



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 5

Student Name: Nitesh Bamber

Branch: BE-CSE

Semester: 6th

Subject Name: Advanced Programming - II

UID: 22BCS13791

Section/Group: 602/A

Date of Performance: 04-03-25

Subject Code: 22CSP-351

Problems Solved –

389.[Find the difference](#)

976.[Largest Perimeter Triangle](#)

414.[Third Maximum Number](#)

451.[Sort Characters By Frequency](#)

452.[Minimum Number of Arrows to Burst Balloons](#)

881.[Boats to Save People](#)

973.[K Closest Points to Origin](#)

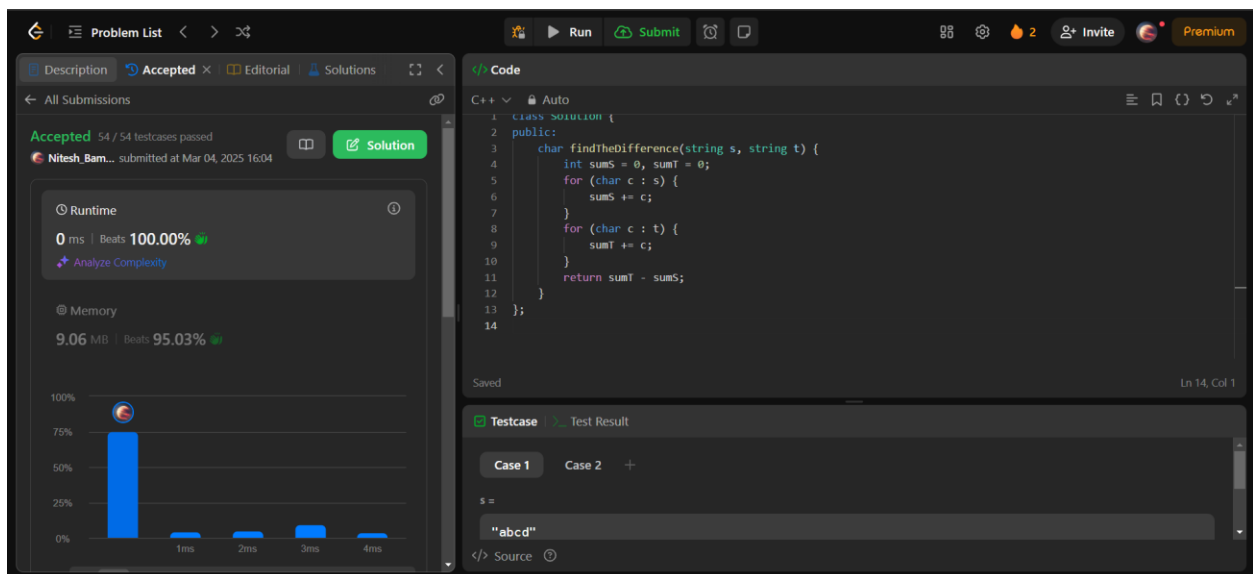
1338.[Reduce Array Size to The Half](#)

389. Find the Difference

Aim - Given two strings *s* and *t*, where *t* is formed by shuffling *s* and adding one extra character, find the extra character.

CODE:-

```
class Solution {  
  
public:  
  
    char findTheDifference(string s, string t) {  
  
        int sumS = 0, sumT = 0;  
  
        for (char c : s) {  
  
            sumS += c;  
  
        }  
  
        for (char c : t) {  
  
            sumT += c;  
  
        }  
  
        return sumT - sumS;  
  
    }  
  
};
```

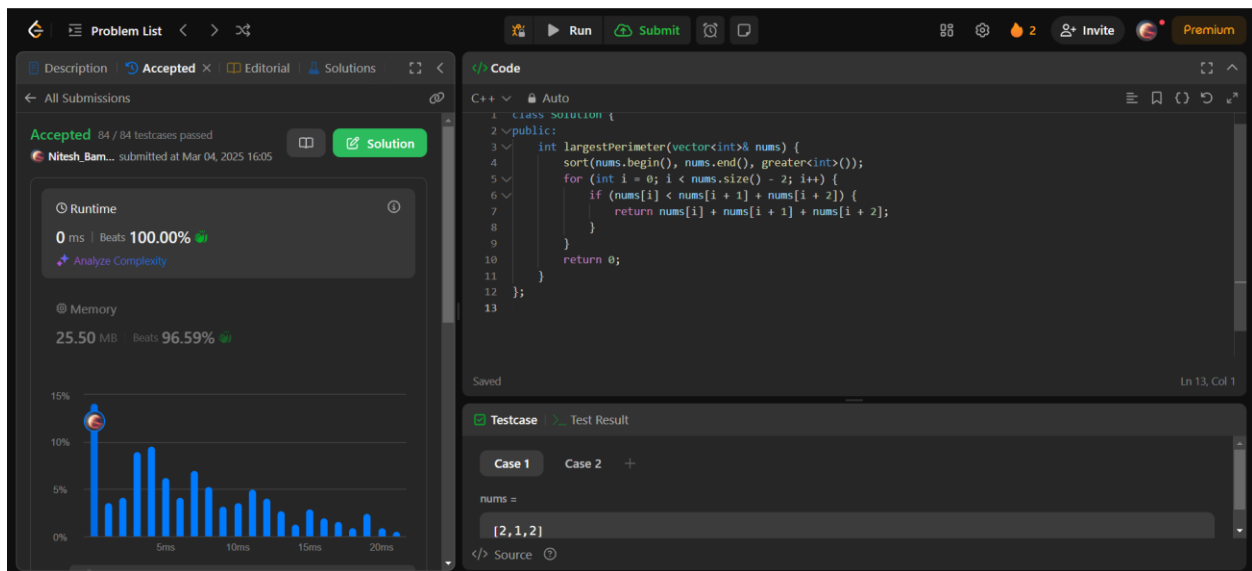


976. Largest Perimeter Triangle

Aim - Given an array of positive integers, return the largest perimeter of a triangle that can be formed using three of these numbers. If no valid triangle can be formed, return 0.

CODE:-

```
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;  
  
    }  
  
};
```

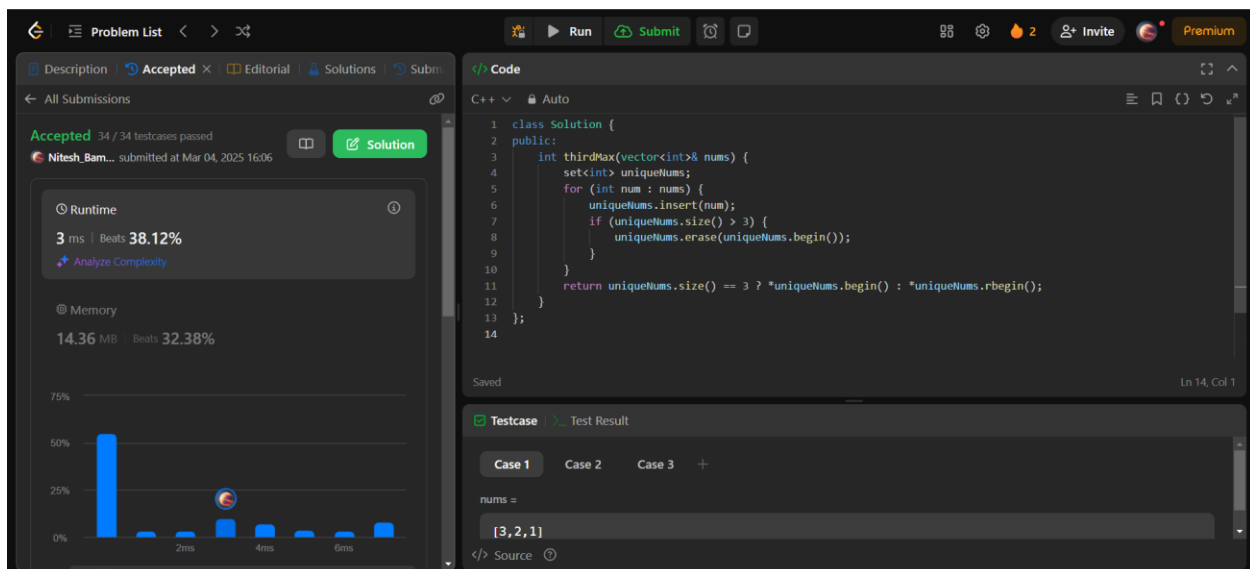


414. Third Maximum Number

Aim - Given an integer array `nums`, return the third largest unique number. If it does not exist, return the maximum number.

CODE:-

```
class Solution {  
public:  
    int thirdMax(vector<int>& nums) {  
        set<int> uniqueNums;  
        for (int num : nums) {  
            uniqueNums.insert(num);  
            if (uniqueNums.size() > 3) {  
                uniqueNums.erase(uniqueNums.begin());  
            }  
        }  
        return uniqueNums.size() == 3 ? *uniqueNums.begin() : *uniqueNums.rbegin();  
    }  
};
```



The screenshot displays a code editor interface with the following components:

- Problem List:** Shows the problem "414. Third Maximum Number" as "Accepted".
- Runtime:** 3 ms, Beats 38.12%.
- Memory:** 14.36 MB, Beats 32.38%.
- Code:** The C++ code is shown in the editor, matching the provided code block.
- Testcase:** A test case is shown with `nums = [3, 2, 1]`.

451. Sort Characters By Frequency

Aim - Given a string s, sort it in decreasing order based on the frequency of characters.

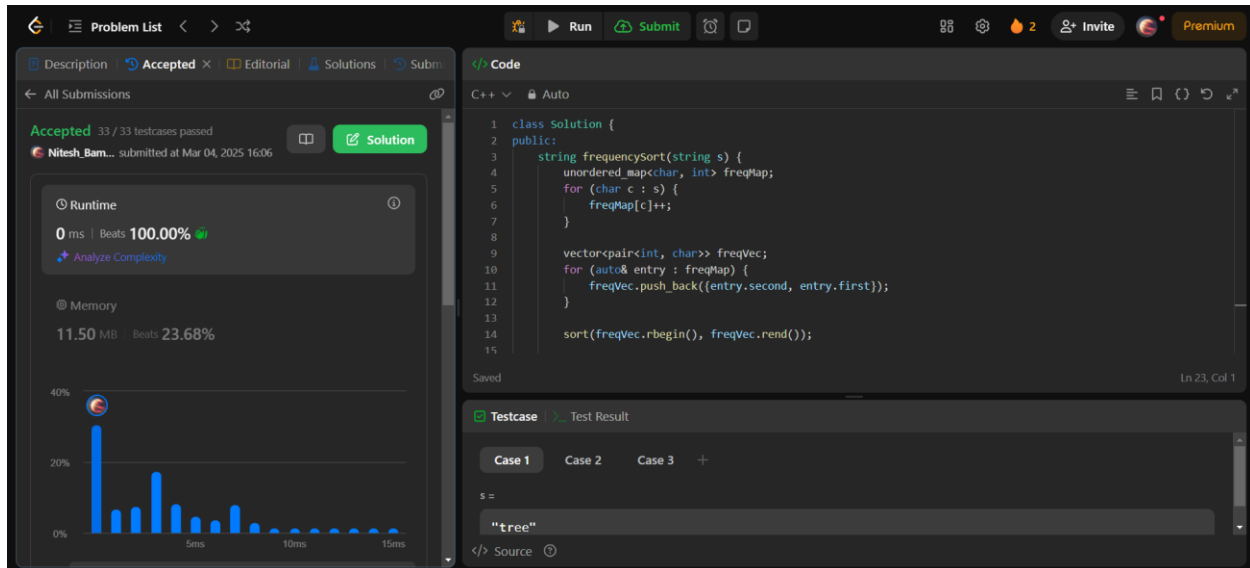
CODE:-

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

        vector<pair<int, char>> freqVec;
        for (auto& entry : freqMap) {
            freqVec.push_back({entry.second, entry.first});
        }

        sort(freqVec.rbegin(), freqVec.rend());

        string result;
        for (auto& entry : freqVec) {
            result.append(entry.first, entry.second);
        }
        return result;
    }
};
```



The screenshot displays a coding platform interface. On the left, a sidebar shows the submission status: 'Accepted' with 33/33 testcases passed. It also shows runtime details: 0 ms (Beats 100.00%) and memory usage: 11.50 MB (Beats 23.68%). A bar chart at the bottom of the sidebar shows the distribution of runtime. The main editor area displays C++ code for a function named 'frequencySort'. The code uses an unordered map to count character frequencies and a vector to store the characters sorted by frequency. The bottom right section shows a test case with the input 'tree'.

452. Minimum Number of Arrows to Burst Balloons

Aim - Given an array of balloon intervals, return the minimum number of arrows needed to burst all balloons.

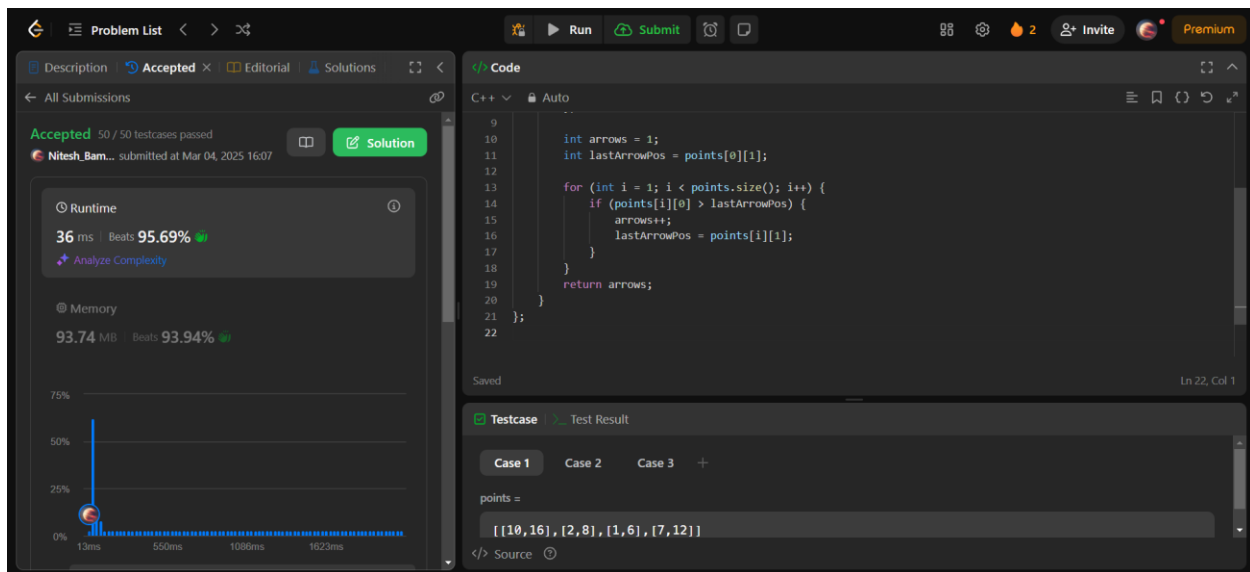
CODE:-

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

        sort(points.begin(), points.end(), [](vector<int>& a, vector<int>& b) {
            return a[1] < b[1];
        });

        int arrows = 1;
        int lastArrowPos = points[0][1];
```

```
        for (int i = 1; i < points.size(); i++) {  
            if (points[i][0] > lastArrowPos) {  
                arrows++;  
                lastArrowPos = points[i][1];  
            }  
        }  
        return arrows;  
    }  
};
```



The screenshot displays a C++ IDE with a solution for a problem. The code is in C++ and implements a greedy algorithm to find the maximum number of arrows that can be shot. The runtime is 36 ms, beating 95.69% of other solutions. The memory usage is 93.74 MB, beating 93.94% of other solutions. The test case shows points = [[10,16], [2,8], [1,6], [7,12]].

```
9  
10  
11 int arrows = 1;  
12 int lastArrowPos = points[0][1];  
13  
14 for (int i = 1; i < points.size(); i++) {  
15     if (points[i][0] > lastArrowPos) {  
16         arrows++;  
17         lastArrowPos = points[i][1];  
18     }  
19 }  
20 return arrows;  
21 }  
22
```

Runtime: 36 ms | Beats 95.69%
Memory: 93.74 MB | Beats 93.94%

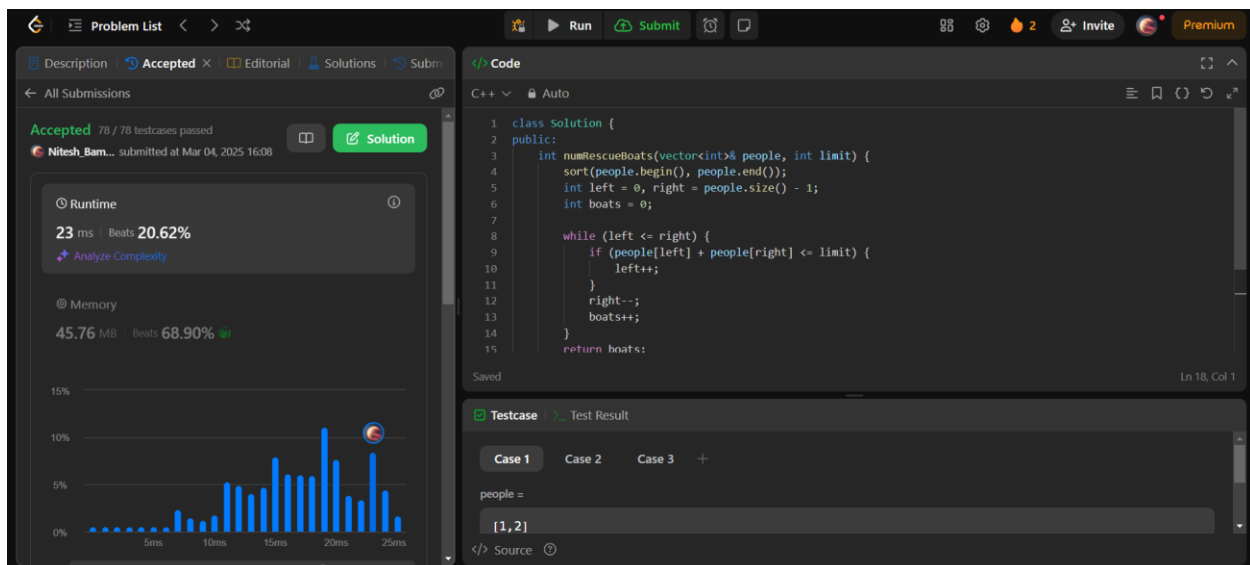
Testcase: Case 1 Case 2 Case 3 +
points =
[[10,16], [2,8], [1,6], [7,12]]

881. Boats to Save People

Aim - Given people's weights and a boat limit, return the minimum number of boats needed to carry everyone. Each boat can carry at most two people.

CODE:-

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



973. K Closest Points to Origin

Aim - Given an array of points and an integer k, return the k closest points to the origin (0,0).

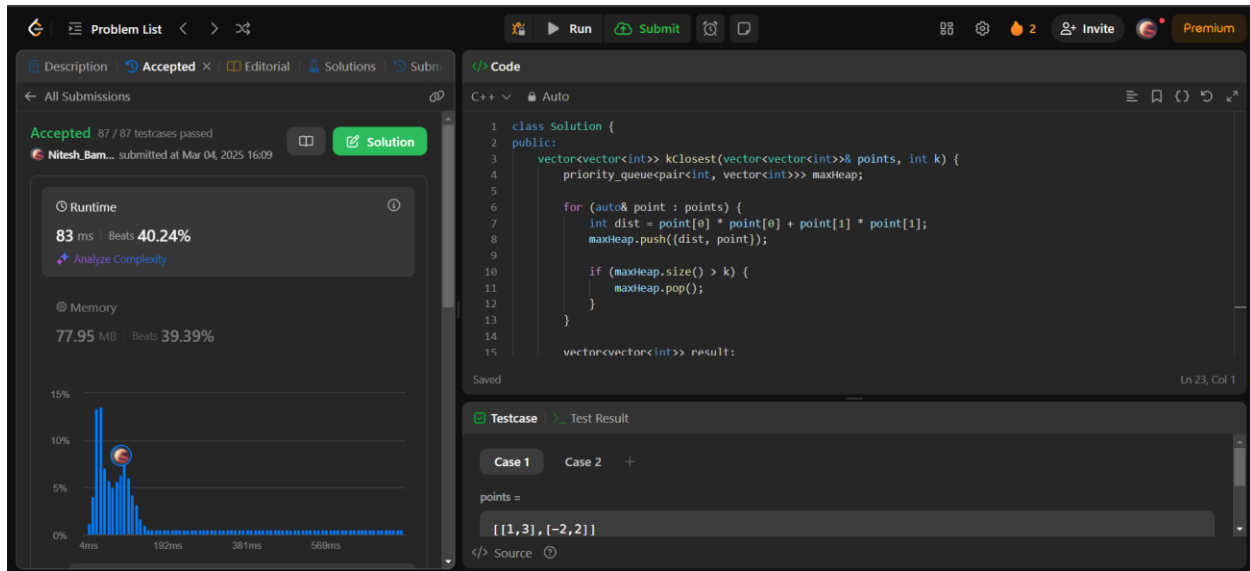
CODE:-

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

        for (auto& point : points) {
            int dist = point[0] * point[0] + point[1] * point[1];
            maxHeap.push({dist, point});

            if (maxHeap.size() > k) {
                maxHeap.pop();
            }
        }

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



1338. Reduce Array Size to The Half

Aim - Given an array, remove the minimum number of elements such that at least half of the elements are removed. Return the minimum number of elements to remove.

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& entry : freq) {
            counts.push_back(entry.second);
        }
    }
};
```

```

    }

    sort(counts.rbegin(), counts.rend());

    int removed = 0, halfSize = arr.size() / 2, setSize = 0;

    for (int count : counts) {

        removed += count;

        setSize++;

        if (removed >= halfSize) {

            break;

        }

    }

    return setSize;

}

};

```

