Practice Problems Solutions

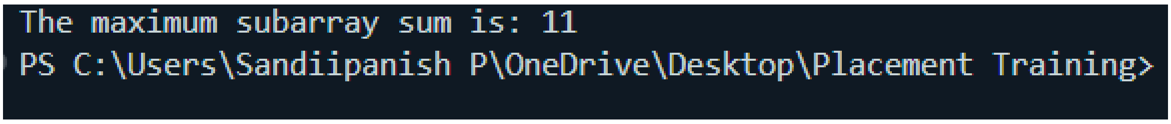
Day 1 Set

[09/11/2024]
1.

Code:

```
import java.util.*;

public class maxSubarraySum {
   static long maxSubarraySum(int[] arr, int n) {
      long max = Long.MIN_VALUE;
      long sum = 0;

      for (int i = 0; i < n; i++) {
         sum += arr[i];
         if (sum > max) {
            max = sum;
         }
         if (sum < 0) {
            sum = 0;
         }
      }
      return max;
   }

   public static void main(String args[]) {
      int[] arr = {2, 3, -8, 7, -1, 2, 3};
      int n = arr.length;
      long maxSum = maxSubarraySum(arr, n);
      System.out.println("The maximum subarray sum is: " + maxSum);

   }

}
```

Ouput:



```
The maximum subarray sum is: 11
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

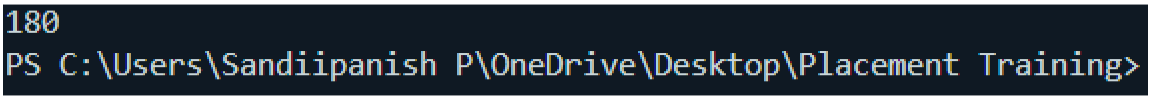
Time Complexity: O(n)

2.

Code:

```
public class MaximumProduct{
    static int MaximumProduct(int arr[]) {
          int n = arr.length;
        int result = arr[0];
        for (int i = 0; i < n; i++) {
            int mul = 1;
            for (int j = i; j < n; j++) {
                mul *= arr[j];
                result = Math.max(result, mul);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        int arr[] = { -2, 6, -3, -10, 0, 2 };
        System.out.println(MaximumProduct(arr));
    }
}
```

Output:



```
180
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Complexity: O(n^2)

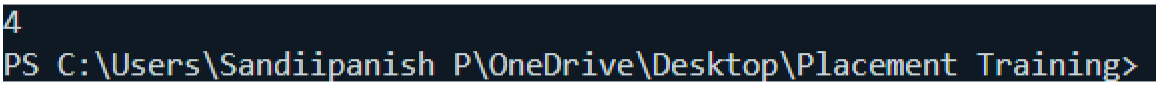
3.

Code:

```
public class RotatedArraySearch {

  public static int searchInRotatedArray(int[] array, int target) {
    int start = 0, end = array.length - 1;

    while (start <= end) {
      int mid = start + (end - start) / 2;

      if (array[mid] == target) {
        return mid;
      }

      if (array[start] <= array[mid]) {
        if (array[start] <= target && target < array[mid]) {
          end = mid - 1;
        } else {
          start = mid + 1;
        }
      } else {
        if (array[mid] < target && target <= array[end]) {
          start = mid + 1;
        } else {
```

```
                    end = mid - 1;
                }
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        int[] array = {4, 5, 6, 7, 0, 1, 2};
        System.out.println(searchInRotatedArray(array, 0));

    }
}
```

Output:



```
4
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

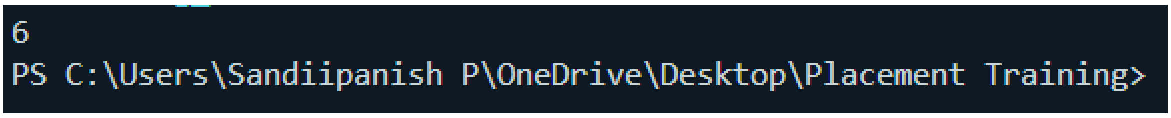
Time Complexity:O(n)

4.

Code:

```
public class containerwithmostwater {
    public static int maxarea(int[] arr) {
        int l = 0, r = arr.length - 1;
        int maxarea = 0;
        while (l < r) {
            int h = Math.min(arr[l], arr[r]);
            int w = r - l;
            int ar = h * w;
            maxarea = Math.max(maxarea, ar);
            if (arr[l] < arr[r]) {
                l++;
            } else {
                r--;
            }
        }
        return maxarea;
    }
    public static void main(String[] args) {
        int[] arr = {1, 5, 4, 3};
        System.out.println(maxarea(arr));
    }
}
```

Output:



```
6
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

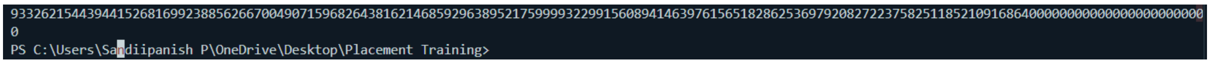
Time Complexity:O(n)

5.

Code:

```
import java.math.BigInteger;
public class Factorial {
    public static BigInteger factorial(int n) {
        BigInteger res = BigInteger.ONE;
        for (int i = 1; i <= n; i++) {
        res = res.multiply(BigInteger.valueOf(i));
        }
        return res;
    }
 public static void main(String[] args) {
    int n = 100;
    BigInteger fact = factorial(n);
    System.out.println(fact);
    }
}
```

Output:



```
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
0
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

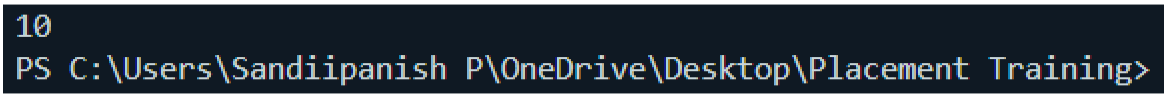
Time Complexity: O(n)

6.

Code:

```
public class trappingwater{
    public static int trap(int[] arr){
        int n =arr.length;
        if(n==0) return 0;
        nt lm[]=new int[n];
        int rm[]=new int[n];
        lm[0]=arr[0];
        for(int i=1;i<n;i++){
            lm[i]=Math.max(lm[i-1],arr[i]);
        }
        rm[n-1]=arr[n-1];
        for(int i=n-2;i>=0;i--){
        rm[i]=Math.max(rm[i+1],arr[i]);
        }
        int trapped=0;
        for(int i=0;i<n;i++){
        trapped+=Math.min(lm[i],rm[i])-arr[i];
        }
        return trapped;
    }
    public static void main(String[] args){
        int arr[]={3, 0, 1, 0, 4, 0, 2};
        System.out.println(trap(arr));
    }
}
```

Output:



```
10
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

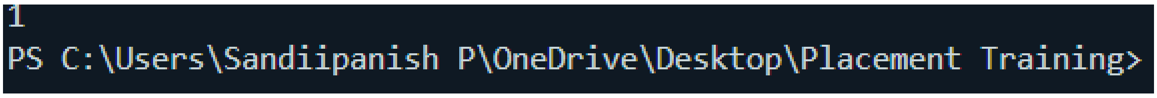
Time Complexity:O(n)

7.

Code:

```
import java.util.Arrays;
public class ChocolateDistribution {
    public static int findMinDifference(int[] packets, int students) {
        Arrays.sort(packets);
        int minDifference = Integer.MAX_VALUE;
        for (int i = 0; i <= packets.length - students; i++) {
            int difference = packets[i + students - 1] - packets[i];
            minDifference = Math.min(minDifference, difference);
        }
        return minDifference;
    }
    public static void main(String[] args) {
        int[] chocolates = {7, 5, 7, 8, 9, 12, 25, 57};
        int numStudents = 3;
        int result = findMinDifference(chocolates, numStudents);
        System.out.println(result);
    }
}
```

Output:



```
1
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

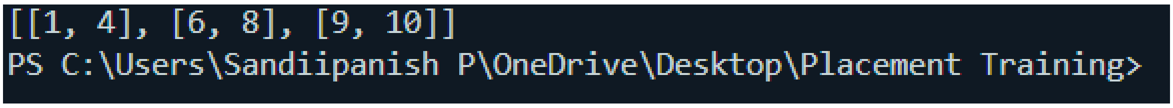
Time Complexity: O(nlogn)

8.

Code:

```
import java.util.*;

public class IntervalMerger {

    public static int[][] mergeIntervals(int[][] ranges) {
        if (ranges.length <= 1) {
            return ranges;
        }

        Arrays.sort(ranges, (a, b) -> Integer.compare(a[0], b[0]));

        List<int[]> merged = new ArrayList<>();
        merged.add(ranges[0]);
        for (int i = 1; i < ranges.length; i++) {
            int[] lastRange = merged.get(merged.size() - 1);
            int[] currentRange = ranges[i];

            if (lastRange[1] >= currentRange[0]) {
                lastRange[1] = Math.max(lastRange[1], currentRange[1]);
            } else {
                merged.add(currentRange);
            }
        }

        return merged.toArray(new int[merged.size()][]);
    }

    public static void main(String[] args) {
        int[][] intervals = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        System.out.println(Arrays.deepToString(mergeIntervals(intervals)));
    }
}
```

Output:



```
[[1, 4], [6, 8], [9, 10]]
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Complexity: O(nlogn)

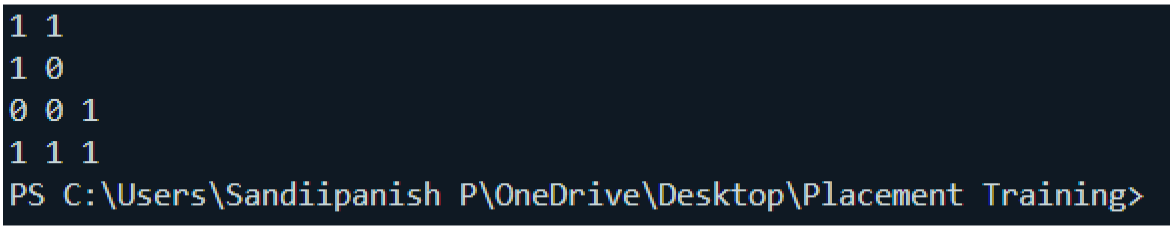
9.

Code:

public class MatrixModifier {

```
public static void setMatrixOnes(int[][] grid) {
    int numRows = grid.length;
    int numCols = grid[0].length;

    boolean[] rowMarkers = new boolean[numRows];
    boolean[] colMarkers = new boolean[numCols];

    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            if (grid[i][j] == 1) {
                rowMarkers[i] = true;
                colMarkers[j] = true;
            }
        }
    }

    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            if (rowMarkers[i] || colMarkers[j]) {
                grid[i][j] = 1;
            }
        }
    }
}

public static void displayMatrix(int[][] grid) {
    for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[0].length; j++) {
                System.out.print(grid[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] grid1 = {{1, 0}, {0, 0}};
        setMatrixOnes(grid1);
        displayMatrix(grid1);

        int[][] grid2 = {{0, 0, 0}, {0, 0, 1}};
        setMatrixOnes(grid2);
        displayMatrix(grid2);
    }
}
```

Output:



```
1 1
1 0
0 0 1
1 1 1
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Complexity: O(m*n)

10.

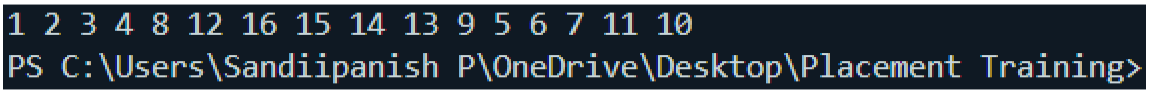
Code:

```
public class SpiralMatrix {


    public static void displaySpiral(int[][] grid) {
```

```
int topBound = 0, bottomBound = grid.length - 1;
int leftBound = 0, rightBound = grid[0].length - 1;

while (topBound <= bottomBound && leftBound <= rightBound) {
   for (int i = leftBound; i <= rightBound; i++) {
      System.out.print(grid[topBound][i] + " ");
   }
   topBound++;

   for (int i = topBound; i <= bottomBound; i++) {
      System.out.print(grid[i][rightBound] + " ");
   }
   rightBound--;

   if (topBound <= bottomBound) {
      for (int i = rightBound; i >= leftBound; i--) {
         System.out.print(grid[bottomBound][i] + " ");
      }
      bottomBound--;
   }

   if (leftBound <= rightBound) {
      for (int i = bottomBound; i >= topBound; i--) {
         System.out.print(grid[i][leftBound] + " ");
      }
      leftBound++;
   }
```

```
        }
    }

    public static void main(String[] args) {
        int[][] grid = {
            {1, 2, 3, 4},
            {5, 6, 7, 8},
            {9, 10, 11, 12},
            {13, 14, 15, 16}
        };
        displaySpiral(grid);
        System.out.println();
    }
}
```

Output:



```
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

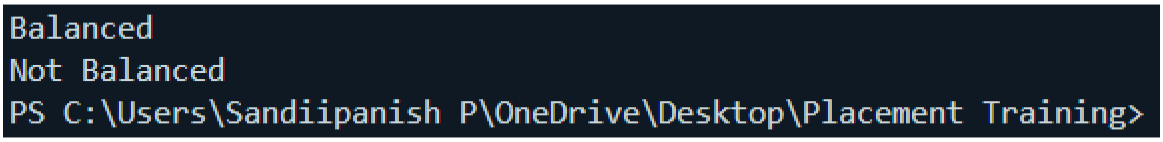
Time Complexity: O(m*n)

11.

Code:

```
public class ParenthesesBalance {

    public static String checkBalance(String expression) {
        int balanceCount = 0;
        for (int i = 0; i < expression.length(); i++) {
            char character = expression.charAt(i);

            if (character == '(') {
                balanceCount++;

            } else if (character == ')') {

                balanceCount--;

            }

            if (balanceCount < 0) {

                return "Not Balanced";

            }

        }

        return balanceCount == 0 ? "Balanced" : "Not Balanced";

    }


    public static void main(String[] args) {

        String test1 = "((()))()()";

        String test2 = "())((())";

        System.out.println(checkBalance(test1));

        System.out.println(checkBalance(test2));

    }

}
```

Output:



```
Balanced
Not Balanced
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Complexity:O(n)

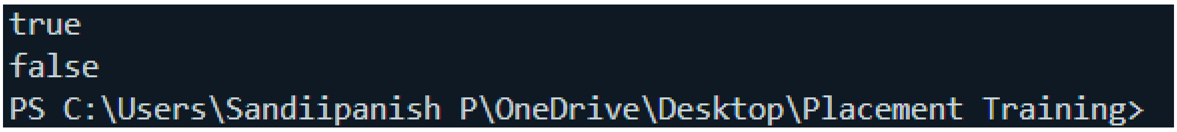
12.

Code:

```
import java.util.HashMap;


public class AnagramChecker {
```

```
public static boolean areAnagrams(String first, String second) {
    if (first.length() != second.length()) {
        return false;
    }
    HashMap<Character, Integer> characterCount = new HashMap<>();
    for (char character : first.toCharArray()) {
        characterCount.put(character, characterCount.getOrDefault(character, 0) + 1);
    }
    for (char character : second.toCharArray()) {
        if (!characterCount.containsKey(character)) {
            return false;
        }
        characterCount.put(character, characterCount.get(character) - 1);
        if (characterCount.get(character) == 0) {
            characterCount.remove(character);
        }
    }
    return characterCount.isEmpty();
}

public static void main(String[] args) {
    String word1 = "geeks";
    String word2 = "kseeg";
    System.out.println(areAnagrams(word1, word2)); // Output: true

    word1 = "allergy";
    word2 = "allergic";

    System.out.println(areAnagrams(word1, word2)); // Output: false

  }

}
```

Output:



```
true
false
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Comeplxity:O(n)

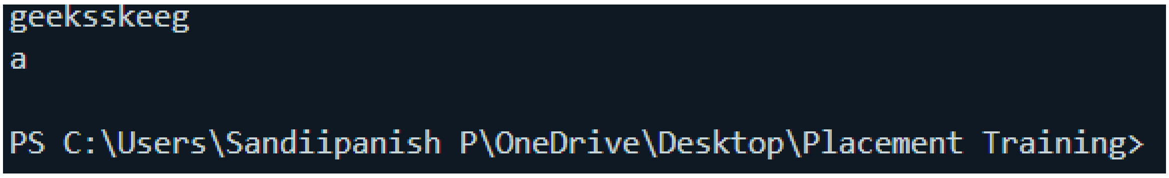
13.

Code:

```
public class LongestPalindromicSubstring {


  public static String findLongestPalindrome(String input) {

    if (input == null || input.length() < 1) {

      return "";

    }

    String result = "";


    for (int index = 0; index < input.length(); index++) {

      String oddLengthPalindrome = expandAroundCenter(input, index, index);

      if (oddLengthPalindrome.length() > result.length()) {

        result = oddLengthPalindrome;

      }

      String evenLengthPalindrome = expandAroundCenter(input, index, index + 1);

      if (evenLengthPalindrome.length() > result.length()) {

        result = evenLengthPalindrome;

      }

    }
```

```
        return result;
    }

    private static String expandAroundCenter(String input, int left, int right) {
        while (left >= 0 && right < input.length() && input.charAt(left) == input.charAt(right)) {
            left--;
            right++;
        }
        return input.substring(left + 1, right);
    }

    public static void main(String[] args) {
        String str1 = "forgeeksskeegfor";
        System.out.println(findLongestPalindrome(str1));

        String str2 = "abc";
        System.out.println(findLongestPalindrome(str2));

        String str3 = "";
        System.out.println(findLongestPalindrome(str3));
    }
}
```

Output:



```
geeksskeeg
a

PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Complexity: O(n^2)

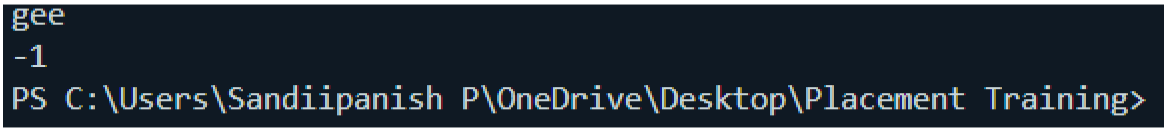
14.

Code:

```
import java.util.Arrays;


public class LongestCommonPrefixFinder {


  public static String findLCP(String[] strings) {

    if (strings == null || strings.length == 0) {

      return "-1";

    }

    Arrays.sort(strings);

    String firstString = strings[0];

    String lastString = strings[strings.length - 1];

    int index = 0;

    while (index < firstString.length() && index < lastString.length() && firstString.charAt(index) ==
lastString.charAt(index)) {

      index++;

    }

    String commonPrefix = firstString.substring(0, index);

    return commonPrefix.isEmpty() ? "-1" : commonPrefix;

  }


  public static void main(String[] args) {

    String[] input1 = {"geeksforgeeks", "geeks", "geek", "geezer"};

    System.out.println(findLCP(input1));


    String[] input2 = {"hello", "world"};
```

```
        System.out.println(findLCP(input2));
    }
}
```

Output:



```
gee
-1
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

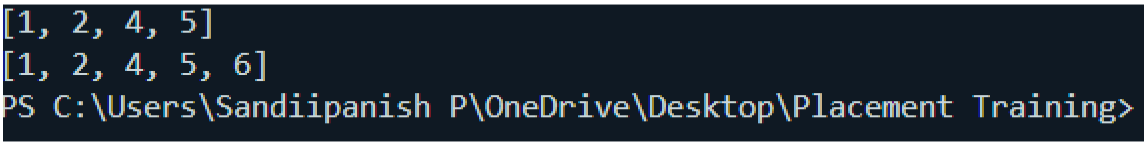
Time Complexity: O(n log n)

15.

Code:

```
import java.util.Stack;


public class RemoveMiddleElement {


    public static void removeMiddle(Stack<Integer> stack, int totalSize, int currentIndex) {
        if (stack.isEmpty() || currentIndex == totalSize / 2) {
            stack.pop();
            return;
        }
        int temp = stack.pop();
        removeMiddle(stack, totalSize, currentIndex + 1);
        stack.push(temp);
    }


    public static void deleteMiddle(Stack<Integer> stack) {
        int totalSize = stack.size();
        if (totalSize == 0) {
            return;
        }
        removeMiddle(stack, totalSize, 0);
    }

    public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<>();
        stack1.push(1);
        stack1.push(2);
        stack1.push(3);
        stack1.push(4);
        stack1.push(5);
        deleteMiddle(stack1);
        System.out.println(stack1);

        Stack<Integer> stack2 = new Stack<>();
        stack2.push(1);
        stack2.push(2);
        stack2.push(3);
        stack2.push(4);
        stack2.push(5);
        stack2.push(6);
        deleteMiddle(stack2);
        System.out.println(stack2);
    }
}
```

Output:



```
[1, 2, 4, 5]
[1, 2, 4, 5, 6]
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Complexity:O(n)

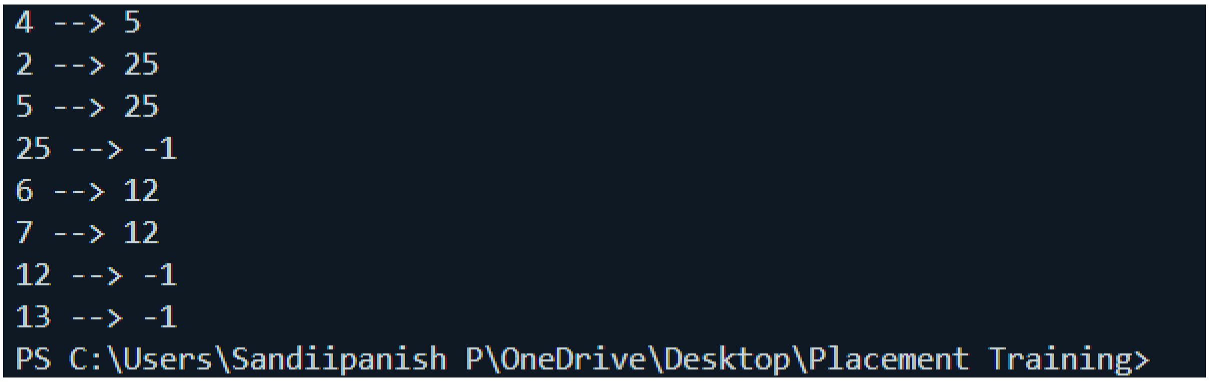
16.

Code:

```
import java.util.Stack;


public class NextGreaterElement {


  public static void findNextGreater(int[] array) {

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

    int length = array.length;


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

      while (!stack.isEmpty() && array[stack.peek()] < array[i]) {

        int index = stack.pop();

        System.out.println(array[index] + " --> " + array[i]);

      }

      stack.push(i);

    }


    while (!stack.isEmpty()) {

      int index = stack.pop();

      System.out.println(array[index] + " --> -1");

    }

  }
```

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

        int[] array1 = {4, 5, 2, 25};

        findNextGreater(array1);



        int[] array2 = {13, 7, 6, 12};

        findNextGreater(array2);

    }

}
```

Output:



```
4 --> 5
2 --> 25
5 --> 25
25 --> -1
6 --> 12
7 --> 12
12 --> -1
13 --> -1
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

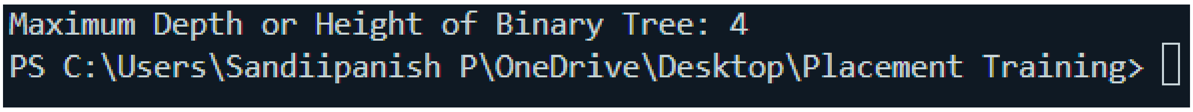
Time Complexity:O(n)

17.

Code:

```
import java.util.*;



class TreeNode {

    int value;

    TreeNode leftChild, rightChild;



    public TreeNode(int value) {

        this.value = value;
        leftChild = rightChild = null;
    }
}

public class BinaryTreeRightSideView {

    public static void rightSideView(TreeNode root) {
        if (root == null) {
            return;
        }
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            for (int i = 1; i <= levelSize; i++) {
                TreeNode currentNode = queue.poll();

                if (i == levelSize) {
                    System.out.print(currentNode.value + " ");
                }

                if (currentNode.leftChild != null) {
                    queue.add(currentNode.leftChild);
                }
                if (currentNode.rightChild != null) {
                    queue.add(currentNode.rightChild);
                }
            }
        }
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.leftChild = new TreeNode(2);
        root.rightChild = new TreeNode(3);
        root.leftChild.leftChild = new TreeNode(4);
        root.leftChild.rightChild = new TreeNode(5);
        root.rightChild.rightChild = new TreeNode(6);
        root.leftChild.leftChild.leftChild = new TreeNode(7);

        System.out.print("Right View: ");
        rightSideView(root);
    }
}
```

Output:



```
Maximum Depth or Height of Binary Tree: 4
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

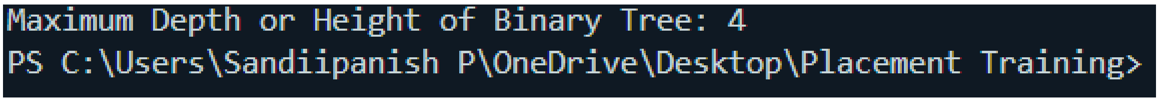
Time Compelxity:O(n)

18.

Code:

```
class TreeNode {
    int value;
    TreeNode leftChild, rightChild;
    public TreeNode(int value) {
        this.value = value;
        leftChild = rightChild = null;
    }
}

public class BinaryTreeDepth {
    public static int findMaxDepth(TreeNode node) {
        if (node == null) {
            return 0;
        }
        int leftDepth = findMaxDepth(node.leftChild);
        int rightDepth = findMaxDepth(node.rightChild);
        return Math.max(leftDepth, rightDepth) + 1;
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.leftChild = new TreeNode(2);
        root.rightChild = new TreeNode(3);
        root.leftChild.leftChild = new TreeNode(4);
        root.leftChild.rightChild = new TreeNode(5);
        root.leftChild.leftChild.leftChild = new TreeNode(6);
        System.out.println("Maximum Depth or Height of Binary Tree: " + findMaxDepth(root));
    }
}
```

Output:



```
Maximum Depth or Height of Binary Tree: 4
PS C:\Users\Sandiipanish P\OneDrive\Desktop\Placement Training>
```

Time Complexity:O(n)