DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 5

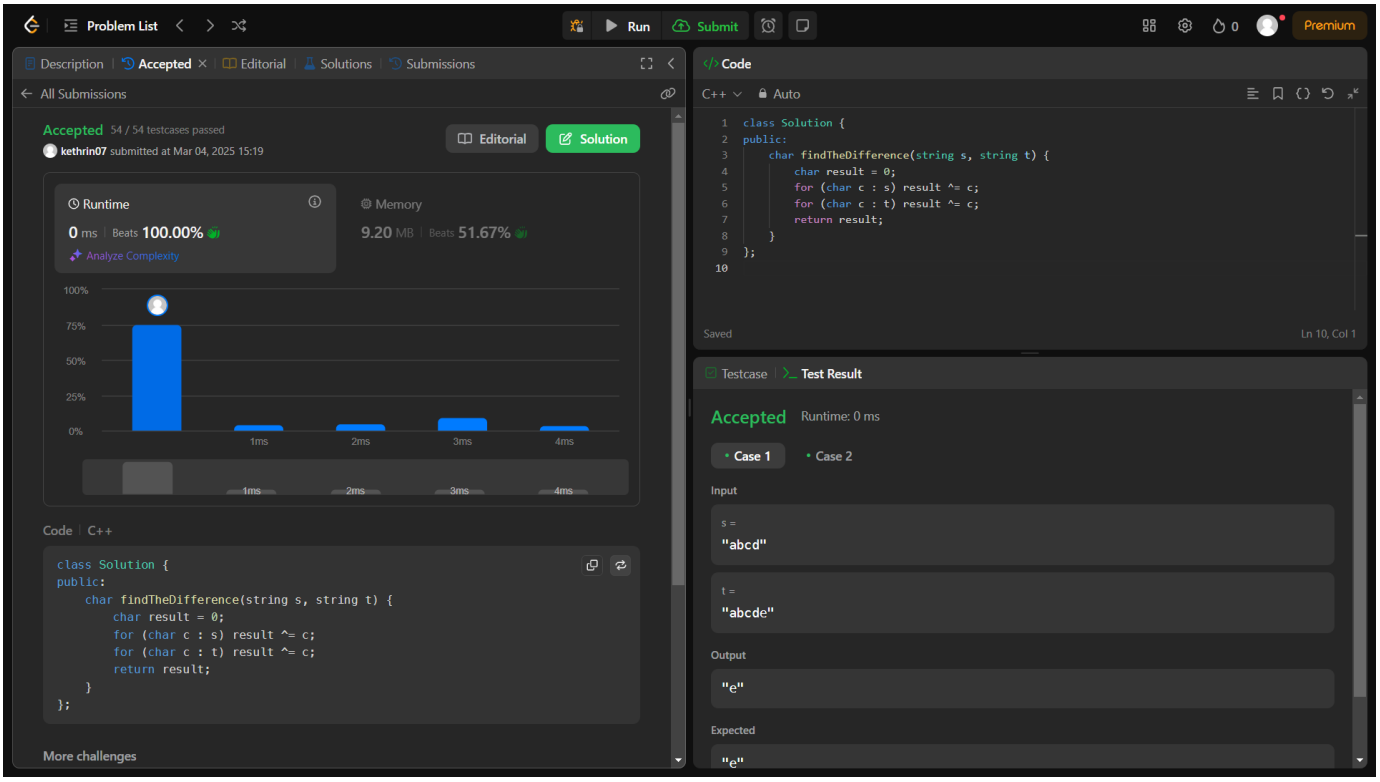
Student Name: Kethrin Naharwal **UID: 22BCS12653**
Branch: BE-CSE **Section: 22BCS_FL_IOT-603/A**
Semester: 6th **Date of Performance: 04/03/25**
Subject Name: AP LAB-II **Subject Code: 22CSP-351**

- **389.Find the diffrence**

CODE:

```
class Solution {
public:
    char findTheDifference(string s, string t) {
        char result = 0;
        for (char c : s) result ^= c;
        for (char c : t) result ^= c;
        return result;
    }
};
```

OUPUT:

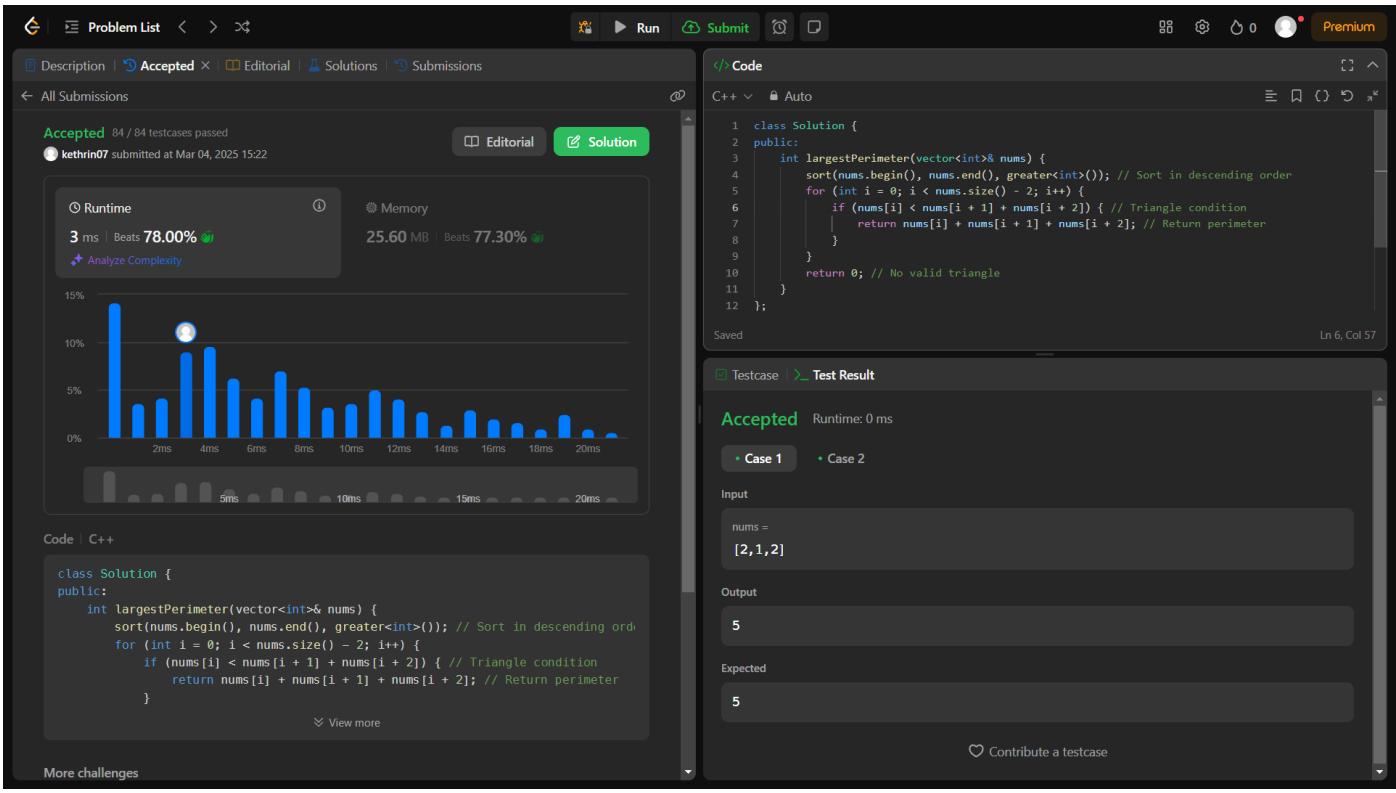


- **976.Largest Perimeter Triangle**

CODE:

```
class Solution {
public:
    int largestPerimeter(vector<int>& nums) {
        sort(nums.begin(), nums.end(), greater<int>()); // Sort in descending order
        for (int i = 0; i < nums.size() - 2; i++) {
            if (nums[i] < nums[i + 1] + nums[i + 2]) { // Triangle condition
                return nums[i] + nums[i + 1] + nums[i + 2]; // Return perimeter
            }
        }
        return 0; // No valid triangle
    }
};
```

OUTPUT:



- **414.Third Maximum Number**

CODE:

```
class Solution {
public:
    int thirdMax(vector<int>& nums) {
```

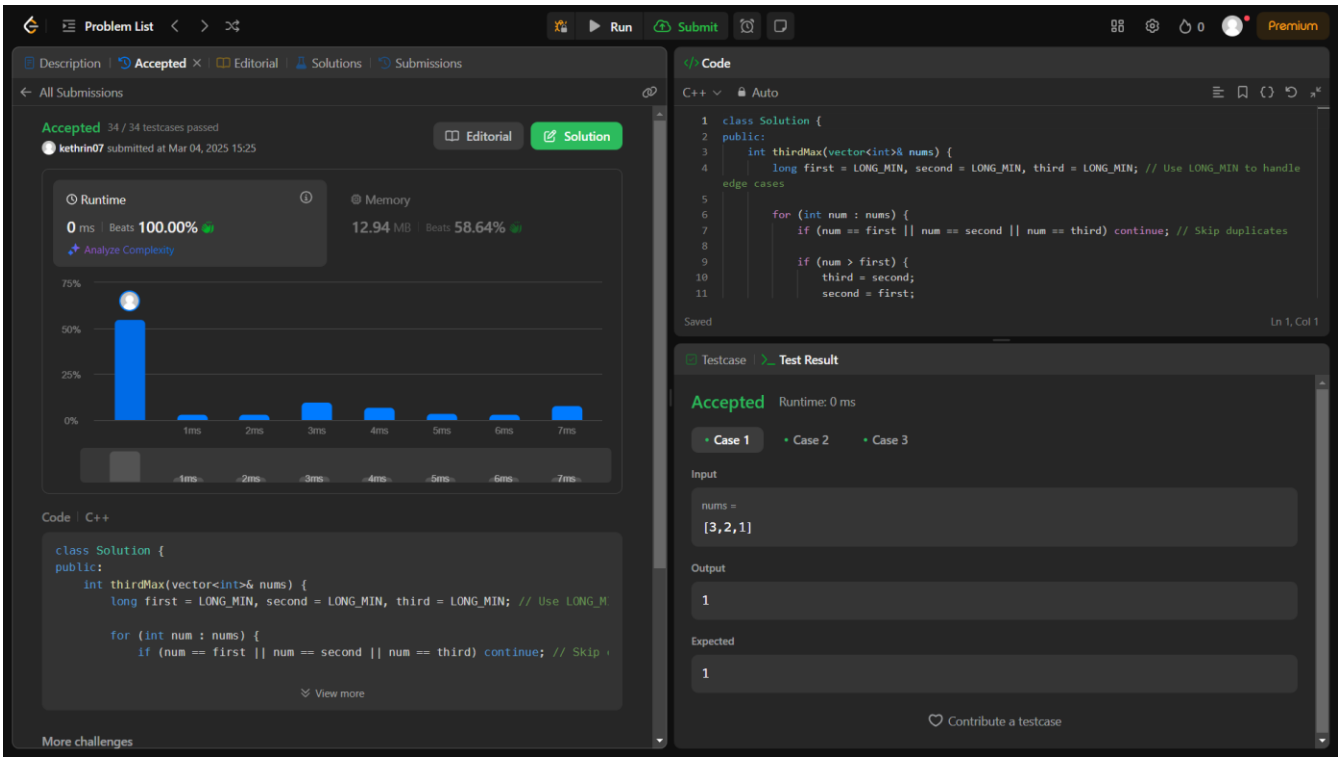
```
        long first = LONG_MIN, second = LONG_MIN, third = LONG_MIN; // Use
    LONG_MIN to handle edge cases

        for (int num : nums) {
            if (num == first || num == second || num == third) continue; // Skip duplicates

            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) ? first : third; // Return third max if exists,
    otherwise max
    }
};
```

OUTPUT:



- **451.Sort Characters By Frequency**

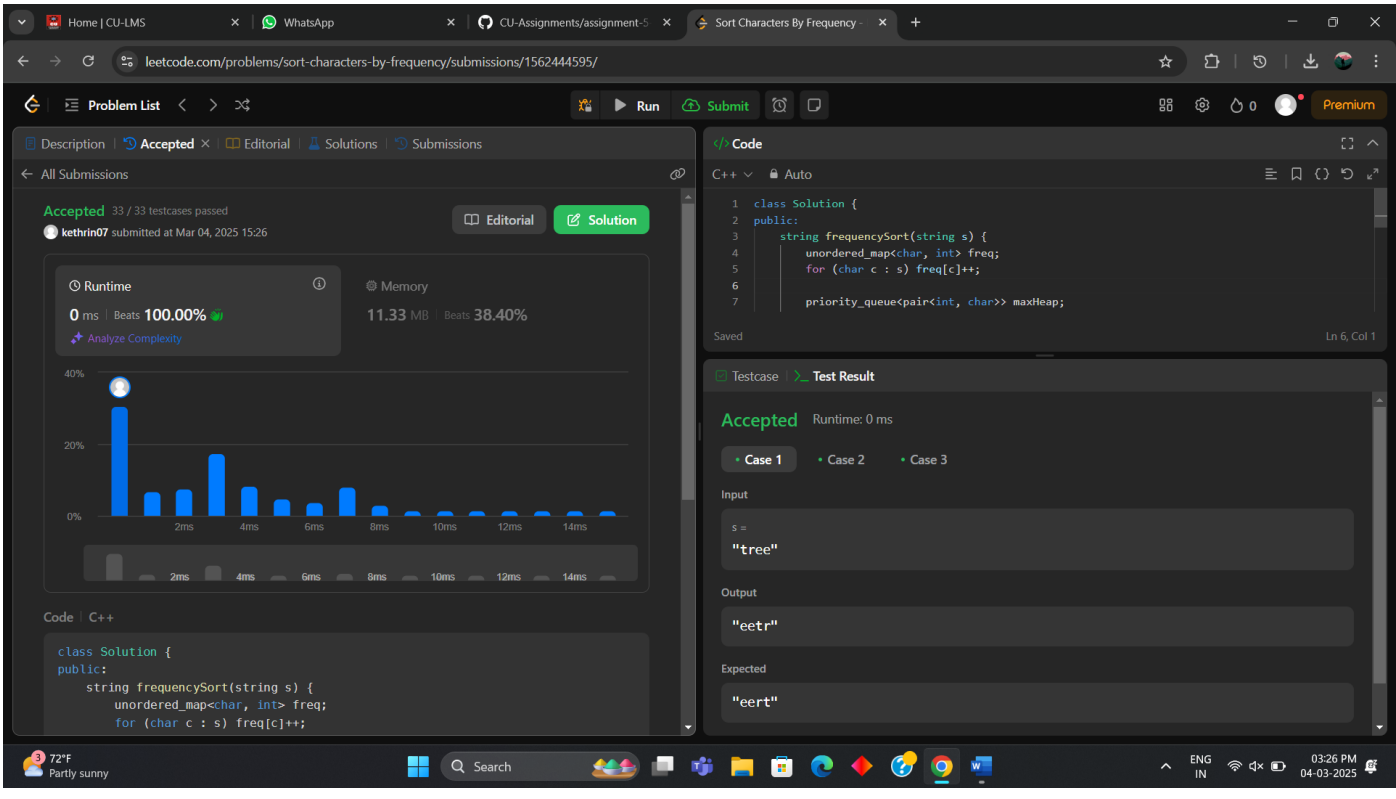
CODE:

```
class Solution {
public:
    string frequencySort(string s) {
        unordered_map<char, int> freq;
        for (char c : s) freq[c]++;

        priority_queue<pair<int, char>> maxHeap;
        for (auto [c, f] : freq) maxHeap.push({f, c});

        string result;
        while (!maxHeap.empty()) {
            auto [f, c] = maxHeap.top();
            maxHeap.pop();
            result.append(f, c);
        }
        return result;
    }
};
```

OUTPUT:



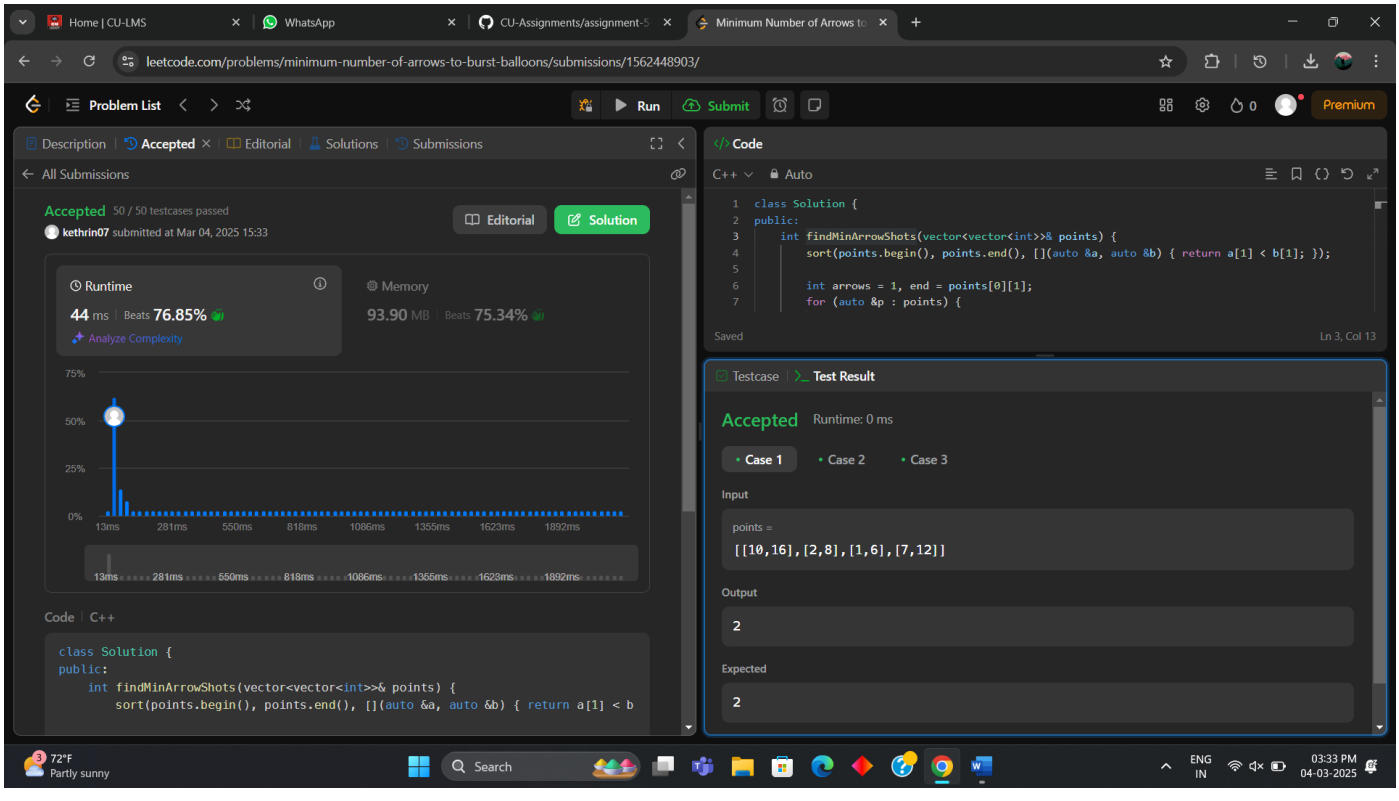
- **452.Minimum Number of Arrows to Burst Balloons**

CODE:

```
class Solution {
public:
    int findMinArrowShots(vector<vector<int>>& points) {
        sort(points.begin(), points.end(), [](auto &a, auto &b) { return a[1] < b[1]; });

        int arrows = 1, end = points[0][1];
        for (auto &p : points) {
            if (p[0] > end) {
                arrows++;
                end = p[1];
            }
        }
        return arrows;
    }
};
```

OUTPUT:



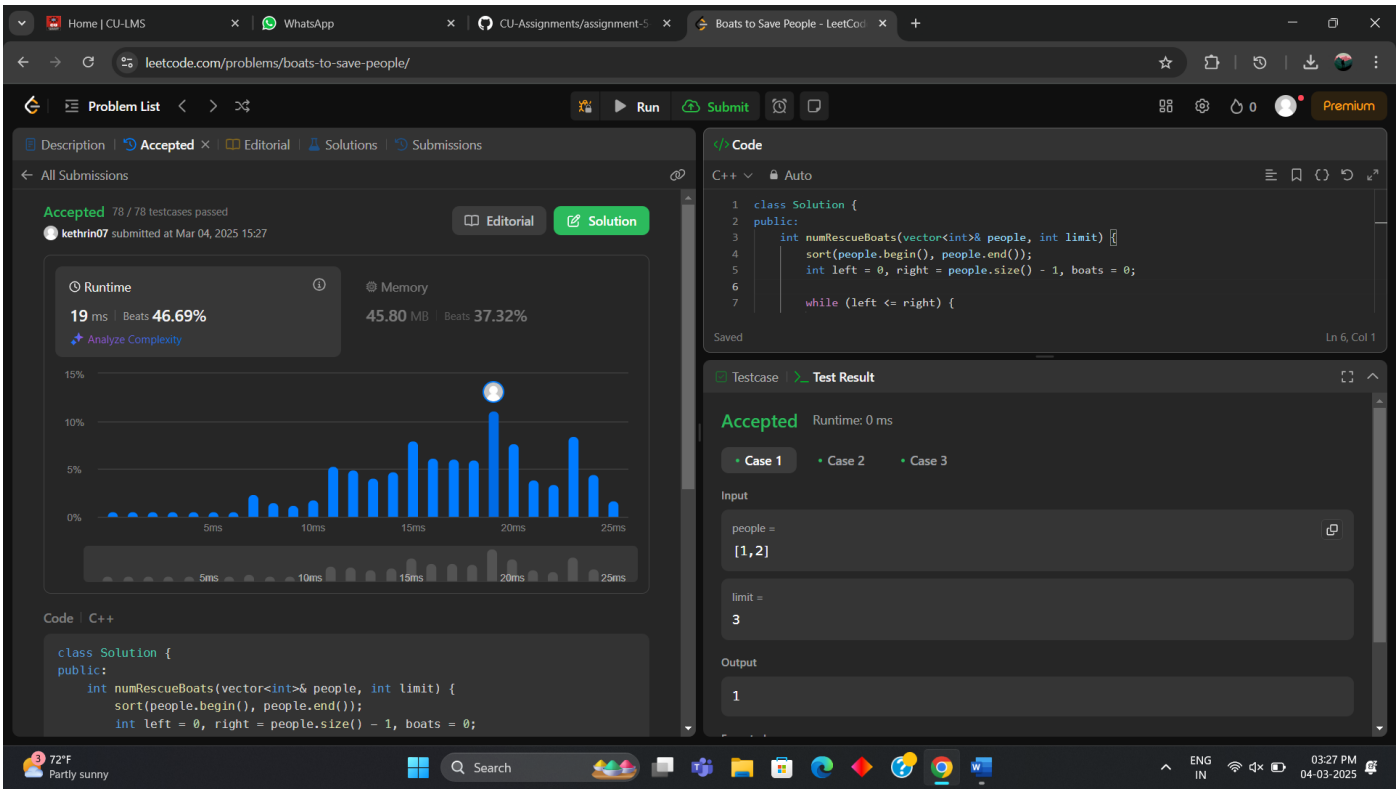
- **881.Boats to Save People**

CODE:

```
class Solution {
public:
    int numRescueBoats(vector<int>& people, int limit) {
        sort(people.begin(), people.end());
        int left = 0, right = people.size() - 1, boats = 0;

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

OUTPUT:



- **973.K Closest Points to Origin**

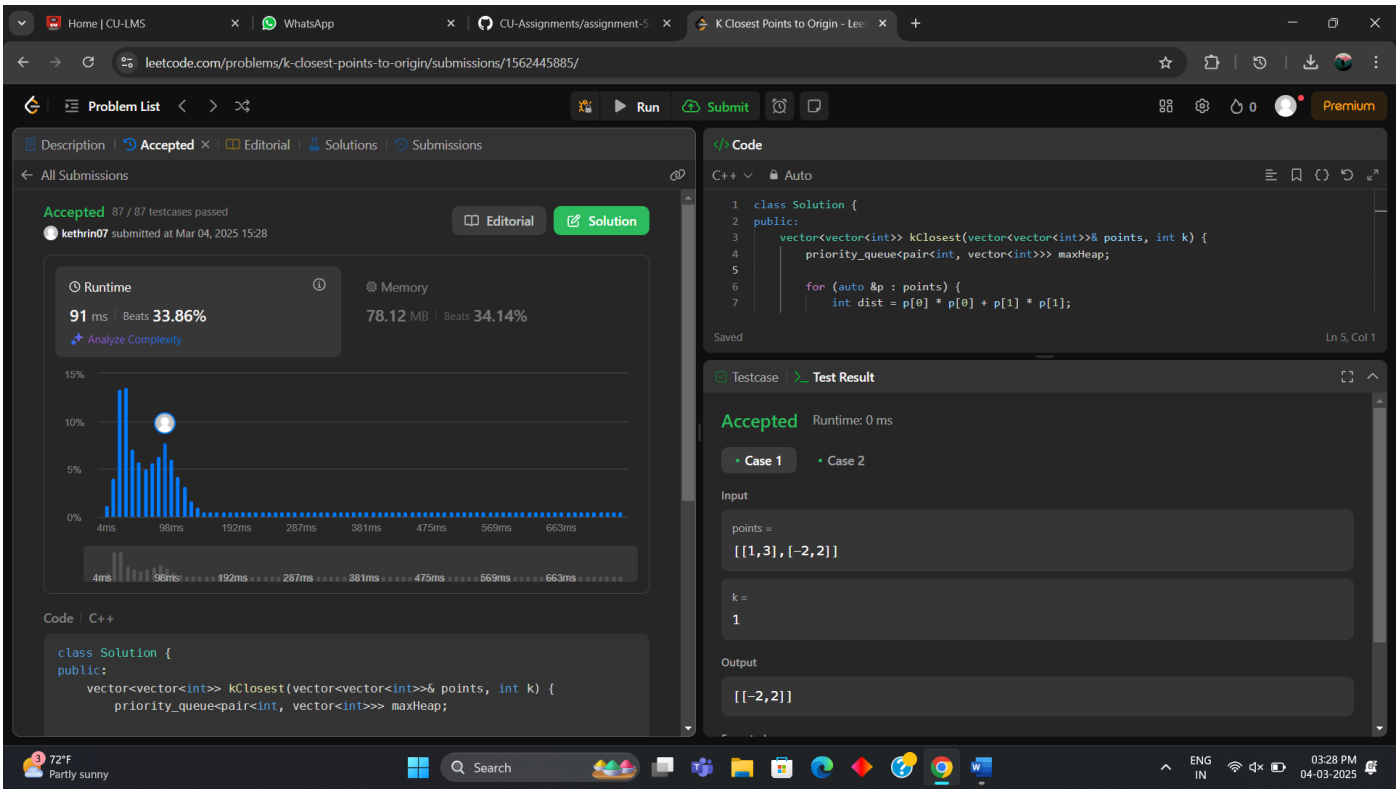
CODE:

```
class Solution {
public:
    vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
        priority_queue<pair<int, vector<int>>> maxHeap;

        for (auto &p : points) {
            int dist = p[0] * p[0] + p[1] * p[1];
            maxHeap.push({dist, p});
            if (maxHeap.size() > k) maxHeap.pop();
        }

        vector<vector<int>> result;
        while (!maxHeap.empty()) {
            result.push_back(maxHeap.top().second);
            maxHeap.pop();
        }
        return result;
    }
};
```

OUTPUT:



- **1338.[Reduce Array Size to The Half]**

CODE:

```
class Solution {
public:
    int minSetSize(vector<int>& arr) {
        unordered_map<int, int> freq;
        for (int num : arr) freq[num]++;

        vector<int> counts;
        for (auto [num, f] : freq) counts.push_back(f);

        sort(counts.rbegin(), counts.rend()); // Sort in descending order

        int removed = 0, setSize = 0, half = arr.size() / 2;
        for (int f : counts) {
            removed += f;
            setSize++;
            if (removed >= half) return setSize;
        }
        return setSize;
    }
};
```

OUTPUT: