Ques 1. Write a java program that inserts a node into its proper sorted position in a sorted linked list.

```java
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
class SortedLinkedList {
    Node head;
    SortedLinkedList() {
        this.head = null;
    }

    void insert(int data) {
        Node newNode = new Node(data);
        if (head == null || head.data >= newNode.data) {
            newNode.next = head;
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null && current.next.data < newNode.data) {
                current = current.next;
            }
```

```java
            newNode.next = current.next;
            current.next = newNode;
        }
    }

    void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        SortedLinkedList sortedList = new SortedLinkedList();
        sortedList.insert(5);
        sortedList.insert(10);
        sortedList.insert(2);
        sortedList.insert(7);

        System.out.println("Sorted Linked List:");
        sortedList.display();
```

```
    }
}
```

Ques 2. Write a java program to compute the height of the binary tree.

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

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

public class BinaryTreeHeight {
    Node root;

    BinaryTreeHeight() {
        root = null;
    }

    int getHeight(Node node) {
        if (node == null) {
            return 0;
        } else {
            int leftHeight = getHeight(node.left);
```

```
        int rightHeight = getHeight(node.right);

        // Return the height of the tree by adding 1 to the maximum height of left or right subtrees
        return Math.max(leftHeight, rightHeight) + 1;
      }
    }

    public static void main(String[] args) {
      BinaryTreeHeight tree = new BinaryTreeHeight();

      // Create a sample binary tree
      tree.root = new Node(1);
      tree.root.left = new Node(2);
      tree.root.right = new Node(3);
      tree.root.left.left = new Node(4);
      tree.root.left.right = new Node(5);
      tree.root.right.left = new Node(6);
      tree.root.right.right = new Node(7);

      int height = tree.getHeight(tree.root);
      System.out.println("Height of the binary tree is: " + height);
    }
}
```

Ques 3. Write a java program to determine whether a given binary tree is a BST or not.

```
class Node {
```

```
    int data;
    Node left, right;

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

public class CheckBST {
    Node root;

    CheckBST() {
        root = null;
    }

    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) {
        CheckBST tree = new CheckBST();

        // Create a sample binary tree (not a BST)
        tree.root = new Node(3);
        tree.root.left = new Node(2);
        tree.root.right = new Node(5);
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(4);

        if (tree.isBST(tree.root)) {
            System.out.println("The given binary tree is a BST.");
        } else {
            System.out.println("The given binary tree is not a BST.");
        }
    }
}
```

Ques 4. Write a java code to Check the given below expression is balanced or not . (using stack) { { [[(( ))]]) } }

```
import java.util.*;

public class BalancedExpression {

    static boolean isBalanced(String expression) {

        if (expression == null || expression.length() == 0) {

            return true; // Empty expression is considered balanced

        }


        Stack<Character> stack = new Stack<>();

        for (char ch : expression.toCharArray()) {

            if (ch == '(' || ch == '[' || ch == '{') {

                stack.push(ch);

            } else if (ch == ')' || ch == ']' || ch == '}') {

                if (stack.isEmpty()) {

                    return false; // Closing bracket without a corresponding opening bracket

                }


                char top = stack.pop();

                if ((ch == ')' && top != '(') || (ch == ']' && top != '[') || (ch == '}' && top != '{')) {

                    return false; // Mismatched opening and closing brackets

                }

            }

        }


        return stack.isEmpty(); // Expression is balanced if the stack is empty
```

```
    }

    public static void main(String[] args) {
        String expression = "{{[[(())]]}}";
        boolean isExpressionBalanced = isBalanced(expression);

        if (isExpressionBalanced) {
            System.out.println("The given expression is balanced.");
        } else {
            System.out.println("The given expression is not balanced.");
        }
    }
}
```

Ques 5. Write a java program to Print left view of a binary tree using queue.

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

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

```
}

public class LeftViewBinaryTree {
    Node root;

    LeftViewBinaryTree() {
        root = null;
    }

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

        Queue<Node> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                Node current = queue.poll();
                if (i == 0) {
                    System.out.print(current.data + " ");
                }
```

```
            if (current.left != null) {
                queue.add(current.left);
            }
            if (current.right != null) {
                queue.add(current.right);
            }
        }
    }
}

public static void main(String[] args) {
    LeftViewBinaryTree tree = new LeftViewBinaryTree();

    // Create a sample binary tree
    tree.root = new Node(12);
    tree.root.left = new Node(10);
    tree.root.right = new Node(30);
    tree.root.right.left = new Node(25);
    tree.root.right.right = new Node(40);

    System.out.println("Left view of the binary tree:");
    tree.leftView();
}
}
```