DEPARTMENT OF
COMPUTER SCIENCE &
Discover. Learn. Empower.

Assignment 5

Student Name: Kumar Devashish **UID:22BCS10248**

Branch: BE-CSE **Section/Group: FL_IOT-602 A**

Semester:6th **Date of Performance: 03-03-25**

Subject Name: Advanced Programming Lab-2 **Subject Code: 22CSP-351**

1️⃣ Find the Difference

Given two strings s and t, where t is s with one extra letter, find that letter.

```
class Solution1 {
    public char findTheDifference(String s, String t) {
        int charSum = 0;

        // Add ASCII values of all characters in t
        for (char c : t.toCharArray()) {
            charSum += c;
        }

        // Subtract ASCII values of all characters in s
        for (char c : s.toCharArray()) {
            charSum -= c;
        }

        // The remaining value is the extra character
        return (char) charSum;
    }
}
```

2️⃣ Largest Perimeter Triangle

Given an array of nums, find the **largest perimeter** of a triangle that can be formed.

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

```
class Solution2 {
    public int largestPerimeter(int[] nums) {
        Arrays.sort(nums); // Sort in ascending order

        // Start from the largest elements
        for (int i = nums.length - 1; i >= 2; i--) {
            if (nums[i - 2] + nums[i - 1] > nums[i]) {
                return nums[i - 2] + nums[i - 1] + nums[i]; // Valid triangle
            }
        }

        return 0; // No valid triangle found
    }
}
```

3️⃣ Third Maximum Number

Find the **third distinct maximum** number in the array. If it doesn't exist, return the **largest**.

```
import java.util.TreeSet;

class Solution3 {
    public int thirdMax(int[] nums) {
        TreeSet<Integer> maxSet = new TreeSet<>();

        for (int num : nums) {
            maxSet.add(num); // Maintain unique numbers
            if (maxSet.size() > 3) {
                maxSet.pollFirst(); // Remove the smallest to keep only 3
            }
        }

        return maxSet.size() == 3 ? maxSet.first() : maxSet.last(); // If < 3, return max
    }
}
```

4️⃣ Sort Characters By Frequency

Sort a string s based on the frequency of characters.

```
import java.util.*;

class Solution4 {
    public String frequencySort(String s) {
        Map<Character, Integer> freqMap = new HashMap<>();

        for (char c : s.toCharArray()) {
            freqMap.put(c, freqMap.getOrDefault(c, 0) + 1);
        }

        // Sort characters by frequency in descending order
        PriorityQueue<Character> maxHeap = new PriorityQueue<>(
            (a, b) -> freqMap.get(b) - freqMap.get(a)
        );
        maxHeap.addAll(freqMap.keySet());

        StringBuilder result = new StringBuilder();
        while (!maxHeap.isEmpty()) {
            char c = maxHeap.poll();
            result.append(String.valueOf(c).repeat(freqMap.get(c))); // Repeat char
        }

        return result.toString();
    }
}
```

5️⃣ Minimum Number of Arrows to Burst Balloons

Given points, find the **minimum arrows** needed to burst overlapping balloons.

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

```
class Solution5 {
    public int findMinArrowShots(int[][] points) {
        if (points.length == 0) return 0;

        Arrays.sort(points, (a, b) -> Integer.compare(a[1], b[1])); // Sort by end

        int arrows = 1;
        int end = points[0][1];

        for (int[] point : points) {
            if (point[0] > end) { // New non-overlapping balloon
                arrows++;
                end = point[1];
            }
        }

        return arrows;
    }
}
```

6️⃣ Boats to Save People

Given people and limit, return the **minimum number of boats** needed.

```
import java.util.Arrays;

class Solution6 {
    public int numRescueBoats(int[] people, int limit) {
        Arrays.sort(people); // Sort people by weight
        int left = 0, right = people.length - 1, boats = 0;

        while (left <= right) {
            if (people[left] + people[right] <= limit) {
                left++; // Pair the lightest person
            }
            right--; // Heaviest always boards
            boats++;
```

```
        }

        return boats;
    }
}
```

7️⃣ K Closest Points to Origin

Find k closest points (x, y) to (0, 0), using **Euclidean distance**.

```
import java.util.*;

class Solution7 {
    public int[][] kClosest(int[][] points, int k) {
        PriorityQueue<int[]> maxHeap = new PriorityQueue<>(
            (a, b) -> Integer.compare(
                (b[0] * b[0] + b[1] * b[1]),
                (a[0] * a[0] + a[1] * a[1])
            )
        );

        for (int[] point : points) {
            maxHeap.offer(point);
            if (maxHeap.size() > k) {
                maxHeap.poll(); // Remove farthest point
            }
        }

        int[][] result = new int[k][2];
        for (int i = 0; i < k; i++) {
            result[i] = maxHeap.poll();
        }

        return result;
    }
}
```

8️⃣ Reduce Array Size to Half

Remove the **minimum number of elements** to make the array half its original size.

```
import java.util.*;

class Solution8 {
    public int minSetSize(int[] arr) {
        Map<Integer, Integer> freqMap = new HashMap<>();
        for (int num : arr) {
            freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
        }

        List<Integer> freqList = new ArrayList<>(freqMap.values());
        Collections.sort(freqList, Collections.reverseOrder());

        int removed = 0, count = 0, halfSize = arr.length / 2;
        for (int freq : freqList) {
            removed += freq;
            count++;
            if (removed >= halfSize) return count;
        }

        return count;
    }
}
```