			return lca(node.left, n1, n2);

		if (node.key < n1 && node.key < n2)
			return lca(node.right, n1, n2);

		return node;
	}
```

```
    public static void main(String[] args) {
        Program5 p = new Program5();

        // 1st level
        p.root = new Node(20);

        // 2nd level
        p.root.left = new Node(8);
        p.root.right = new Node(22);

        // 3rd level
        p.root.left.left = new Node(4);
        p.root.left.right = new Node(12);

        // 4th level
        p.root.left.right.left = new Node(10);
        p.root.left.right.right = new Node(14);

        int n1 = 10, n2 = 14;
        Node t = p.lca(p.root, n1, n2);
        System.out.println("Lowest Common Ancestor of " + n1 + " and " +
n2 + " is " + t.key);

        n1 = 4;
        n2 = 10;
        t = p.lca(p.root, n1, n2);
        System.out.println("Lowest Common Ancestor of " + n1 + " and " +
n2 + " is " + t.key);

        n1 = 10;
        n2 = 22;
        t = p.lca(p.root, n1, n2);
        System.out.println("Lowest Common Ancestor of " + n1 + " and " +
n2 + " is " + t.key);
```

```
        }

}
```

Output Snapshot: