**ADVANCED PROGRAMMING LAB-2 ASSIGNMENT**

**Submitted By:**

**Name- Anshika Yadav**

**UID- 22BCS10051**

**Section-22BCS\_IOT\_605-B**

1. **Find the difference**

class Solution {

public:

char findTheDifference(string s, string t) {

char result = 0;

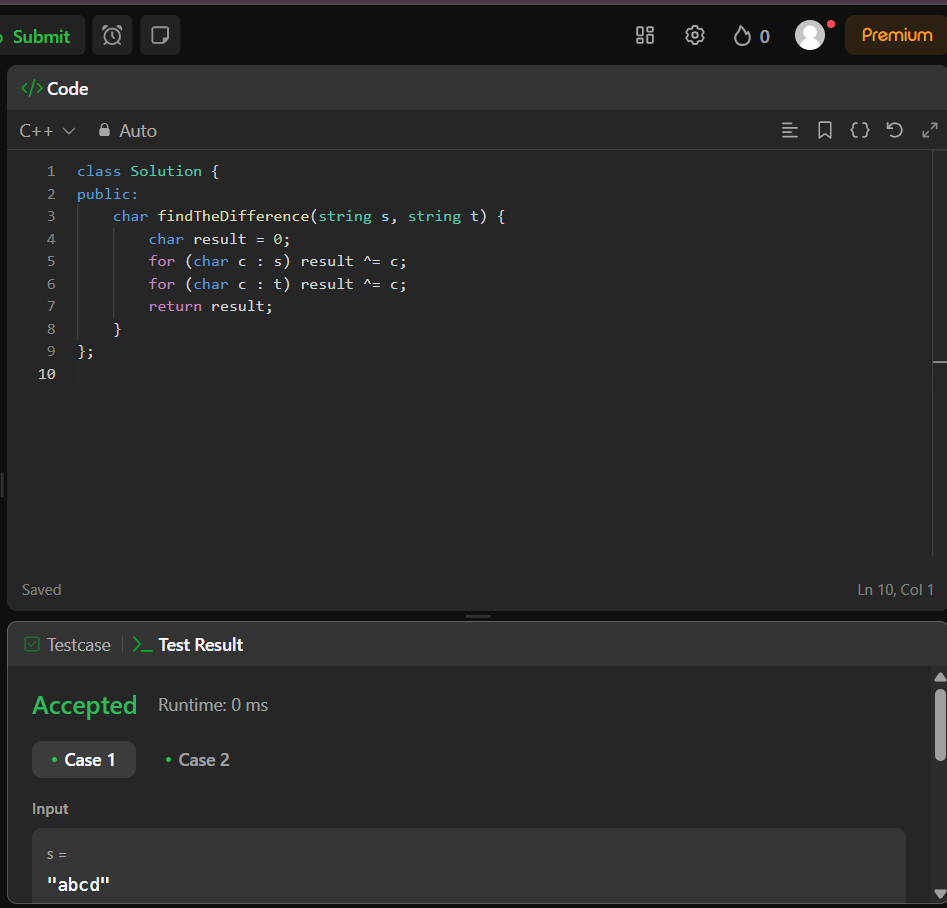
for (char c : s) result ^= c;

for (char c : t) result ^= c;

return result;

}

};



1. **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];

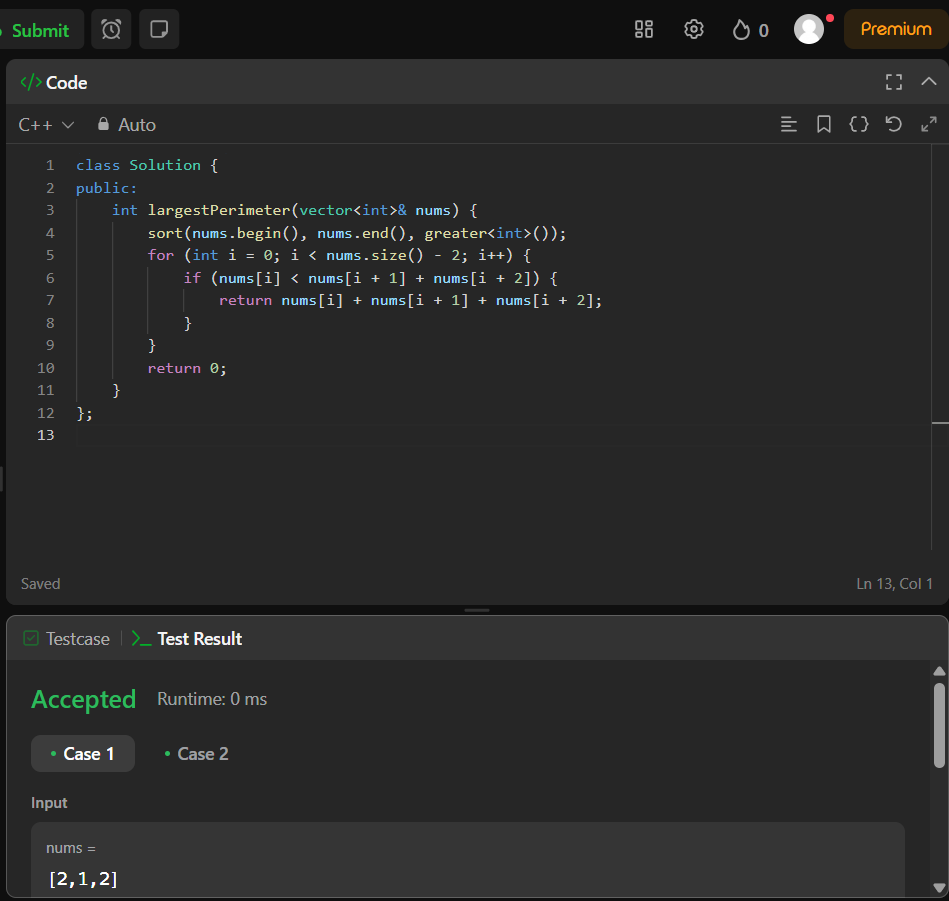
}

}

return 0;

}

};



1. **Third maximum number**

class Solution {

public:

int thirdMax(vector<int>& nums) {

long long first = LLONG\_MIN, second = LLONG\_MIN, third = LLONG\_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;

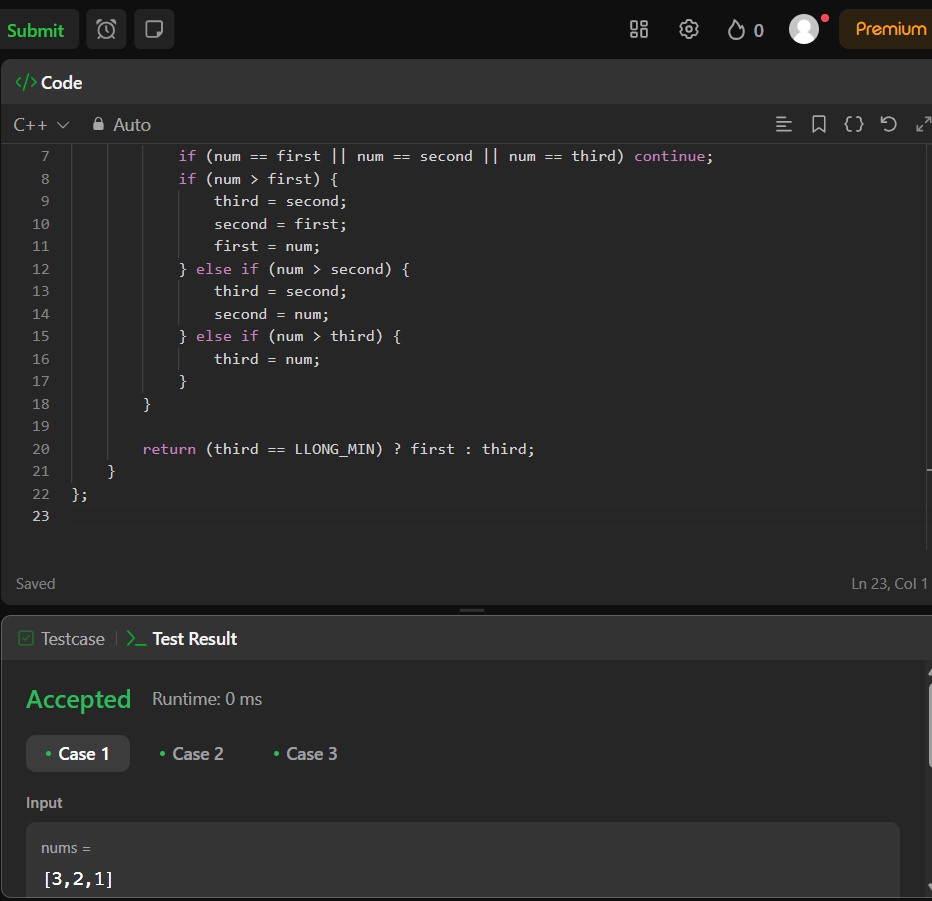
}

}

return (third == LLONG\_MIN) ? first : third;

}

};



1. **Sort characters by frequency**

class Solution {

public:

string frequencySort(string s) {

string ans="";

unordered\_map<char, int> freq;

for(auto x:s){

freq[x]++;

}

vector<pair<int, char>> count;

for(auto x : freq){

count.push\_back({x.second, x.first});

}

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

for(auto x : count){

int i = x.first;

while(i--){

ans += x.second;

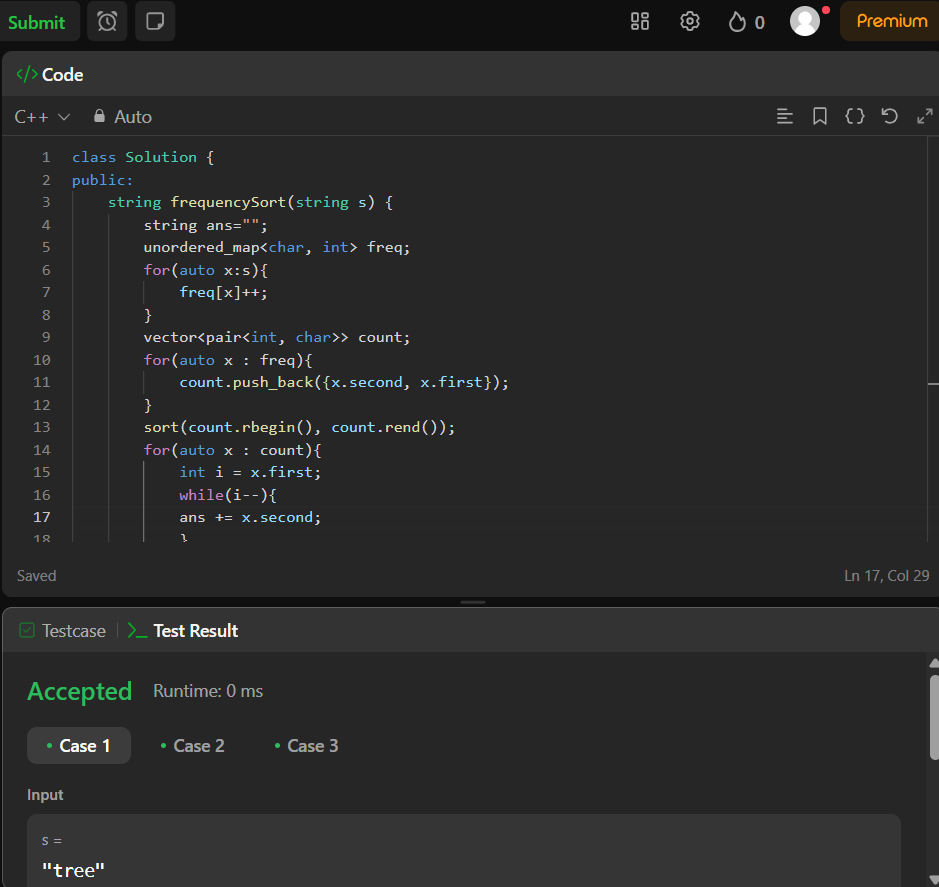
}

}

return ans;

}

};



1. **Minimum number of arrows needed to burst all balloons**

class Solution {

public:

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

if (points.empty()) return 0;

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 (const auto& balloon : points) {

if (balloon[0] > end) {

arrows++;

end = balloon[1];

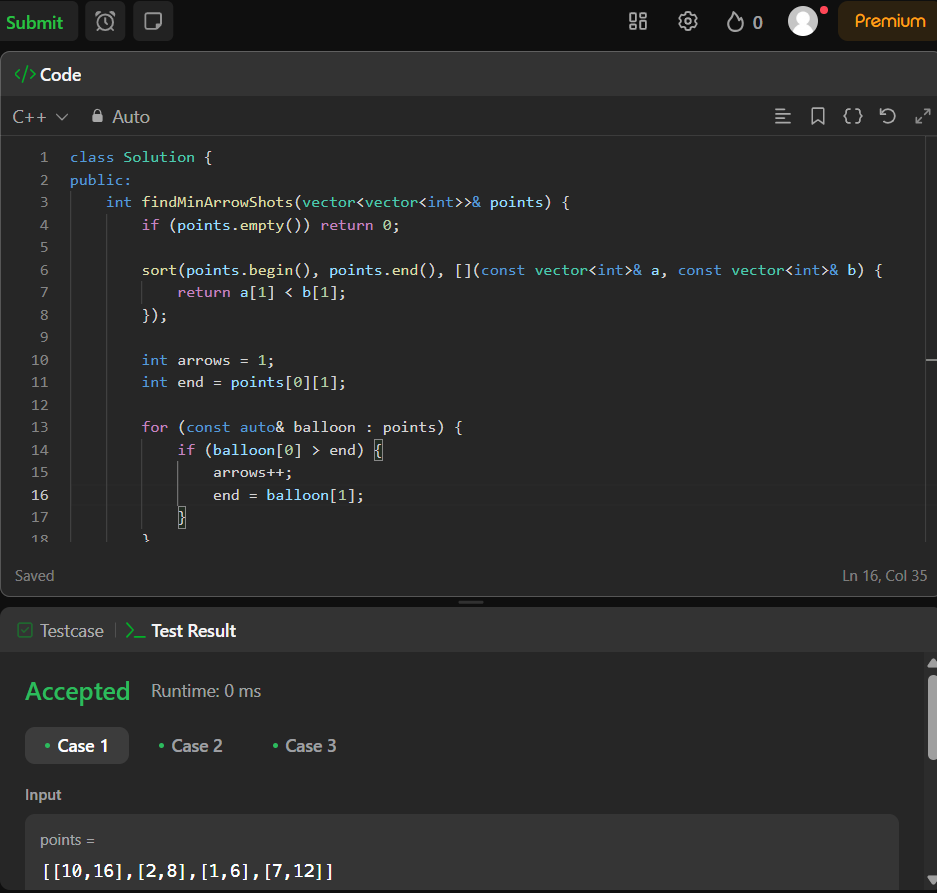
}

}

return arrows;

}

};



1. **Boats to save people**

class Solution {

public:

int numRescueBoats(vector<int>& people, int limit) {

vector<int> sortedPeople = people;

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

int left=0, right=sortedPeople.size()-1;

int boats=0;

while(left<=right){

if(sortedPeople[left]+sortedPeople[right]<=limit){

left++ ;

}

right-- ;

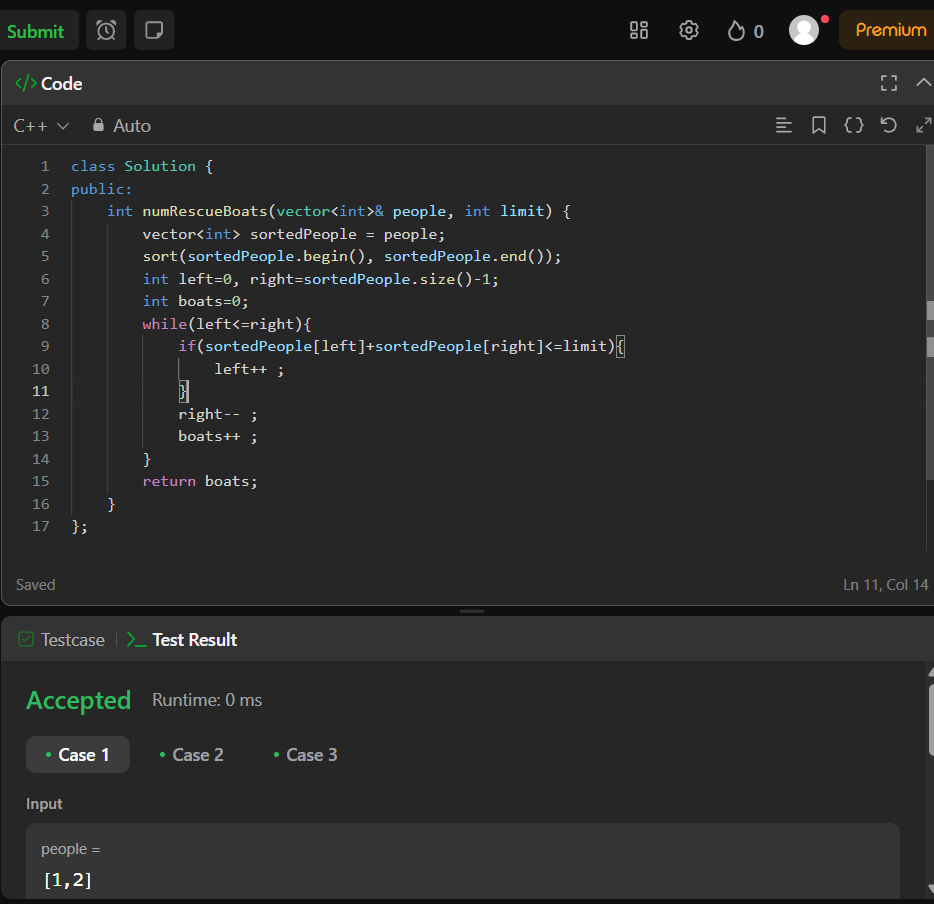
boats++ ;

}

return boats;

}

};



1. **K closest points to the origin**

class Solution {

public:

vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {

priority\_queue<pair<int, vector<int>>> maxHeap;

for (const 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);

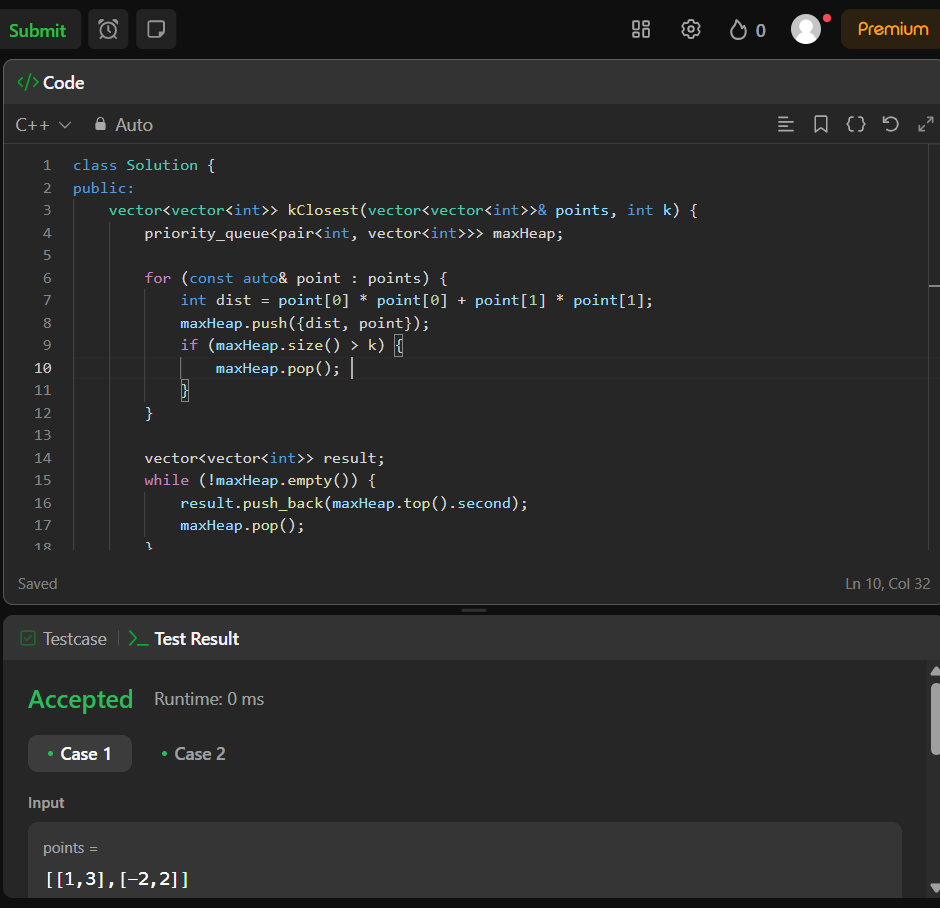
maxHeap.pop();

}

return result;

}

};



1. **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]++;

}

vector<int> counts;

for (const auto& pair : freq) {

counts.push\_back(pair.second);

}

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

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

for (int count : counts) {

removed += count;

elements++;

if (removed >= halfSize) {

return elements;

}

}

return elements;

}

};

