Worksheet – 6

Q1. Write a java program that inserts a node into its proper sorted position in a sorted linked list.

```
import java.util.*;
import java.lang.*;
import java.io.*;
public class Main{

 public static void main (String[] args) throws java.lang.Exception{

  MyLinkedList obj = new MyLinkedList();
  obj.addAtHead(10);
  obj.addAtHead(7);
  obj.addAtHead(4);
  obj.printList();

  obj.sortedInsert(2);
  obj.sortedInsert(5);
  obj.sortedInsert(15);
  obj.printList();

 }

 public static class MyLinkedList {

   class Node{
     Node next = null;
     int val = 0;
```

```
    public Node(int val){
        this.val = val;
    }
}

private Node head;
private int size;
public MyLinkedList() {
    this.head = null;
    this.size = 0;
}

public Node getNodeAt(int index){
    Node curr = head;
    while(index-- > 0){
        curr = curr.next;
    }

    return curr;
}

public void addAtHead(int val) {
    Node node = new Node(val);
    if(this.size == 0){
        this.head = node;
    }
    else{
        node.next = this.head;
        this.head = node;
    }
    this.size++;
}

public void sortedInsert(int val) {
    Node newNode = new Node(val);
    Node previous = null;
```

```
        Node current = head;

        while (current != null && val > current.val) {
            previous = current;
            current = current.next;
        }

        newNode.next = current;
      if (previous == null)
        head = newNode;
      else
        previous.next = newNode;

      }
    public void printList(){
      Node curr = head;
      while(curr!=null){
        System.out.print(curr.val+" ");
        curr = curr.next;
      }
      System.out.println("");
    }

  }
}
```

Q2. Write a java program to compute the height of the binary tree.

```
import java.util.*;

public class Main
{

    // Binary tree class
    public static class BinaryTree
    {
```

```
        // Node class
        public class Node
        {

            int data;
            Node left;
            Node right;

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

        private Node root;

        public BinaryTree (int[]pre, int[]post)
        {
            this.root =
                    this.construct (pre, 0, pre.length - 1, post, 0,
post.length - 1);
        }

        private Node construct (int[]pre, int presi, int preei,
int[]post,int postsi, int postei)
        {

// this case occurs when a node has only one child
            if (presi > preei)
            {
                return null;
            }

            Node node = new Node (pre[presi]);
            node.left = null;
            node.right = null;

            if (presi == preei)
            {
                return node;
            }

//Searching pre[presi + 1] in postorder array
            int pos = -1;
            for (int i = postsi; i <= postei; i++)
            {
                if (post[i] == pre[presi + 1])
                {
                    pos = i;
                    break;
                }
            }

//Number of elements in left subtree
            int clc = pos-postsi + 1;

//Left subtree
            node.left =
                    this.construct (pre, presi + 1, presi + clc, post,
```

```
postsi, pos);

//Right subtree
            node.right =
                    this.construct (pre, presi + clc + 1, preei, post, pos
+ 1,postei - 1);

            return node;
        }

        public int height ()
        {
            return this.height (this.root);
        }

        //Function to find height of binary tree
        private int height (Node node)
        {

//Base case
            if (node == null)
            {
                return -1;
            }

//Calculate height of left subtree
            int lht = this.height (node.left);

//Calculate height of right subtree
            int rht = this.height (node.right);

//Height of the tree is max of left and right subtree height + 1 (for the
node itself)
            int rv = Math.max (lht, rht) + 1;
            return rv;
        }
    }

    public static void main (String[]args) throws Exception
    {

// Construct binary tree

        int[] pre = { 50, 25, 12, 37, 30, 40, 75, 62, 60, 70, 87 };
        int[] post = { 12, 30, 40, 37, 25, 60, 70, 62, 87, 75, 50 };

        BinaryTree bt = new BinaryTree (pre, post);
        System.out.println ("Height of the tree is : "+bt.height ());

    }
}
```

Q3. Write a java program to determine whether a given binary tree is a BST or not.

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

    Node(int data) {
        this.data = data;
    }
}

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);
    }
}
```

Q4. Write a java code to Check the given below expression is balanced or not .

(using stack)

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

```
import java.util.*;
class Main
{
    public static void main(String[] args)
    {
        String expression;
        int i, length;
        char ch;
        Scanner s = new Scanner(System.in);

        System.out.print("Enter the Expression: ");
        expression = s.next();

        Stack<Character> stack = new Stack<Character>();
        length = expression.length();

        for(i=0; i<length; i++)
```

```
        {
            ch = expression.charAt(i);
            if(ch=='(' || ch=='{' || ch=='[')
            {
                stack.push(ch);
            }
            else if(ch==')')
            {
                if(stack.isEmpty() || stack.pop() != '(')
                {
                    System.out.println("\nUnbalanced Parentheses!");
                    return;
                }
            }
            else if(ch=='}')
            {
                if(stack.isEmpty() || stack.pop() != '{')
                {
                    System.out.println("\nUnbalanced Parentheses!");
                    return;
                }
            }
            else if(ch==']')
            {
                if(stack.isEmpty() || stack.pop() != '[')
                {
                    System.out.println("\nUnbalanced Parentheses!");
                    return;
                }
            }
        }
        if(stack.isEmpty())
            System.out.println("\nBalanced Parentheses.");
    }
}
```

Q5. Write a java program to Print left view of a binary tree using queue.

```
import java.util.ArrayDeque;
import java.util.Queue;

// A class to store a binary tree node
class Node
{
    int key;
    Node left = null, right = null;

    Node(int key) {
        this.key = key;
    }
}

class Main
```

```
{
    // Iterative function to print the left view of a given binary tree
    public static void leftView(Node root)
    {
        // return if the tree is empty
        if (root == null) {
            return;
        }

        // create an empty queue and enqueue the root node
        Queue<Node> queue = new ArrayDeque<>();
        queue.add(root);

        // to store the current node
        Node curr;

        // loop till queue is empty
        while (!queue.isEmpty())
        {
            // calculate the total number of nodes at the current level
            int size = queue.size();
            int i = 0;

            // process every node of the current level and enqueue their
            // non-empty left and right child
            while (i++ < size)
            {
                curr = queue.poll();

                // if this is the first node of the current level, print it
                if (i == 1) {
                    System.out.print(curr.key + " ");
                }

                if (curr.left != null) {
                    queue.add(curr.left);
                }

                if (curr.right != null) {
                    queue.add(curr.right);
                }
            }
        }
    }

    public static void main(String[] args)
    {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.right = new Node(4);
        root.right.left = new Node(5);
        root.right.right = new Node(6);
        root.right.left.left = new Node(7);
        root.right.left.right = new Node(8);

        leftView(root);
    }
}
```