**Name : Baljeet UID : 22BCS11134 Section : 605-B Assignment-5**

**Ques 1.Find the Difference.**

**Code:**

class Solution {

public:

    char findTheDifference(string s, string t) {

        char result = 0;

    for (char c : s) result ^= c;

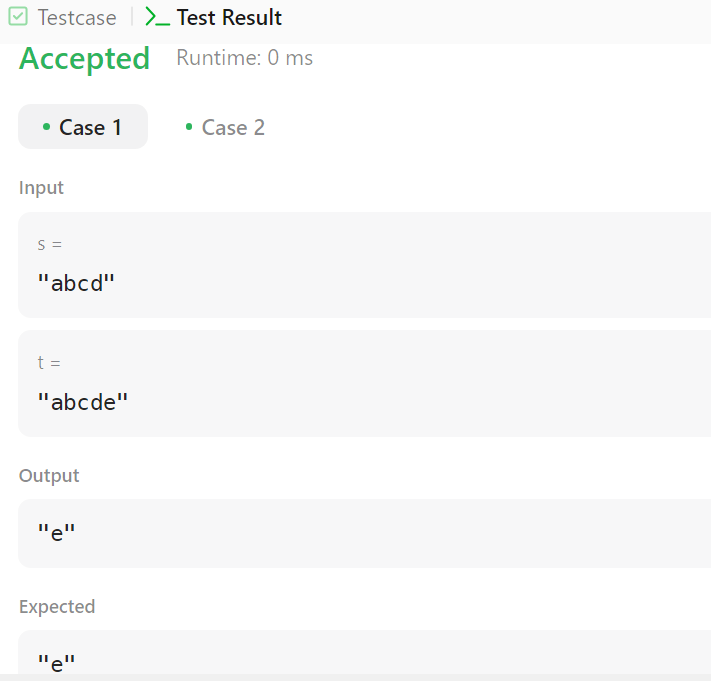
    for (char c : t) result ^= c;

    return result;

    }

};

**Output:**



**Ques 2.** **Largest Perimeter Triangle.**

**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];

            }

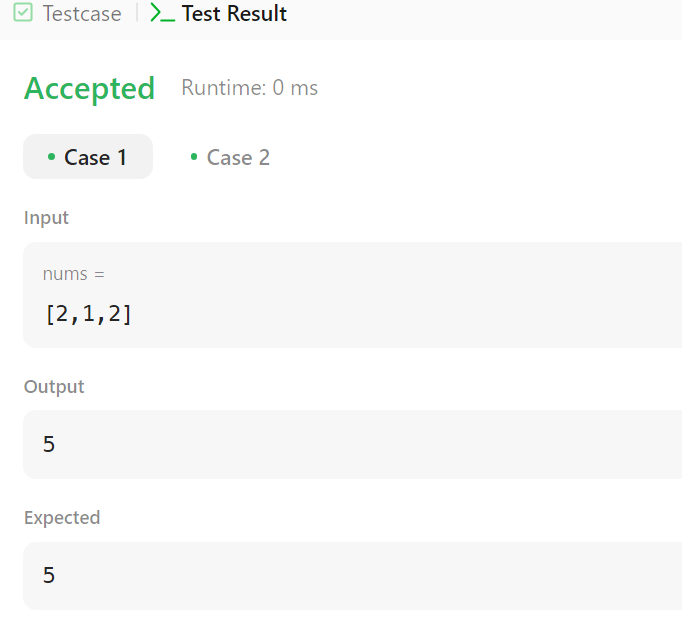
        }

        return 0;

    }

};

**Output:**



**Ques 3.** **Third Maximum Number.**

**Code:**

class Solution {

public:

    int thirdMax(std::vector<int>& nums) {

        set<int> maxSet;

        for (int num : nums) {

            maxSet.insert(num);

            if (maxSet.size() > 3) {

                maxSet.erase(maxSet.begin());

            }

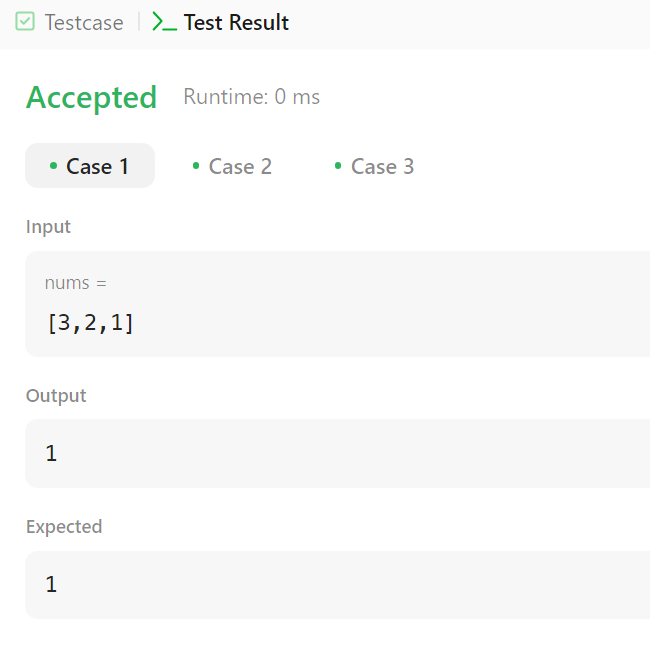
        }

        return (maxSet.size() == 3) ? \*maxSet.begin() : \*maxSet.rbegin();

    }

};

**Output:**



**Ques 4. Sort Characters By Frequency.**

**Code:**

class Solution {

public:

string frequencySort(string s) {

unordered\_map<char, int> freqMap;

for (char c : s) freqMap[c]++;

vector<pair<char, int>> freqVec(freqMap.begin(), freqMap.end());

sort(freqVec.begin(), freqVec.end(), [](const auto &a, const auto &b) {

return a.second > b.second;

});

string result;

for (const auto &pair : freqVec) {

result.append(pair.second, pair.first);

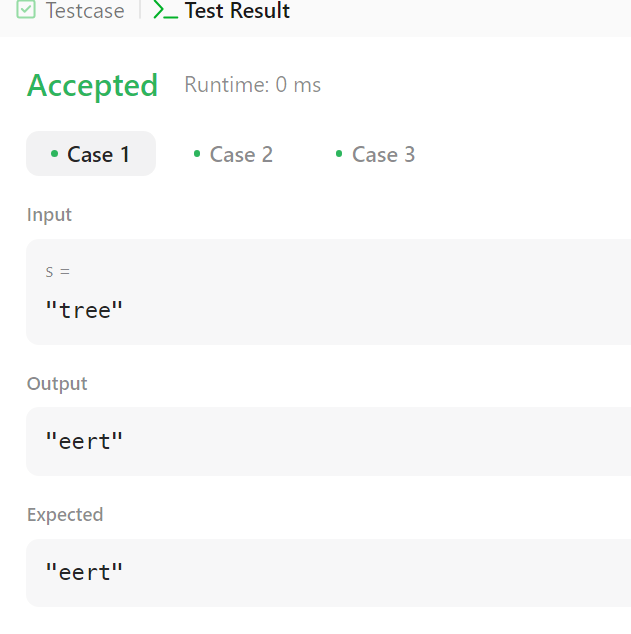
}

return result;

}

};

**Output:**

****

**Ques 5. Minimum Number of Arrows to Burst Balloons.**

**Code:**

class Solution {

public:

int findMinArrowShots(vector<vector<int>>& points) {

sort(points.begin(), points.end(), [](const vector<int>& a, const vector<int>& b) {

return a[1] < b[1];

});

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

for (int i = 1; i < points.size(); i++) {

if (points[i][0] > end) {

arrows++;

end = points[i][1];

}

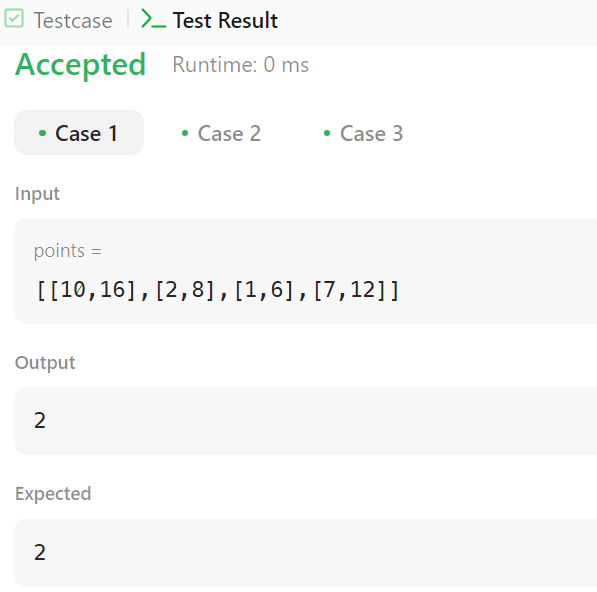
}

return arrows;

}

};

**Output:**



**Ques 6. 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++;

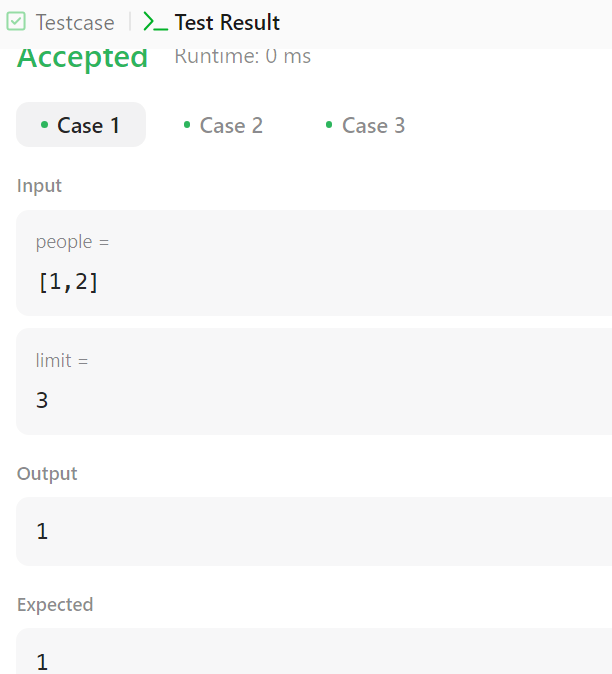
}

return boats;

}

};

**Output:**



**Ques.7 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();

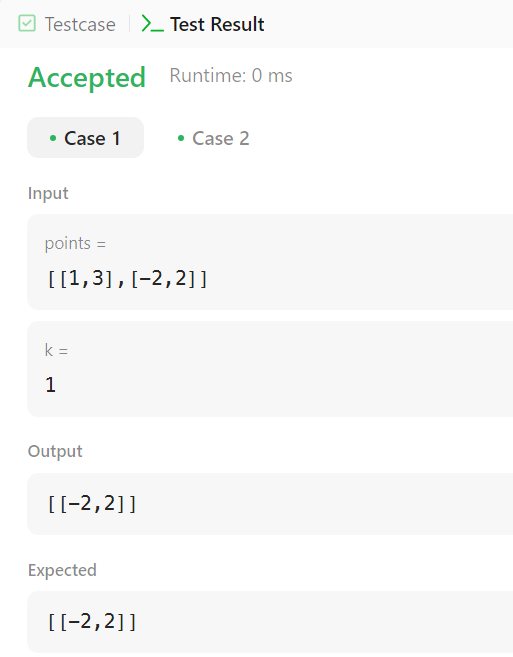
        }

        return result;

    }

};

**Output:**



**Ques 8. Reduce Array Size to The Half.**

**Code:**

class Solution {

public:

int minSetSize(vector<int>& arr) {

unordered\_map<int, int> freqMap;

for (int num : arr) {

freqMap[num]++;

}

vector<int> frequencies;

for (auto& pair : freqMap) {

frequencies.push\_back(pair.second);

}

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

int removed = 0, count = 0, half = arr.size() / 2;

for (int freq : frequencies) {

removed += freq;

count++;

if (removed >= half) {

break;

}

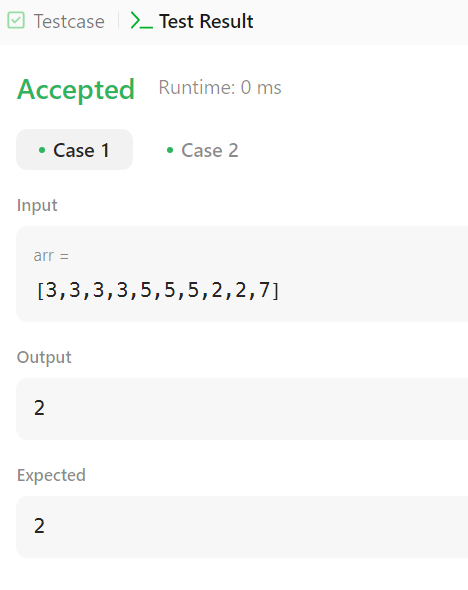
}

return count;

}

};

**Output:**

****