DSA Coding Practice-1

1. Maximum Subarray Sum – Kadane"s Algorithm:

```
public class MaximumSubarraySum {
    public static int maxSubArraySum(int[] arr) {
        int maxSoFar = arr[0];
        int maxEndingHere = arr[0];

        for (int i = 1; i < arr.length; i++) {
            maxEndingHere = Math.max(arr[i], maxEndingHere + arr[i]);
            maxSoFar = Math.max(maxSoFar, maxEndingHere);
        }

        return maxSoFar;
    }

    public static void main(String[] args) {
        int[] arr1 = {2, 3, -8, 7, -1, 2, 3};
        int[] arr2 = {-2, -4};
        int[] arr3 = {5, 4, 1, 7, 8};

        System.out.println("Maximum subarray sum for arr1: " + maxSubArraySum(arr1));
        System.out.println("Maximum subarray sum for arr2: " + maxSubArraySum(arr2));
        System.out.println("Maximum subarray sum for arr3: " + maxSubArraySum(arr3));
 }
}
```

Output:

```
C:\demo>javac MaximumSubarraySum.java

C:\demo>java MaximumSubarraySum
Maximum subarray sum for arr1: 11
Maximum subarray sum for arr2: -2
Maximum subarray sum for arr3: 25

C:\demo>
```

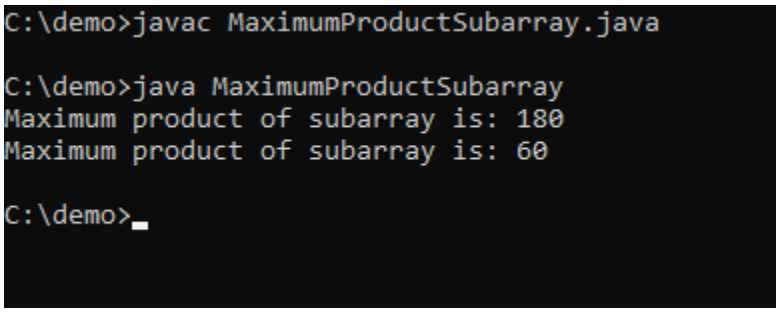
Time Complexity : O(n)

2. Maximum Product Subarray:

```
public class MaximumProductSubarray {
    public static int maxProduct(int[] arr) {
        if (arr.length == 0) {
            return 0;
        }
```

```
        int maxProd = arr[0];
        int minProd = arr[0];
        int result = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < 0) {
                int temp = maxProd;
                maxProd = minProd;
                minProd = temp;
            }
            maxProd = Math.max(arr[i], maxProd * arr[i]);
            minProd = Math.min(arr[i], minProd * arr[i]);
            result = Math.max(result, maxProd);
        }
        return result;
    }
    public static void main(String[] args) {
        int[] arr1 = {-2, 6, -3, -10, 0, 2};
        System.out.println("Maximum product of subarray is: " + maxProduct(arr1));
        int[] arr2 = {-1, -3, -10, 0, 60};
        System.out.println("Maximum product of subarray is: " + maxProduct(arr2));
    }
}
```

Output:



Time Complexity : O(n)

3. Search in a sorted and rotated Array:

```
public class SearchInSortedRotated {
    public static int search(int[] arr, int key) {
        int low = 0;
        int high = arr.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == key) {
                return mid;
```

```
            }
            if (arr[low] <= arr[mid]) {
              if (key >= arr[low] && key < arr[mid]) {
                 high = mid - 1;
              } else {
                 low = mid + 1;
              }
            } else {
               // Right side is sorted
               if (key > arr[mid] && key <= arr[high]) {
                  low = mid + 1;
               } else {
                  high = mid - 1;
               }
            }
         }
         return -1;
      }
      public static void main(String[] args) {
         int[] arr1 = {4, 5, 6, 7, 0, 1, 2};
         int key1 = 0;
         System.out.println("Index of " + key1 + ": " + search(arr1, key1));
         int[] arr2 = {4, 5, 6, 7, 0, 1, 2};
         int key2 = 3;
         System.out.println("Index of " + key2 + ": " + search(arr2, key2));
         int[] arr3 = {50, 10, 20, 30, 40};
         int key3 = 10;
         System.out.println("Index of " + key3 + ": " + search(arr3, key3));
      }
}
```

Output:

```
C:\demo>javac SearchInSortedRotated.java

C:\demo>java SearchInSortedRotated
Index of 0: 4
Index of 3: -1
Index of 10: 1

C:\demo>
```

Time Complexity : O(n)

4. Container with Most Water:

```
public class ContainerWithMostWater {
    public static int maxArea(int[] heights) {
        int left = 0;
        int right = heights.length - 1;
        int maxArea = 0;
        while (left < right) {
            int height = Math.min(heights[left], heights[right]);
            int width = right - left;
            int area = height * width;
            maxArea = Math.max(maxArea, area);
            if (heights[left] < heights[right]) {
                left++;
            } else {
                right--;
            }
        }
        return maxArea;
    }
    public static void main(String[] args) {
        int[] arr = {1, 5, 4, 3};
        System.out.println("Maximum area: " + maxArea(arr));
    }
}
```

Output:

```
C:\demo>javac ContainerWithMostWater.java

C:\demo>java ContainerWithMostWater
Maximum area: 6

C:\demo>_
```

Time Complexity: O(n^2)

5. Find the Factorial of a large number:

```
import java.math.BigInteger;
public class Factorial {
    public static void main(String[] args) {
        int number = 100;
        System.out.println("Factorial of " + number + " is: ");
        System.out.println(factorial(number));
    }
    public static BigInteger factorial(int n) {
        BigInteger result = BigInteger.ONE;
        for (int i = 2; i <= n; i++) {
```

```
            result = result.multiply(BigInteger.valueOf(i));
        }
        return result;
    }
}
```

Output:

```
C:\demo>java Factorial
Factorial of 100 is:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582
511852109168640000000000000000000000
```

Time Complexity: O(n)

6. Trapping Rainwater Problem:

```
public class TrappingRainwater {
    public static void main(String[] args) {
        int[] arr = {3, 0, 1, 0, 4, 0, 2};
        System.out.println(trappedRainwater(arr));
    }
    public static int trappedRainwater(int[] arr) {
        int n = arr.length;
        if (n == 0) return 0;
        int[] left_max = new int[n];
        int[] right_max = new int[n];
        left_max[0] = arr[0];
        for (int i = 1; i < n; i++) {
            left_max[i] = Math.max(arr[i], left_max[i - 1]);
        }
        right_max[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            right_max[i] = Math.max(arr[i], right_max[i + 1]);
        }
        int waterTrapped = 0;
        for (int i = 0; i < n; i++) {
            waterTrapped += Math.min(left_max[i], right_max[i]) - arr[i];
        }
        return waterTrapped;
    }
}
```

Output:

```
C:\demo>javac TrappingRainwater.java

C:\demo>java TrappingRainwater
10

C:\demo>
```

Time Complexity: O(n)

7. Chocolate Distribution Problem:

```
import java.util.Arrays;
public class ChocolateDistribution {
    public static void main(String[] args) {
        int[] arr = {7, 3, 2, 4, 9, 12, 56};
        int m = 3;
        System.out.println(chocolateDistribution(arr, m));
    }
    public static int chocolateDistribution(int[] arr, int m) {
        int n = arr.length;
        if (n < m) {
            return -1;
        }
        Arrays.sort(arr);
        int minDifference = Integer.MAX_VALUE;
        for (int i = 0; i <= n - m; i++) {
            int difference = arr[i + m - 1] - arr[i];
            minDifference = Math.min(minDifference, difference);
        }
        return minDifference;
    }
}
```

Output:

```
C:\demo>javac ChocolateDistribution.java

C:\demo>java ChocolateDistribution
2

C:\demo>
```

Time Complexity: O(n logn)

8. Merge Overlapping Intervals:

```
import java.util.Arrays;
import java.util.ArrayList;
```

```
public class MergeIntervals {
    public static void main(String[] args) {
        int[][] arr = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        System.out.println(Arrays.deepToString(mergeIntervals(arr)));
    }
    public static int[][] mergeIntervals(int[][] intervals) {
        if (intervals.length == 0) return new int[0][0];
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        ArrayList<int[]> merged = new ArrayList<>();
        merged.add(intervals[0]);
        for (int i = 1; i < intervals.length; i++) {
            int[] current = intervals[i];
            int[] last = merged.get(merged.size() - 1);
            if (current[0] <= last[1]) {
                last[1] = Math.max(last[1], current[1]);
            } else {
                merged.add(current);
            }
        }
        return merged.toArray(new int[merged.size()][]);
    }
}
```

Output:

```
C:\demo>javac MergeIntervals.java

C:\demo>java MergeIntervals
[[1, 4], [6, 8], [9, 10]]

C:\demo>
```

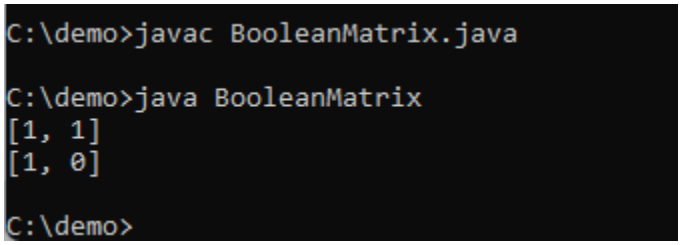
Time Complexity: O(n logn)

9. A Boolean Matrix Question:

```
public class BooleanMatrix {
    public static void main(String[] args) {
        int[][] mat = {{1, 0}, {0, 0}};
        modifyMatrix(mat);
        for (int[] row : mat) {
            System.out.println(java.util.Arrays.toString(row));
        }
    }
    public static void modifyMatrix(int[][] mat) {
        int m = mat.length, n = mat[0].length;
        boolean[] rowFlag = new boolean[m];
```

```
        boolean[] colFlag = new boolean[n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (mat[i][j] == 1) {
                    rowFlag[i] = true;
                    colFlag[j] = true;
                }
            }
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (rowFlag[i] || colFlag[j]) {
                    mat[i][j] = 1;
                }
            }
        }
    }
}
```

Output:



Time Complexity: O(mXn)

10. Print a given matrix in spiral form:

```
public class SpiralMatrix {
    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3, 4},{5, 6, 7, 8},{9, 10, 11, 12},{13, 14, 15, 16}};
        printSpiral(matrix);
    }
    public static void printSpiral(int[][] matrix) {
        int m = matrix.length, n = matrix[0].length;
        int top = 0, bottom = m - 1, left = 0, right = n - 1;
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) System.out.print(matrix[top][i] + " ");
            top++;
            for (int i = top; i <= bottom; i++) System.out.print(matrix[i][right] + " ");
            right--;
            if (top <= bottom) {
```

```
            for (int i = right; i >= left; i--) System.out.print(matrix[bottom][i] + " ");
            bottom--;
        }
        if (left <= right) {
            for (int i = bottom; i >= top; i--) System.out.print(matrix[i][left] + " ");
            left++;
        }
    }
  }
}
```

Output:

```
C:\demo>javac SpiralMatrix.java

C:\demo>java SpiralMatrix
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
C:\demo>
```

Time Complexity: O(mXn)

11. Check if given Parentheses expression is balanced or not:

```
public class BalancedParentheses {
   public static void main(String[] args) {
      String str = "((()))()()";
      System.out.println(isBalanced(str));

      String str2 = "())((())";
      System.out.println(isBalanced(str2));
   }
   public static String isBalanced(String str) {
      int count = 0;
      for (int i = 0; i < str.length(); i++) {
         if (str.charAt(i) == '(') count++;
         else count--;
         if (count < 0) return "Not Balanced";
      }
      return count == 0 ? "Balanced" : "Not Balanced";
   }
}
```

Output:

```
C:\demo>javac BalancedParentheses.java

C:\demo>java BalancedParentheses
Balanced
Not Balanced

C:\demo>
```

Time Complexity : O(n)

12. Check if two Strings are Anagrams of each other:

```
import java.util.Arrays;
public class AnagramCheck {
    public static void main(String[] args) {
        String s1 = "geeks", s2 = "kseeg";
        System.out.println(areAnagrams(s1, s2));

        String s1_2 = "allergy", s2_2 = "allergic";
        System.out.println(areAnagrams(s1_2, s2_2));

        String s1_3 = "g", s2_3 = "g";
        System.out.println(areAnagrams(s1_3, s2_3));
    }
    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) return false;
        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }
}
```

Output:

```
C:\demo>javac AnagramCheck.java

C:\demo>java AnagramCheck
true
false
true

C:\demo>
```

Time Complexity: O(n logn)

13. Longest Palindromic Substring:

```
public class LongestPalindromicSubstring {
    public static void main(String[] args) {
        String str = "forgeeksskeegfor";
        System.out.println(longestPalindrome(str));
    }
    public static String longestPalindrome(String str) {
        if (str == null || str.length() < 1) return "";
        int start = 0, end = 0;
        for (int i = 0; i < str.length(); i++) {
            int len1 = expandAroundCenter(str, i, i);
            int len2 = expandAroundCenter(str, i, i + 1);
            int len = Math.max(len1, len2);
            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
        }
        return str.substring(start, end + 1);
    }
    public static int expandAroundCenter(String str, int left, int right) {
        while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }
}
```

Output:

```
C:\demo>javac LongestPalindromicSubstring.java

C:\demo>java LongestPalindromicSubstring
geeksskeeg

C:\demo>
```

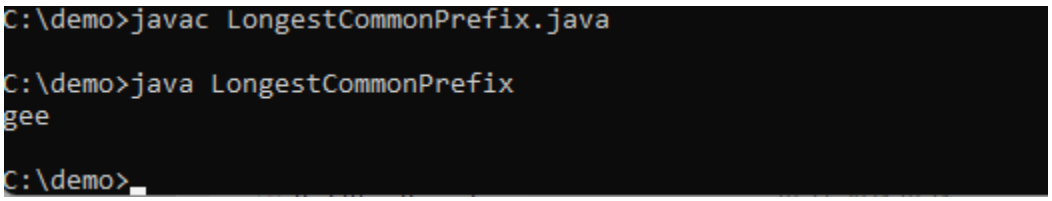
Time Complexity : O(n^2)

14. Longest Common Prefix using Sorting:

```
import java.util.Arrays;
public class LongestCommonPrefix {
    public static void main(String[] args) {
        String[] arr = {"geeksforgeeks", "geeks", "geek", "geezer"};
        System.out.println(longestCommonPrefix(arr));
    }
```

```
    public static String longestCommonPrefix(String[] arr) {
        if (arr == null || arr.length == 0) return "-1";
        Arrays.sort(arr);
        String first = arr[0], last = arr[arr.length - 1];
        int i = 0;
        while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {
            i++;
        }
        return i == 0 ? "-1" : first.substring(0, i);
    }
}
```

Output:



Time Complexity: O(n logn + m)

15. Delete middle element of a stack:

```
import java.util.Stack;
public class DeleteMiddleElement {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        deleteMiddle(stack, stack.size() / 2);
        System.out.println(stack);
    }
    public static void deleteMiddle(Stack<Integer> stack, int mid) {
        if (mid == 0) {
            stack.pop();
            return;
        }
        int temp = stack.pop();
        deleteMiddle(stack, mid - 1);
        stack.push(temp);
    }
}
```

Output:

```
C:\demo>javac DeleteMiddleElement.java

C:\demo>java DeleteMiddleElement
[1, 2, 4, 5]
```

Time Complexity : O(n)

16. Next Greater Element (NGE) for every element in given Array:

```
import java.util.Stack;
public class NextGreaterElement {
    public static void main(String[] args) {
        int[] arr = {4, 5, 2, 25};
        findNextGreaterElements(arr);
    }
    public static void findNextGreaterElements(int[] arr) {
        Stack<Integer> stack = new Stack<>();
        int[] result = new int[arr.length];
        for (int i = arr.length - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            result[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(arr[i]);
        }
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i] + " -> " + result[i]);
        }
    }
}
```

Output:

```
C:\demo>javac NextGreaterElement.java

C:\demo>java NextGreaterElement
4 -> 5
5 -> 25
2 -> 25
25 -> -1
```

Time Complexity: O(n)

17. Print Right View of a Binary Tree:

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int value) {
        val = value;
        left = right = null;
    }
}

public class RightViewBinaryTree {
    public static void rightView(TreeNode root) {
        if (root == null) return;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll();
                if (i == levelSize - 1) System.out.print(currentNode.val + " ");
                if (currentNode.left != null) queue.add(currentNode.left);
                if (currentNode.right != null) queue.add(currentNode.right);
            }
        }
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.right.right = new TreeNode(5);
        System.out.println("Right View of the Binary Tree:");
        rightView(root);
    }
}
```

Output:

```
C:\demo>javac RightViewBinaryTree.java

C:\demo>java RightViewBinaryTree
Right View of the Binary Tree:
1 3 5
```

Time Complexity: O(n)

18. Maximum Depth or Height of Binary Tree:

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTreeHeight {
    public static int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        } else {
            int leftDepth = maxDepth(root.left);
            int rightDepth = maxDepth(root.right);
            return Math.max(leftDepth, rightDepth) + 1;
        }
    }

    public static void main(String[] args) {
        TreeNode root1 = new TreeNode(12);
        root1.left = new TreeNode(8);
        root1.right = new TreeNode(18);
        root1.left.left = new TreeNode(5);
        root1.left.right = new TreeNode(11);
        System.out.println("The height of the binary tree (Example 1) is: " + maxDepth(root1));
        TreeNode root2 = new TreeNode(1);
        root2.left = new TreeNode(2);
        root2.right = new TreeNode(3);
        root2.left.left = new TreeNode(4);
        root2.right.left = new TreeNode(5);
        root2.right.left.left = new TreeNode(6);
        root2.right.left.right = new TreeNode(7);
        System.out.println("The height of the binary tree (Example 2) is: " + maxDepth(root2));
    }
}
```

Output:

```
C:\demo>javac BinaryTreeHeight.java

C:\demo>java BinaryTreeHeight
The height of the binary tree (Example 1) is: 3
The height of the binary tree (Example 2) is: 4
```

Time Complexity: O(n)