

AP Assignment 5

**aliza asif**

22BCS50175

22BCS\_FL\_IOT\_601\_A

**AP ASSIGNMENT 5**

**Q1.** **Find the Difference**

Implementation 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;

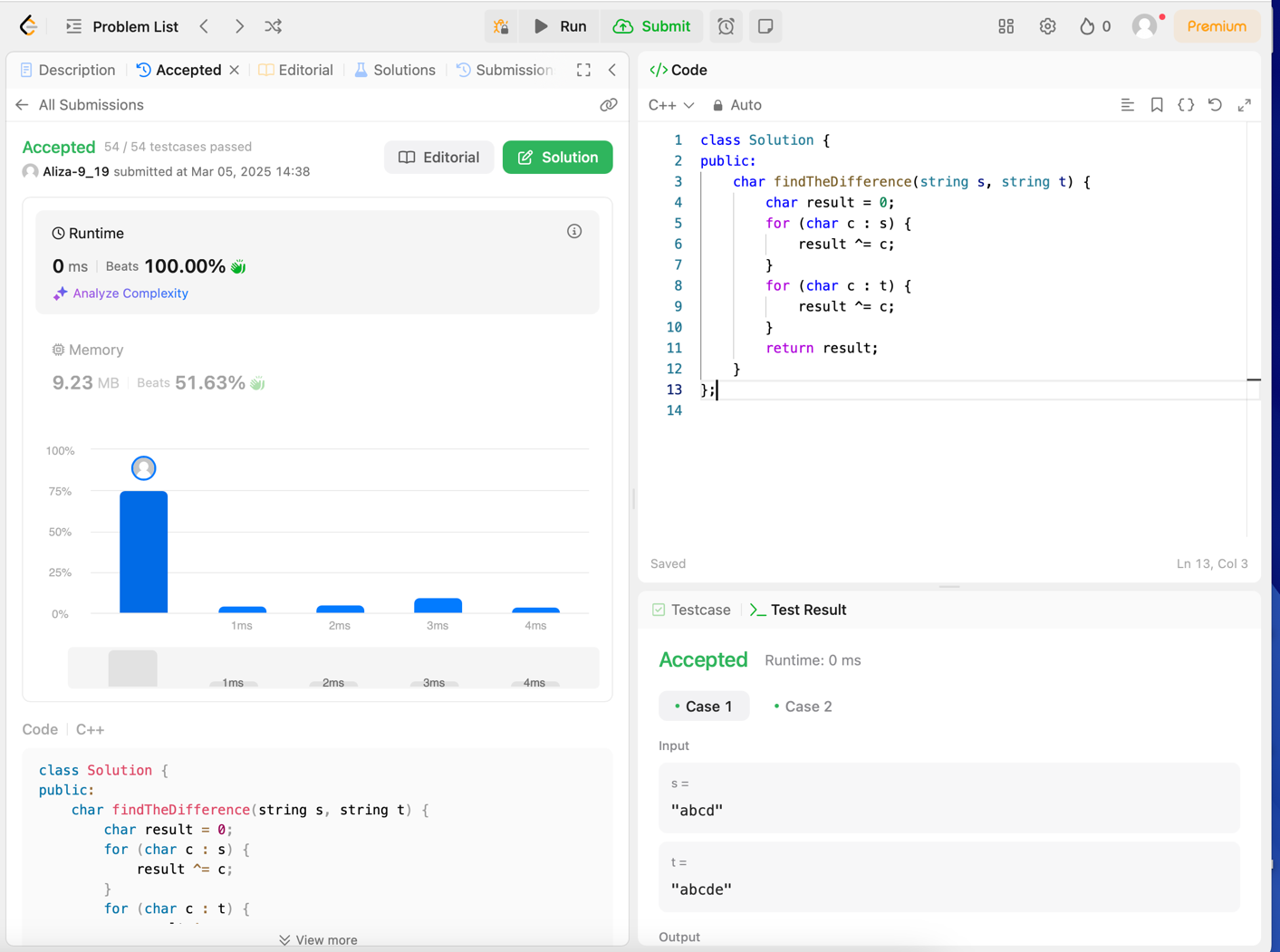
}

return result;

}

};

Output:



**Q2. Largest Perimeter Triangle**

Implementation Code:

class Solution {

public:

int largestPerimeter(vector<int>& nums) {

sort(nums.begin(), nums.end());

for (int i = nums.size() - 3; i >= 0; --i) {

if (nums[i] + nums[i+1] > nums[i+2]) {

return nums[i] + nums[i+1] + nums[i+2];

}

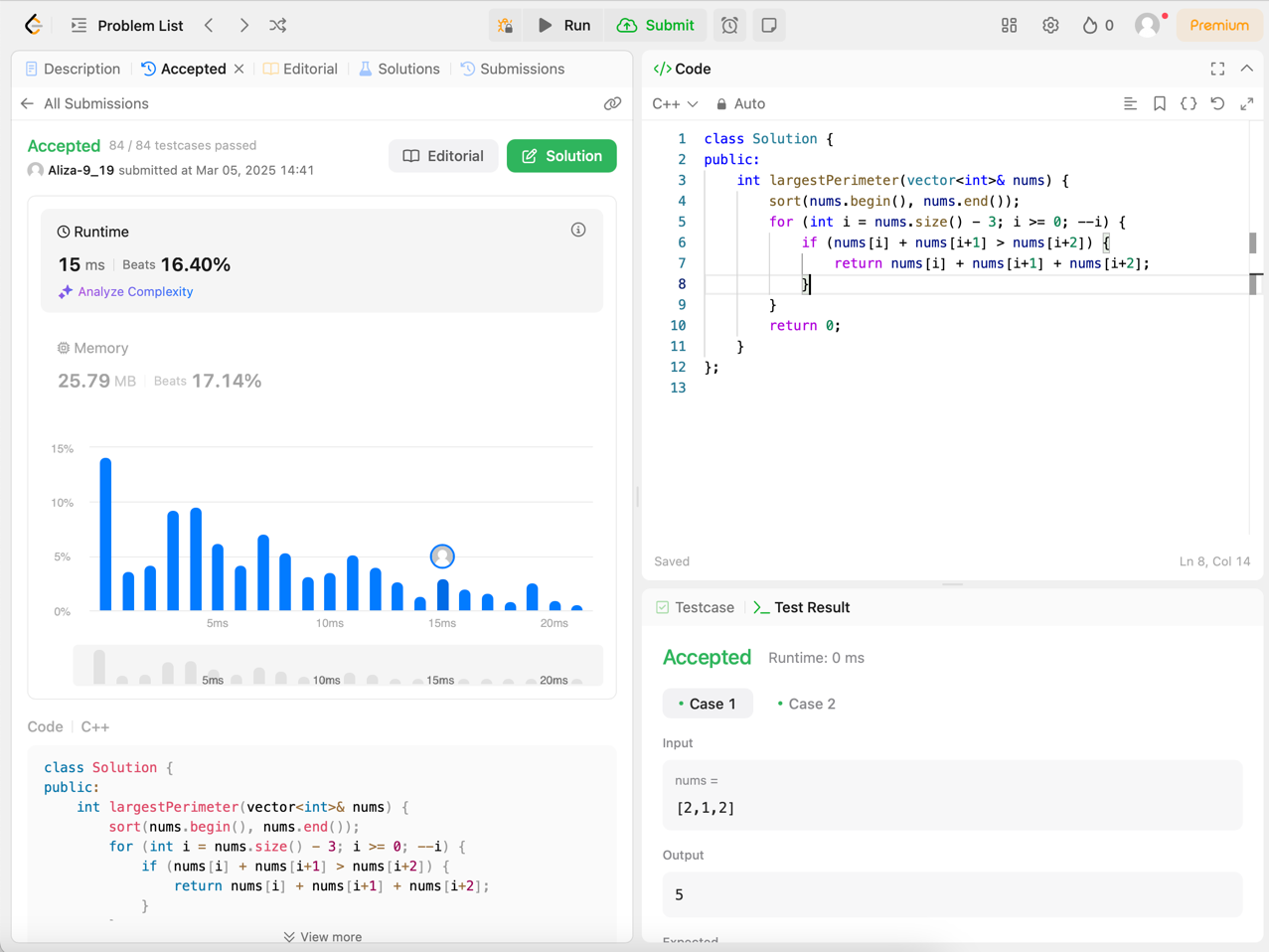
}

return 0;

}

};

Output:



**Q3. Third Maximum Number**

Implementation Code:

class Solution {

public:

int thirdMax(vector<int>& nums) {

set<int> s;

for (int num : nums) {

s.insert(num);

if (s.size() > 3) {

s.erase(s.begin());

}

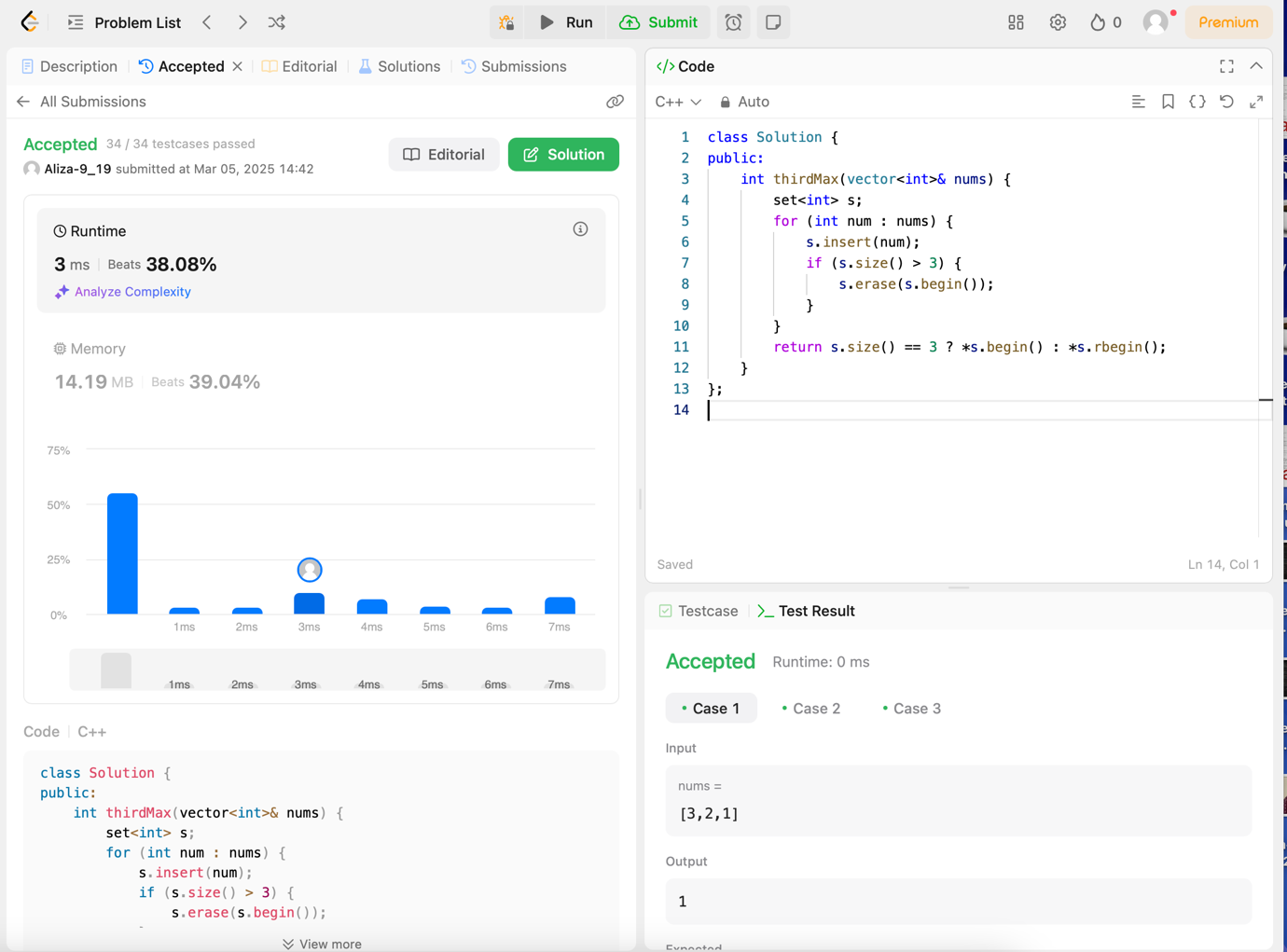
}

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

}

};

Output:



**Q4. Sort Characters by frequency**

Implementation Code:

class Solution {

public:

string frequencySort(string s) {

unordered\_map<char, int> freq;

for (char c : s) {

freq[c]++;

}

vector<pair<int, char>> freqVec;

for (auto& it : freq) {

freqVec.push\_back({it.second, it.first});

}

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

string result;

for (auto& p : freqVec) {

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

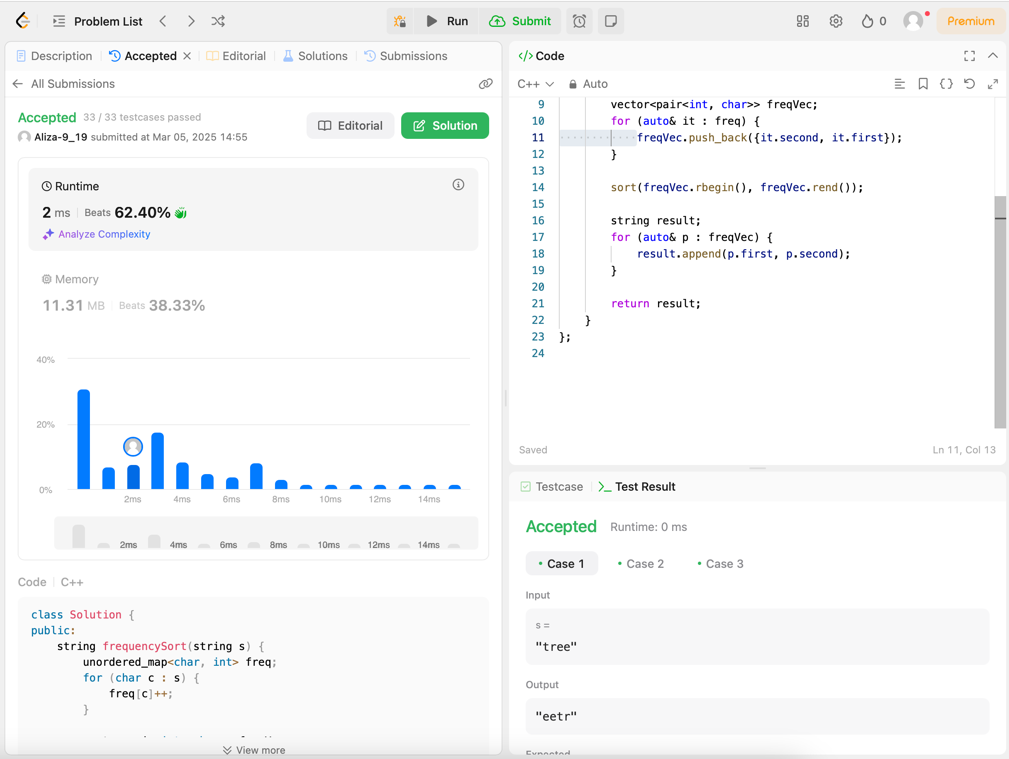
}

return result;

}

};

Output

:

**Q5. Minimum Number of Arrows to Burst Balloons**

Implementation 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;

int end = points[0][1];

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

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

arrows++;

end = points[i][1];

}

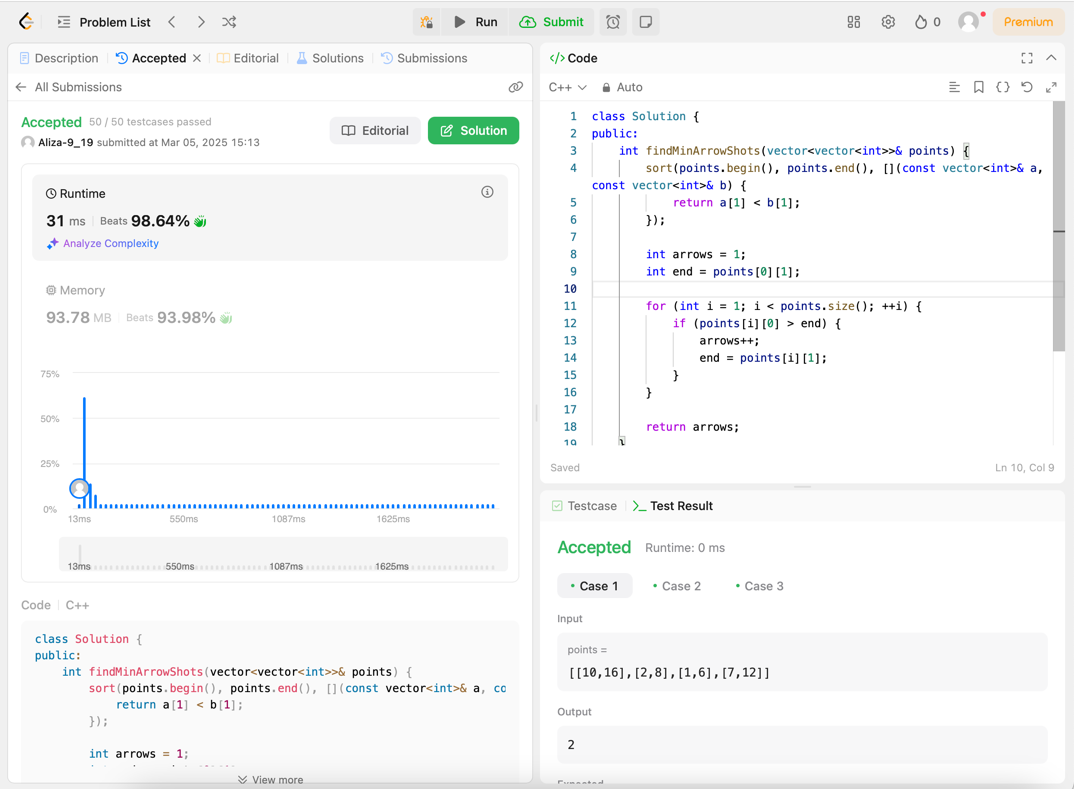
}

return arrows;

}

};

Output:



**Q6. Boats to save people**

Implementation 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++;

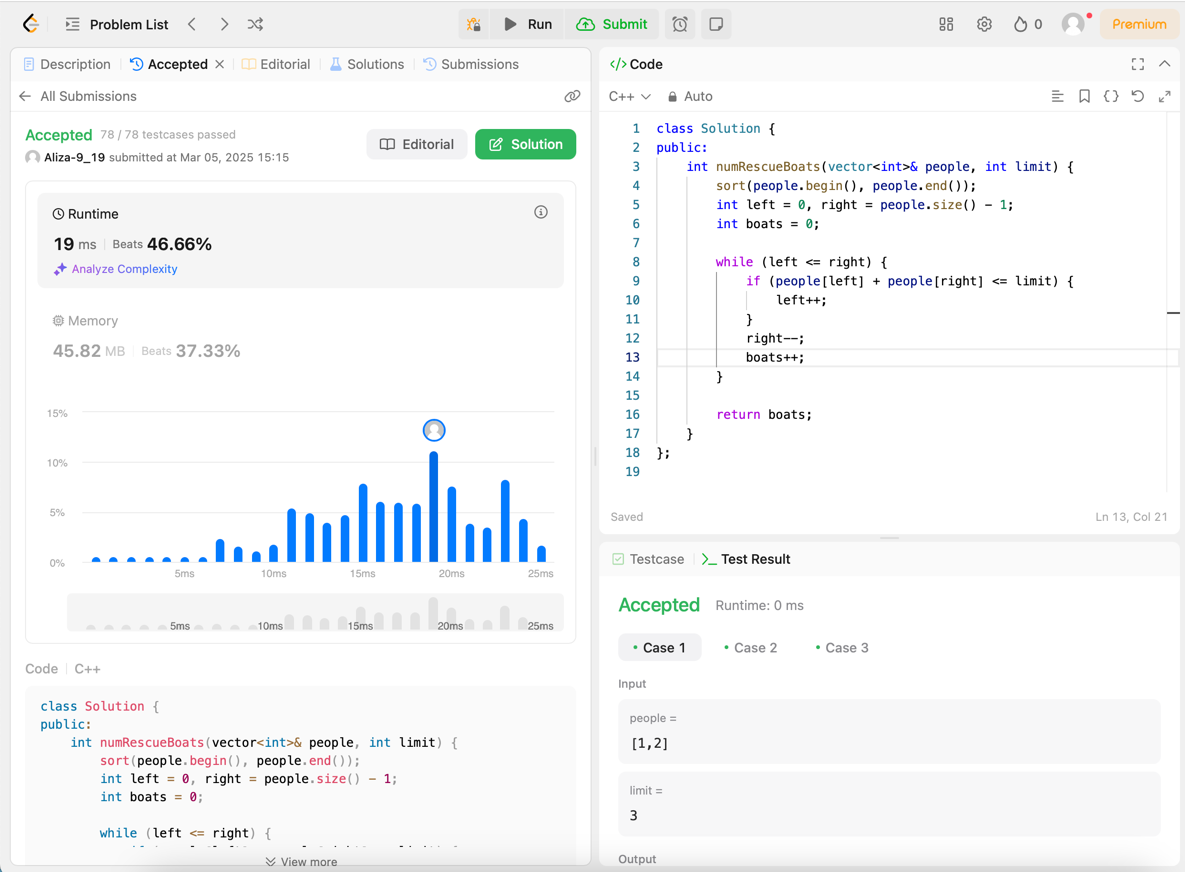
}

return boats;

}

};

Output:



**Q7. K closest points to origin**

Implementation 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();

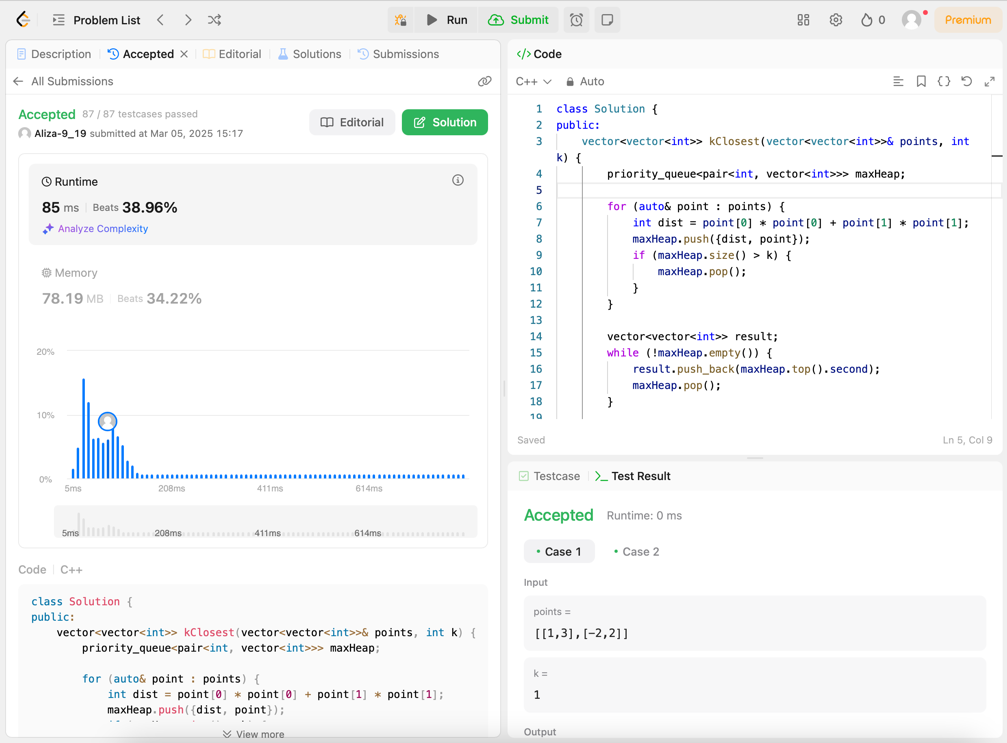
}

return result;

}

};

Output:



**Q8. Reduce array size to half**

Implementation Code:

class Solution {

public:

int minSetSize(vector<int>& arr) {

unordered\_map<int, int> freq;

for (int num : arr) {

freq[num]++;

}

priority\_queue<int> maxHeap;

for (auto& [key, count] : freq) {

maxHeap.push(count);

}

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

while (removed < half) {

removed += maxHeap.top();

maxHeap.pop();

sets++;

}

return sets;

}

};

Output:

