FULL STACK DEVELOPMENT – WORKSHEET- A

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

<u>Program :</u>

```
package com.java.Trees;

public class LinkedList {

        Node head; // head of list

  /* Linked list Node*/
  class Node {
    int data;
    Node next;
    Node(int d)
    {
      data = d;
      next = null;
    }
  }

  /* function to insert a
new_node in a list. */
  void sortedInsert(Node new_node)
  {
    Node current;

    /* Special case for head node */
    if (head == null || head.data
>= new_node.data) {
      new_node.next = head;
      head = new_node;
    }
    else {

      /* Locate the node before point of insertion. */
      current = head;

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

        current = current.next;
      }

      new_node.next = current.next;
```

```
            current.next = new_node;
        }
    }

    /*Utility functions*/

    /* Function to create a node */
    Node newNode(int data)
    {
        Node x = new Node(data);
        return x;
    }

    /* Function to print linked list */
    void printList()
    {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
    }

    /* Driver function to test above methods */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();
        Node new_node;
        new_node = llist.newNode(115);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(110);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(7);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(3);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(12);
        llist.sortedInsert(new_node);
        new_node = llist.newNode(9);
        llist.sortedInsert(new_node);
        System.out.println("Created Linked List");
        llist.printList();
    }

}
```

Output :

Created Linked List
3 7 9 12 110 115

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

<u>Program :</u>

```
package com.java.Trees;

 class Node {
            int data;
    Node left, right;

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

class BinaryTree {
    Node root;

    /* Compute the "maxDepth" of a tree -- the number of
       nodes along the longest path from the root node
       down to the farthest leaf node.*/
    int maxHeight(Node node)
    {
        if (node == null)
            return 0;
        else {
            /* compute the depth of each subtree */
            int lHeight = maxHeight (node.left);
            int rHeight = maxHeight (node.right);

            /* use the larger one */
            if (lHeight > rHeight)
                return (lHeight + 1);
            else
                return (rHeight + 1);
        }
    }

    /* Driver program to test above functions */
    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.left = new Node(4);
        tree.root.left.right = new Node(5);


        System.out.println("Height of tree is "
                + tree. maxHeight (tree.root));
    }
}
```

Output :

Height of tree is 3

Q) 3. Write a java program to determine whether a given binary tree is a BSTor not.

Program :

```
package com.java.Trees;

public class Node1 {
        int data;
        Node1 left, right;


        /* Helper function that allocates a new node with the
           given data and NULL left and right pointers. */
        static Node1 newNode(int data)
        {
                Node1 Node = new Node1();
          Node.data = data;
          Node.left = Node.right = null;

          return Node;
        }

        static int maxValue(Node1 Node)
        {
          if (Node == null) {
            return Integer.MIN_VALUE;
          }
          int value = Node.data;
          int leftMax = maxValue(Node.left);
          int rightMax = maxValue(Node.right);

          return Math.max(value, Math.max(leftMax, rightMax));
```

```
}

static int minValue(Node1 Node)
{
  if (Node == null) {
    return Integer.MAX_VALUE;
  }
  int value = Node.data;
  int leftMax = minValue(Node.left);
  int rightMax = minValue(Node.right);

  return Math.min(value, Math.min(leftMax, rightMax));
}

/* Returns true if a binary tree is a binary search tree
   */
Static nt isBST(Node1 Node)
{
  if (Node == null) {
    return 1;
  }

  /* false if the max of the left is > than us */
  if (Node.left != null
     && maxValue(Node.left) > Node.data) {
   return 0;
  }

  /* false if the min of the right is <= than us */
  if (Node.right != null
     && minValue(Node.right) < Node.data) {
   return 0;
  }

  /* false if, recursively, the left or right is not a
      * BST*/
  if (isBST(Node.left) != 1
     || isBST(Node.right) != 1) {
    return 0;
  }

  /* passing all that, it's a BST */
  return 1;
}

public static void main(String[] args)
{
```

```
            Node1 root = newNode(4);
            root.left = newNode(2);
            root.right = newNode(5);
            root.left = newNode(32);

            // root->right->left = newNode(7);
            root.left.left = newNode(1);
            root.left.right = newNode(3);

            // Function call
            if (isBST(root) == 1) {
              System.out.print("Is BST");
            }
            else {
              System.out.print("Not a BST");
            }
          }
}
```

Output :

Not a BST

Q) 4. Write a java code to Check the given below expression is balancedor not .

(using stack) { { [[(( ))]]) } } .

Program :

```
package com.java.Trees;

import java.util.Stack;

public class Main {
          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");
        }
    }
}
```

<u>Output :</u>

The expression is not balanced

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

Program :

```
package com.java.Trees;

import java.util.Queue;

import java.util.LinkedList;

public class PrintRightView {

        private static class Node {
      int data;
      Node left, right;

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

   // function to print left view of binary tree
   private static void printLeftView(Node root)
   {
      if (root == null)
         return;

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

      while (!queue.isEmpty()) {
         // number of nodes at current level
         int n = queue.size();

         // Traverse all nodes of current level
         for (int i = 1; i <= n; i++) {
            Node temp = queue.poll();

            // Print the left most element at
            // the level
            if (i == 1)
               System.out.print(temp.data + " ");
```

```
            // Add left node to queue
            if (temp.left != null)
                queue.add(temp.left);

            // Add right node to queue
            if (temp.right != null)
                queue.add(temp.right);
        }
    }
}

// Driver code
public static void main(String[] args)
{

    Node root = new Node(10);
    root.left = new Node(2);
    root.right = new Node(3);
    root.left.left = new Node(7);
    root.left.right = new Node(8);
    root.right.right = new Node(15);
    root.right.left = new Node(12);
    root.right.right.left = new Node(14);

    printLeftView(root);
}
}
```

Output :

10 2 7 14