WORKSHEET

**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.

\* Online Java Compiler and Editor \*/

// A Linked List Node

class Node

{

    int data;

    Node next;

    Node(int data, Node next)

    {

        this.data = data;

        this.next = next;

    }

    Node(int data) {

        this.data = data;

    }

}

public class Main

{

    // Helper function to print a given linked list

    public static void printList(Node head)

    {

        Node ptr = head;

        while (ptr != null)

        {

            System.out.print(ptr.data + " —> ");

            ptr = ptr.next;

        }

        System.out.println("null");

    }

    // Function to insert a given node at its correct sorted position into

    // a given list sorted in increasing order

    public static Node sortedInsert(Node head, Node newNode)

    {

        // special case for the head end

        if (head == null || head.data >= newNode.data)

        {

            newNode.next = head;

            head = newNode;

            return head;

        }

        // locate the node before the point of insertion

        Node current = head;

        while (current.next != null && current.next.data < newNode.data) {

            current = current.next;

        }

        newNode.next = current.next;

        current.next = newNode;

        return head;

    }

    public static void main(String[] args)

    {

        // input keys

        int[] keys = {2, 3, 4, 6, 8};

        // points to the head node of the linked list

        Node head = null;

        // construct a linked list

        for (int i = keys.length - 1; i >= 0; i--) {

            head = new Node(keys[i], head);

        }

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

        head = sortedInsert(head, new Node(9));

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

        // print linked list

        printList(head);

    }

}

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

// Java program for above approach

// A class to store a binary tree node

class Node

{

    int key;

    Node left = null, right = null;

    Node(int key) {

        this.key = key;

    }

}

public class Main

{

    // Recursive function to calculate the height of a given binary tree

    public static int height(Node root)

    {

        // base case: empty tree has a height of 0

        if (root == null) {

            return 0;

        }

        // recur for the left and right subtree and consider maximum depth

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

    }

    public static void main(String[] args)

    {

        Node root = new Node(15);

        root.left = new Node(10);

        root.right = new Node(20);

        root.left.left = new Node(8);

        root.left.right = new Node(12);

        root.right.left = new Node(16);

        root.right.right = new Node(25);

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

    }

}

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

 A class to store a BST node

class Node

{

    int data;

    Node left = null, right = null;

    Node(int data) {

        this.data = data;

    }

}

public class Main

{

    // Recursive function to insert a key into a BST

    public static Node insert(Node root, int key)

    {

        // if the root is null, create a new node and return it

        if (root == null) {

            return new Node(key);

        }

        // if the given key is less than the root node, recur for the left subtree

        if (key < root.data) {

            root.left = insert(root.left, key);

        }

        // if the given key is more than the root node, recur for the right subtree

        else {

            root.right = insert(root.right, key);

        }

        return root;

    }

    // Function to determine whether a given binary tree is a BST by keeping a

    // valid range (starting from [-INFINITY, INFINITY]) and keep shrinking

    // it down for each node as we go down recursively

    public static boolean isBST(Node node, int minKey, int maxKey)

    {

        // base case

        if (node == null) {

            return true;

        }

        // if the node's value falls outside the valid range

        if (node.data < minKey || node.data > maxKey) {

            return false;

        }

        // recursively check left and right subtrees with an updated range

        return isBST(node.left, minKey, node.data) &&

            isBST(node.right, node.data, maxKey);

    }

    // Function to determine whether a given binary tree is a BST

    public static void isBST(Node root)

    {

        if (isBST(root, Integer.MIN\_VALUE, Integer.MAX\_VALUE)) {

            System.out.println("The tree is a BST.");

        }

        else {

            System.out.println("The tree is not a BST!");

        }

    }

    private static void swap(Node root)

    {

        Node left = root.left;

        root.left = root.right;

        root.right = left;

    }

    public static void main(String[] args)

    {

        int[] keys = { 15, 10, 20, 8, 12, 16, 25 };

        Node root = null;

        for (int key: keys) {

            root = insert(root, key);

        }

        // swap left and right nodes

        swap(root);

        isBST(root);

    }

}

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

(using stack)

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

import java.util.Stack;

public class Main

{

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

    public static boolean isBalanced(String exp)

    {

        // base case: length of the expression must be even

        if (exp == null || exp.length() % 2 == 1) {

            return false;

        }

        // take an empty stack of characters

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

        // traverse the input expression

        for (char c: exp.toCharArray())

        {

            // if the current character in the expression is an opening brace,

            // push it into the stack

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

                stack.push(c);

            }

            // if the current character in the expression is a closing brace

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

            {

                // return false if a mismatch is found (i.e., if the stack is empty,

                // the expression cannot be balanced since the total number of opening

                // braces is less than the total number of closing braces)

                if (stack.empty()) {

                    return false;

                }

                // pop character from the stack

                Character top = stack.pop();

                // if the popped character is not an opening brace or does not pair

                // with the current character of the expression

                if ((top == '(' && c != ')') || (top == '{' && c != '}')

                        || (top == '[' && c != ']')) {

                    return false;

                }

            }

        }

        // the expression is balanced only when the stack is empty at this point

        return stack.empty();

    }

    public static void main(String[] args)

    {

        String exp = "{ { [ [ ( ( ) ) ] ) } }";

        if (isBalanced(exp)) {

            System.out.println("The expression is balanced");

        }

        else {

            System.out.println("The expression is not balanced");

        }

    }

}

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

// Java program to print the

// left view of Binary Tree

import java.util.\*;

public class Main

{

// A Binary Tree Node

static class node

{

    int data;

    node left, right;

};

// A utility function to create a new

// Binary Tree node

static node newNode(int item)

{

    node temp = new node();

    temp.data = item;

    temp.left = null;

    temp.right = null;

    return temp;

}

static Queue<node> q;

// Utility function to print the left view of

// the binary tree

static void leftViewUtil( node root )

{

    if (root == null)

        return;

    // add root

    q.add(root);

    // Delimiter

    q.add(null);

    while (q.size() > 0)

    {

        node temp = q.peek();

        if (temp != null)

        {

            // Prints first node

            // of each level

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

            // add children of all nodes at

            // current level

            while (q.peek() != null)

            {

                // If left child is present

                // add into queue

                if (temp.left != null)

                    q.add(temp.left);

                // If right child is present

                // add into queue

                if (temp.right != null)

                    q.add(temp.right);

                // remove the current node

                q.remove();

                temp = q.peek();

            }

            // add delimiter

            // for the next level

            q.add(null);

        }

        // remove the delimiter of

        // the previous level

        q.remove();

    }

}

// Function to print the leftView

// of Binary Tree

static void leftView( node root)

{

    // Queue to store all

    // the nodes of the tree

    q = new LinkedList<node>();

    leftViewUtil(root);

}

// Driver Code

public static void main(String args[])

{

    node root = newNode(10);

    root.left = newNode(12);

    root.right = newNode(3);

    root.left.right = newNode(4);

    root.right.left = newNode(5);

    root.right.left.right = newNode(6);

    root.right.left.right.left = newNode(18);

    root.right.left.right.right = newNode(7);

    leftView(root);

}

}