FULL STACK DEVELOPMENT – WORKSHEET - 6

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

Ans:-

Here is 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;
        next = null;
    }
}

class SortedLinkedList {
    Node head;

    SortedLinkedList() {
        head = null;
    }
```

```
void insert(int data) {
    Node newNode = new Node(data);

    if (head == null || data < head.data) {
        newNode.next = head;
        head = newNode;
    } else {
        Node current = head;
        while (current.next != null && current.next.data < data) {
            current = current.next;
        }

        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(7);
        sortedList.insert(3);
        sortedList.insert(8);

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

This program creates a sorted linked list and has a method `insert()` that inserts a node into its proper sorted position in the linked list. The `display()` method is used to print the elements of the linked list.

In the `insert()` method, the program checks if the list is empty or if the data is smaller than the head's data. If so, the new node becomes the new head. Otherwise, it traverses the linked list to find the correct position to insert the new node. It keeps moving forward as long as the next node has data smaller than the new node's data. Once found, it links the new node to the next node and the

current node to the new node.

In the `main()` method, we create a sorted linked list, insert some elements, and then display the sorted linked list.

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

Ans:-

Here is a Java program that computes the height of a binary tree:

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

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

class BinaryTree {
    Node root;

    BinaryTree() {
        root = null;
    }
```

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

            if (leftHeight > rightHeight) {
                return leftHeight + 1;
            } else {
                return rightHeight + 1;
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        // 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);
```

```
        int height = tree.height(tree.root);

        System.out.println("The height of the binary tree is: " + height);
    }
}
```

In this program, we create a binary tree with some sample data. The `height()` method takes a node as input and recursively calculates the height of the binary tree.

The basic idea behind the height calculation is to find the height of the left and right subtrees, and then return the maximum of the two tree heights plus 1 (to account for the current node). This is done using a recursive approach.

In the `main()` method, we create an instance of the `BinaryTree` class, build a sample binary tree, and then calculate its height using the `height()` method. The result is then printed to the console.

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

Ans:-

Here is a Java program that determines whether a given binary tree is a Binary Search Tree (BST) or not:

java

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

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

class BinaryTree {
    Node root;

    boolean isBST() {
        return isBSTUtil(root, 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 - 1) &&
               isBSTUtil(node.right, node.data + 1, max);
    }
}

public class Main {
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        // Create a sample binary tree
        tree.root = new Node(4);
        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(3);

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

In this program, we create a binary tree with some sample data. The `isBST()`

method is used to determine whether the binary tree is a BST or not. It calls the `isBSTUtil()` method, passing the root node and the minimum and maximum values (`Integer.MIN_VALUE` and `Integer.MAX_VALUE`) to check if the current node satisfies the BST property.

The `isBSTUtil()` method recursively checks if each subtree is a valid BST by comparing the values of its nodes with the minimum and maximum boundaries. If at any point the current node value is less than the minimum or greater than the maximum, the method returns `false`. Otherwise, it recursively checks the left and right subtrees.

In the `main()` method, we create an instance of the `BinaryTree` class, build a sample binary tree, and then check if it is a BST using the `isBST()` method. The result is then printed to the console.

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

{{[[(())]]}}

Ans:-

Here is a Java code that checks whether a given expression is balanced or not using a stack:

```
java
import java.util.Stack;

public class Main {
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<Character>();

        for (int i = 0; i < expression.length(); i++) {
            char c = expression.charAt(i);
```

```
            if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
            } else if (c == ')' || c == ']' || c == '}') {
                if (stack.isEmpty()) {
                    return false;
                }

                char top = stack.pop();

                if ((c == ')' && top != '(') || (c == ']' && top != '[')
|| (c == '}' && top != '{')) {
                    return false;
                }
            }
        }

        return stack.isEmpty();
    }

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

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

In this code, we use a stack to check the balance of parentheses, square brackets, and curly braces in the given expression. We iterate through each character of the expression and perform the following steps:

1. If the character is an opening bracket (i.e., '(', '[', or '{'), we push it onto the stack.
2. If the character is a closing bracket (i.e., ')', ']', or '}'), we check if the stack is empty. If it is, we return `false` because there is no corresponding opening bracket.
3. If the stack is not empty, we pop the top element from the stack and check if it matches the current closing bracket. If they do not

match, we return `false` because the brackets are not balanced.
4. After iterating through all the characters, if the stack is empty, it means that all opening brackets had corresponding closing brackets, and hence the expression is balanced. Otherwise, if the stack is not empty, it means that there are unmatched opening brackets, and hence the expression is not balanced.

In the `main()` method, we call the `isBalanced()` function with a sample expression `{{[[(())]]}}`. The result is then printed to the console. In this case, the output will be "The given expression is balanced."

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

Ans:-

Here's a Java program that prints the left view of a binary tree using a queue:

```
java
import java.util.LinkedList;
import java.util.Queue;

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

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

```
}

class BinaryTree {
    Node root;

    void printLeftView() {
        if (root == null)
            return;

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

        while (!queue.isEmpty()) {
            int n = queue.size();

            for (int i = 1; i <= n; i++) {
                Node temp = queue.poll();

                if (i == 1)
                    System.out.print(temp.data + " ");

                if (temp.left != null)
                    queue.add(temp.left);
```

```
            if (temp.right != null)
                queue.add(temp.right);
        }
    }
}

public static void main(String args[]) {
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.right = new Node(4);
    tree.root.right.left = new Node(5);
    tree.root.right.right = new Node(6);

    System.out.println("Left View of Binary Tree:");
    tree.printLeftView();
}
}
```

The program defines a Node class representing nodes in a binary tree. We have a BinaryTree class that has a root node. The `printLeftView()` method uses a queue to perform a level order traversal of the tree and print the leftmost node at each level.

In the `printLeftView()` method, we add the root node to the queue and start the while loop until the queue is empty. At each iteration, we process all the nodes in the current level. If the current node is the first node of the level (i.e., `i == 1`), we print its data. Then, we check if the current node has left and right children. If they exist, we add them to the queue.

In the `main()` method, we create a sample binary tree and call the `printLeftView()` method. The left view of the binary tree is printed to the console. In this case, the output will be:

Left View of Binary Tree:

1 2 4