**Worksheet A**

**1.**

public class SortedLinkedList {

static class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

static Node sortedInsert(Node head, Node newNode) {

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;

}

newNode.next = current.next;

current.next = newNode;

}

return head;

}

static void printList(Node head) {

Node current = head;

while (current != null) {

System.out.print(current.data + " ");

current = current.next;

}

System.out.println();

}

public static void main(String[] args) {

Node head = null;

head = sortedInsert(head, new Node(5));

head = sortedInsert(head, new Node(10));

head = sortedInsert(head, new Node(7));

head = sortedInsert(head, new Node(3));

head = sortedInsert(head, new Node(1));

printList(head);

}

}

**2.**

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinaryTree {

Node root;

public 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 static void main(String args[]) {

BinaryTree tree = new BinaryTree();

// Creating the 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);

System.out.println("Height of tree is : " + tree.height(tree.root));

}

}

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

**3.**

class BinaryTree {

Node root;

public boolean isBST() {

return isBSTUtil(root, Integer.MIN\_VALUE, Integer.MAX\_VALUE);

}

public 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 static void main(String args[]) {

BinaryTree tree = new BinaryTree();

// Creating the 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("Given binary tree is a BST");

else

System.out.println("Given binary tree is not a BST");

}

}

**4.**

import java.util.\*;

public class BalancedExpressionChecker {

// Function to check if the given expression is balanced or not

public static boolean isBalanced(String expression) {

// Create an empty stack to store opening brackets

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

// Loop through each character in the expression

for (int i = 0; i < expression.length(); i++) {

char c = expression.charAt(i);

// If the current character is an opening bracket, push it onto the stack

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

stack.push(c);

}

// If the current character is a closing bracket, check if it matches the top of the stack

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

// If the stack is empty or the top of the stack does not match the current character, the expression is not balanced

if (stack.isEmpty() || !matches(stack.pop(), c)) {

return false;

}

}

}

// If the stack is not empty, the expression is not balanced

return stack.isEmpty();

}

// Helper function to check if two brackets match

private static boolean matches(char open, char close) {

return (open == '(' && close == ')')

**5.**

import java.util.LinkedList;

import java.util.Queue;

class Node {

int data;

Node left, right;

public Node(int data) {

this.data = data;

left = right = null;

}

}

public class LeftViewOfBinaryTree {

Node root;

public LeftViewOfBinaryTree() {

root = null;

}

void printLeftView() {

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) {

LeftViewOfBinaryTree tree = new LeftViewOfBinaryTree();

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);

tree.root.right.left.left = new Node(7);

tree.printLeftView();

}

}