ADVANCED PROGRAMMING LAB-2 ASSIGNMENT 5

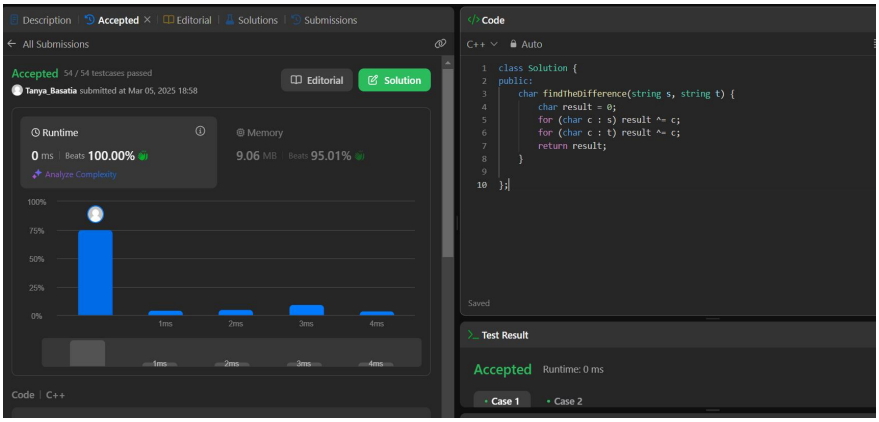
Name- Tanya

UID- 22BCS15446

Section-22BCS_IOT_605-B

1. Find the difference

```
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;
    }
```

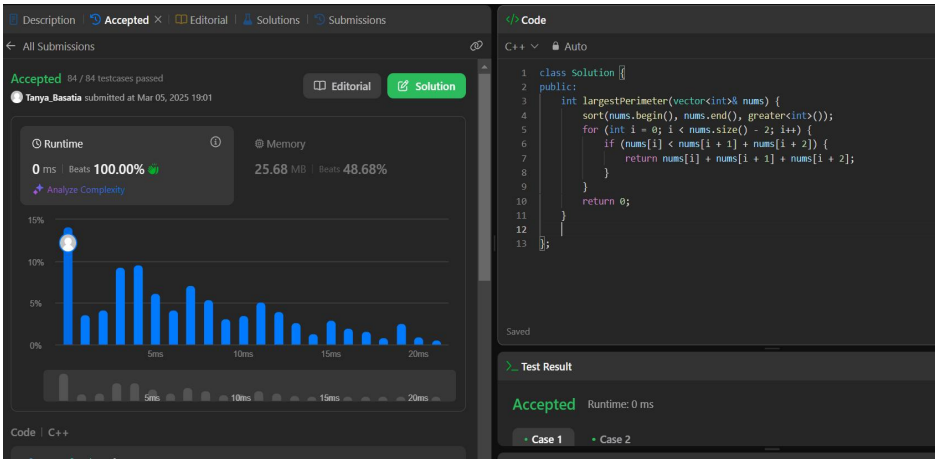


};

2. Largest Perimeter Triangle

```
class Solution {
public:
    int largestPerimeter(vector<int>& nums) {
        sort(nums.begin(), nums.end(), greater<int>());
        for (int i = 0; i < nums.size() - 2; i++) {
            if (nums[i] < nums[i + 1] + nums[i + 2]) {
                return nums[i] + nums[i + 1] + nums[i + 2];
            }
        }
        return 0;
    }

};
```

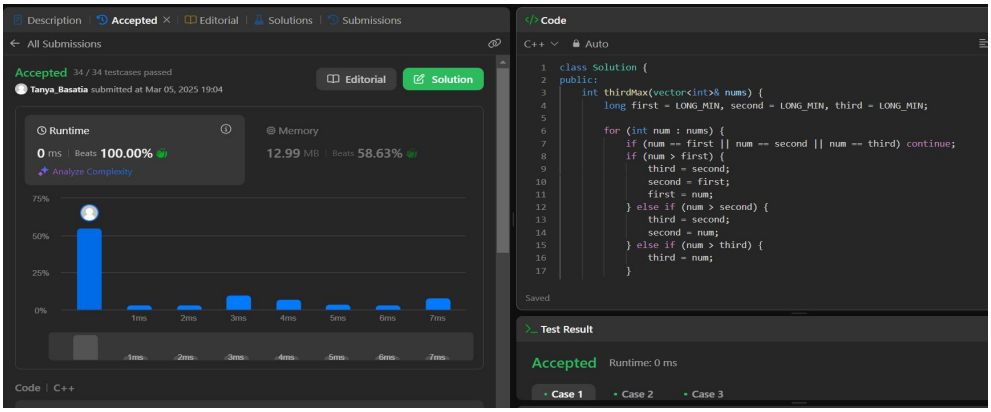


3. Third Maximum Number

```
class Solution {
public:
  int thirdMax(vector<int>& nums) {
    long first = LONG_MIN, second = LONG_MIN, third = LONG_MIN;

    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 ? first : third;
  }
};
```



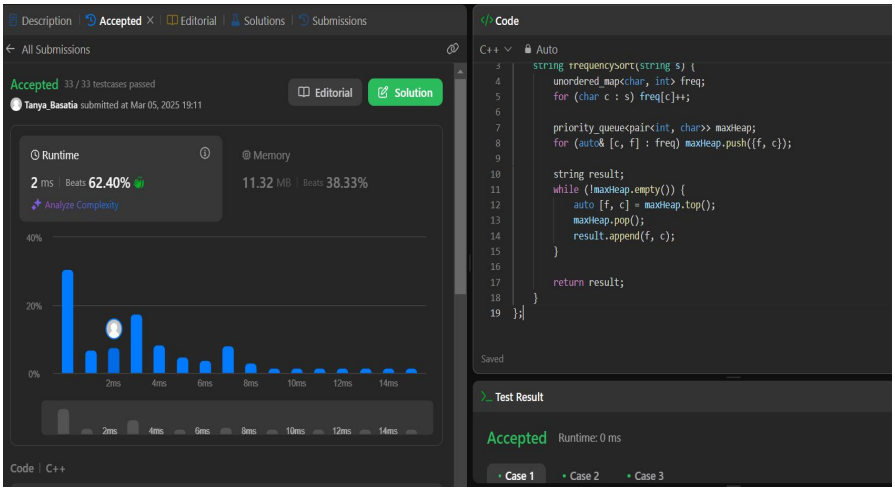
4. Sort Characters By Frequency

```
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;
  }
};
```



5. Minimum No. Of arrows to Burst Balloon

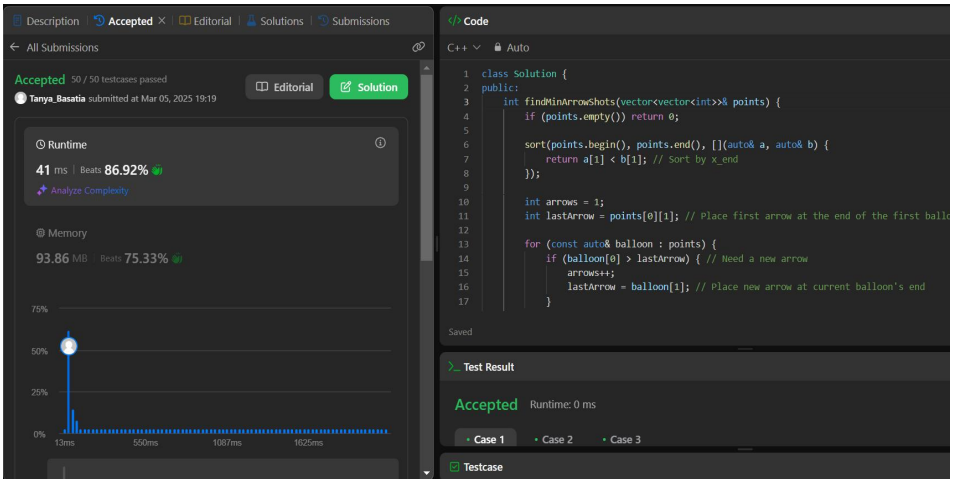
```
class Solution {
public:
  int findMinArrowShots(vector<vector<int>>& points) {
    if (points.empty()) return 0;

    sort(points.begin(), points.end(), [](auto& a, auto& b) {
      return a[1] < b[1]; // Sort by x_end
    });

    int arrows = 1;
    int lastArrow = points[0][1]; // Place first arrow at the end of the first balloon
```

```
        for (const auto& balloon : points) {
            if (balloon[0] > lastArrow) { // Need a new arrow
                arrows++;
                lastArrow = balloon[1]; // Place new arrow at current balloon's end
            }
        }

        return arrows;
    }
};
```

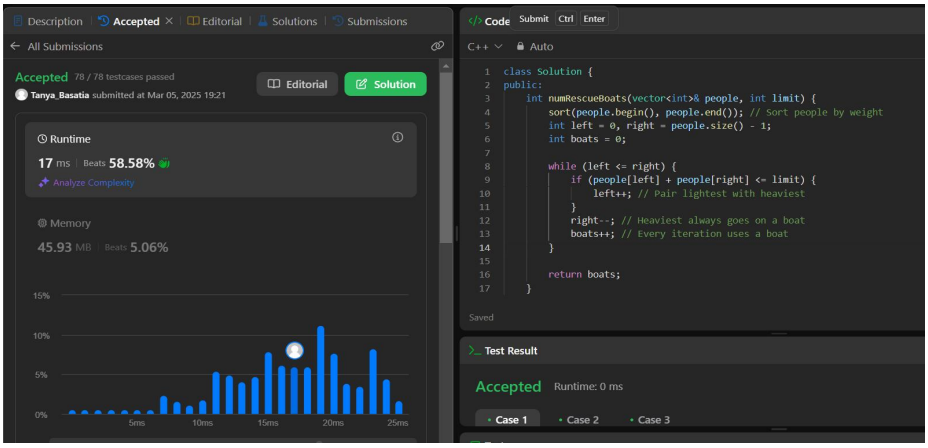


6. Boats To Save People

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

        while (left <= right) {
            if (people[left] + people[right] <= limit) {
                left++; // Pair lightest with heaviest
            }
            right--; // Heaviest always goes on a boat
            boats++; // Every iteration uses a boat
        }

        return boats;
    }
};
```



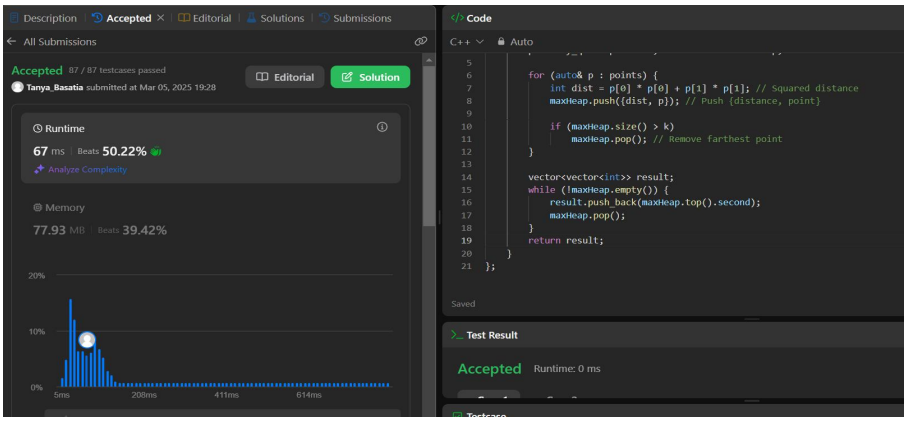
7. K Closest Points To Origin

```
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]; // Squared distance
      maxHeap.push({dist, p}); // Push {distance, point}

      if (maxHeap.size() > k)
        maxHeap.pop(); // Remove farthest point
    }

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



8. Reduce Array size to the Half

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

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

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

        int removed = 0, totalRemoved = 0, halfSize = arr.size() / 2;
        for (int count : counts) {
            removed += count;
            totalRemoved++;
            if (removed >= halfSize) break;
        }
        return totalRemoved;
    }
};
```