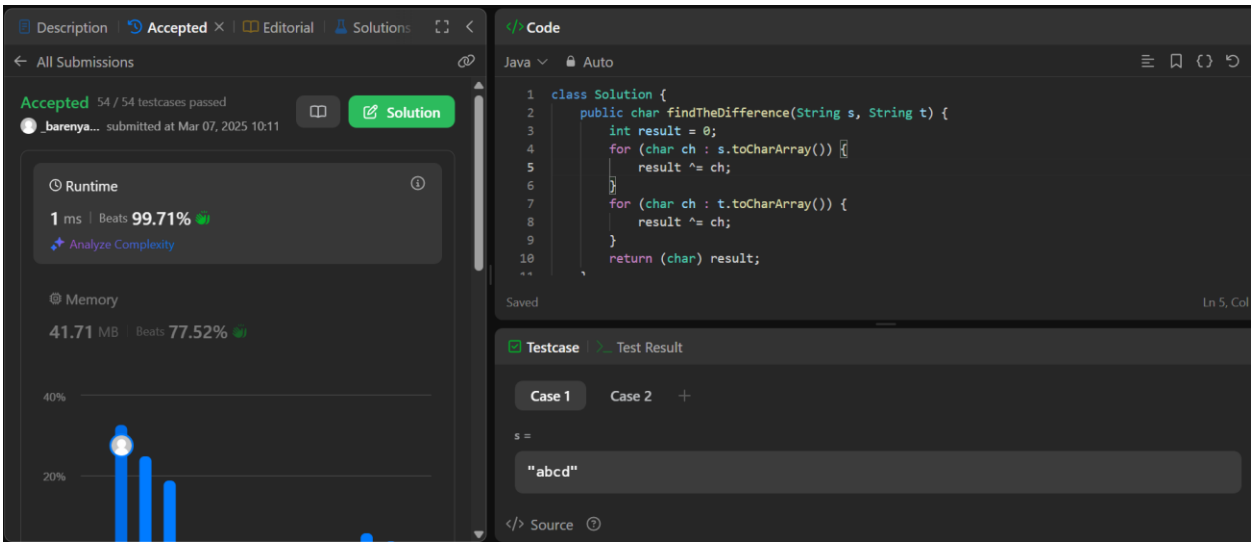
Experiment 5

Student Name: Barenya Behera **UID: 22BCS15121**
Branch: BE-CSE **Section/Group:FL_IOT-602/A**
Semester: 6th **Date of Performance: 4/03/25**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22CSP-351**

1. Implementation/Code:

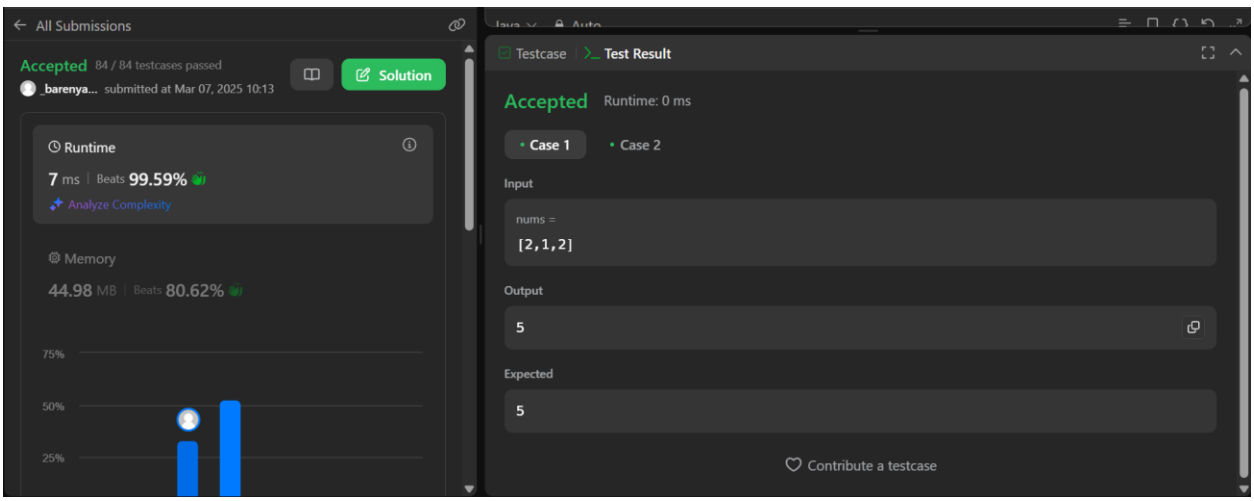
i. Find the difference

```
class Solution {
  public char findTheDifference(String s, String t) {
    int result = 0;
    for (char ch : s.toCharArray()) {
      result ^= ch;
    }
    for (char ch : t.toCharArray()) {
      result ^= ch;
    }
    return (char) result;
  }
}
```



ii. Largest Perimeter Traiangle

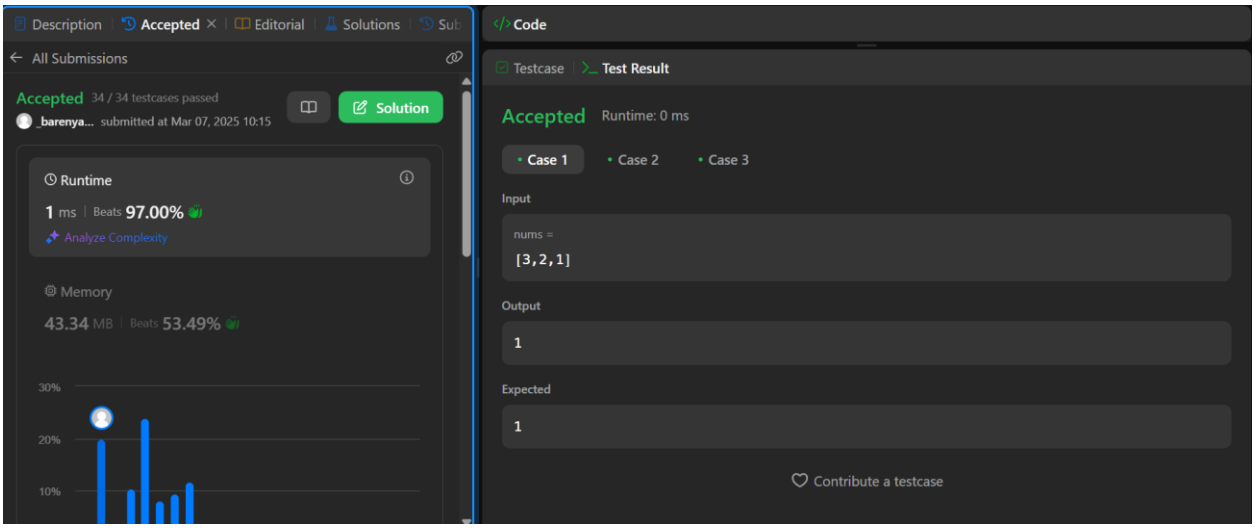
```
class Solution {
  public int largestPerimeter(int[] nums) {
    Arrays.sort(nums);
    for (int i = nums.length - 1; i >= 2; i--) {
      if (nums[i - 2] + nums[i - 1] > nums[i]) {
        return nums[i - 2] + nums[i - 1] + nums[i];
      }
    }
    return 0;
  }
}
```



iii. Third Maximum Number

```
class Solution {
  public int thirdMax(int[] nums) {
    long first = Long.MIN_VALUE, second = Long.MIN_VALUE, third =
Long.MIN_VALUE;
    for (int num : nums) {
      if (num == first || num == second || num == third) continue;
      if (num > first) {
        third = second;
        second = first;
```

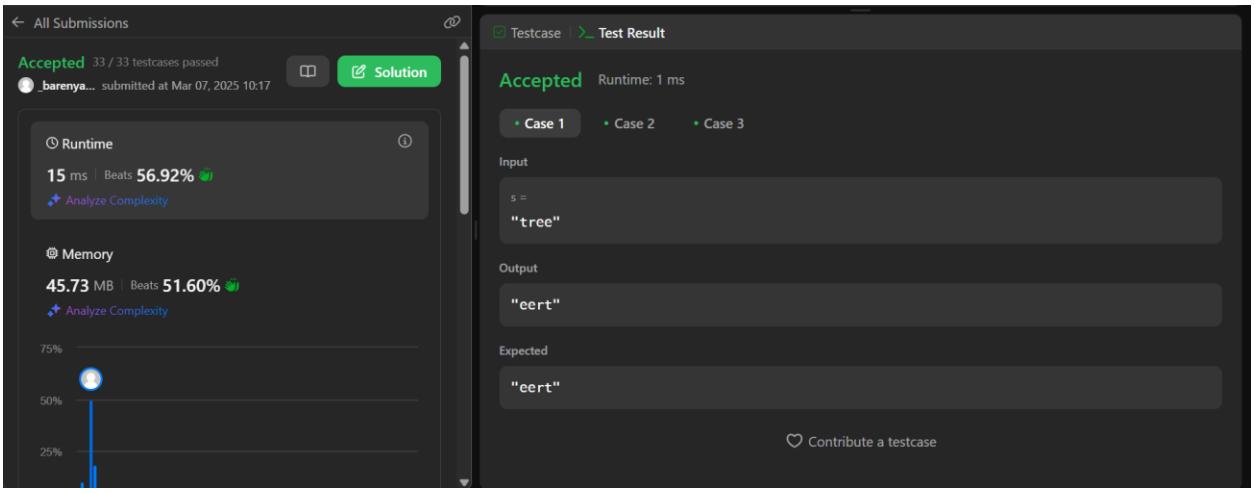
```
            first = num;
        } else if (num > second) {
            third = second;
            second = num;
        } else if (num > third) {
            third = num;
        }
    }
    return third == Long.MIN_VALUE ? (int) first : (int) third;
  }
}
```



iv. Sort Characters By Frequency

```
class Solution {
  public String frequencySort(String s) {
    Map<Character, Integer> freqMap = new HashMap<>();
    for (char c : s.toCharArray()) {
      freqMap.put(c, freqMap.getOrDefault(c, 0) + 1);
    }

    PriorityQueue<Character> maxHeap = new PriorityQueue<>((a, b) ->
freqMap.get(b) - freqMap.get(a));
```

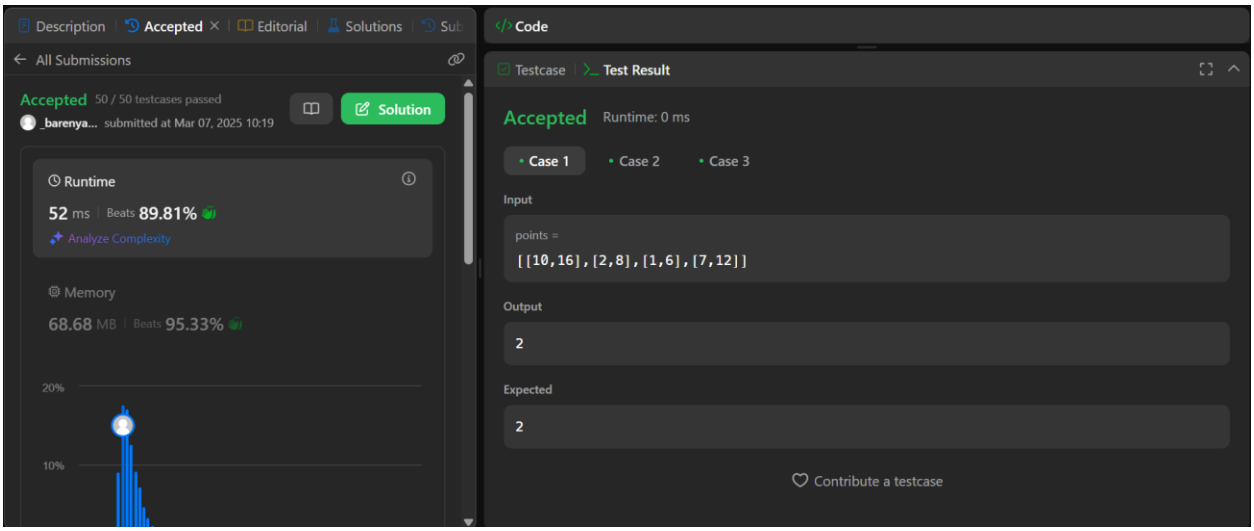
```
        maxHeap.addAll(freqMap.keySet());

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

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



v. Minimum Number of Arrows to Burst Ballons

```
class Solution {
    public int findMinArrowShots(int[][] points) {
        Arrays.sort(points, (a, b) -> Integer.compare(a[1], b[1]));
        int arrows = 1;
        int end = points[0][1];

        for (int i = 1; i < points.length; i++) {
            if (points[i][0] > end) {
                arrows++;
```

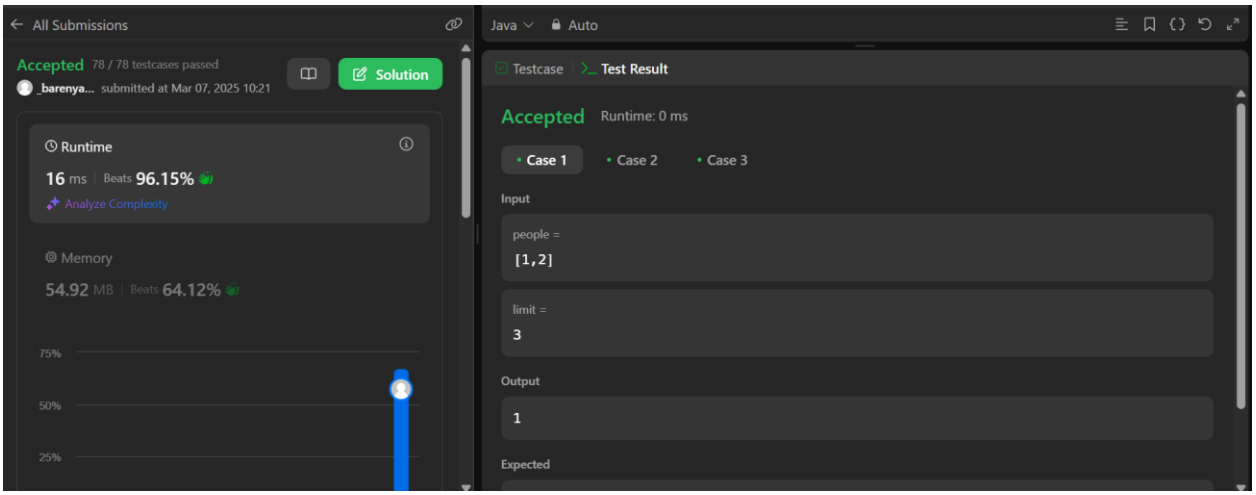
```
                end = points[i][1];
            }
        }

        return arrows;
    }
}
```



vi. Boats to Save People

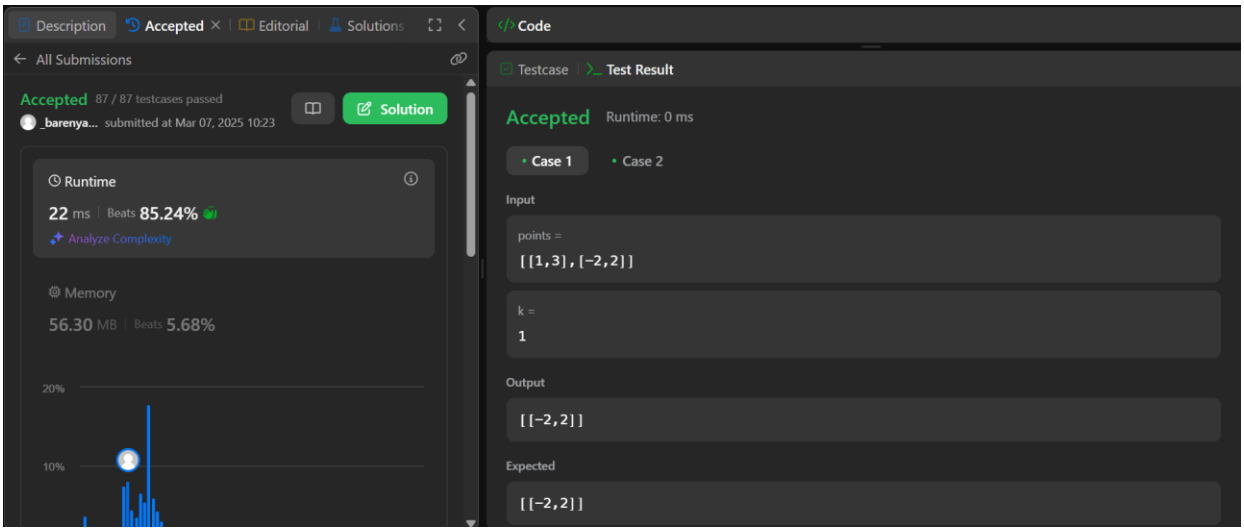
```
class Solution {
    public int numRescueBoats(int[] people, int limit) {
        Arrays.sort(people);
        int left = 0, right = people.length - 1;
        int boats = 0;

        while (left <= right) {
            if (people[left] + people[right] <= limit) {
                left++;
            }
            right--;
            boats++;
        }
```

```
        return boats;
    }
}
```



vii. K Closest Points to Origin

```
class Solution {
    public int[][] kClosest(int[][] points, int k) {
        Arrays.sort(points, (a, b) -> Integer.compare(a[0] * a[0] + a[1] * a[1],
                                                      b[0] * b[0] + b[1] * b[1]));
        return Arrays.copyOfRange(points, 0, k);
    }
}
```



viii. Reduce Array Size to the Half

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

    PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
    maxHeap.addAll(freqMap.values());

    int halfSize = arr.length / 2, removed = 0, count = 0;
    while (removed < halfSize) {
      removed += maxHeap.poll();
      count++;
    }

    return count;
  }
}
```