# NAME- Abir Chakraborty| UID- 22BCS14321 | SECTION- 601/A 1 FIND THE DIFFERENCE

class Solution {

public:

char findTheDifference(string s, string t) { unordered\_map<char, int> ans;

for (char a : t) ans[a]++; for (char b : s) ans[b]--;

for (auto it : ans) {

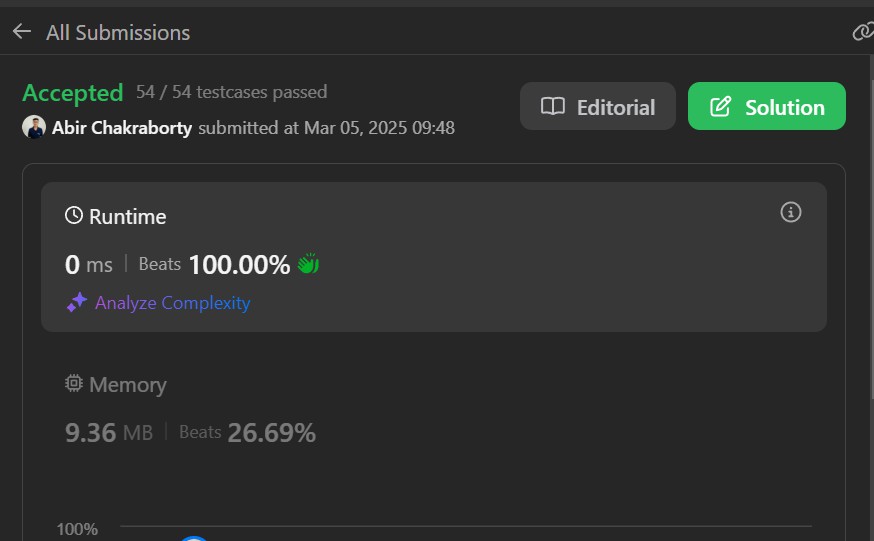
if (it.second > 0) return it.first;

}

return '\0';

}

};



# 2. LARGEST PERIMETER TRIANGLE

class Solution { public:

int largestPerimeter(vector<int>& nums) { sort(nums.begin(),nums.end());

for(int i=nums.size()-1;i>1;i--){

if(nums[i]<nums[i-1]+nums[i-2]){ return nums[i]+nums[i-1]+nums[i-2];

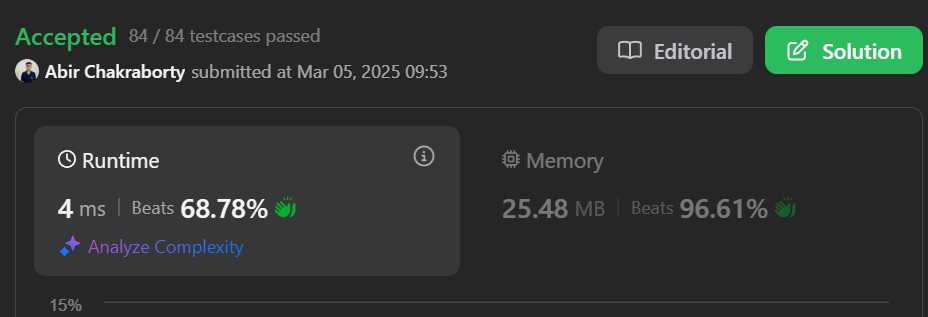
}

}

return 0;

}

};



# 3 THIRD MAXIMUM NUMBER

class Solution { public:

int thirdMax(vector<int>& nums) {

long 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;

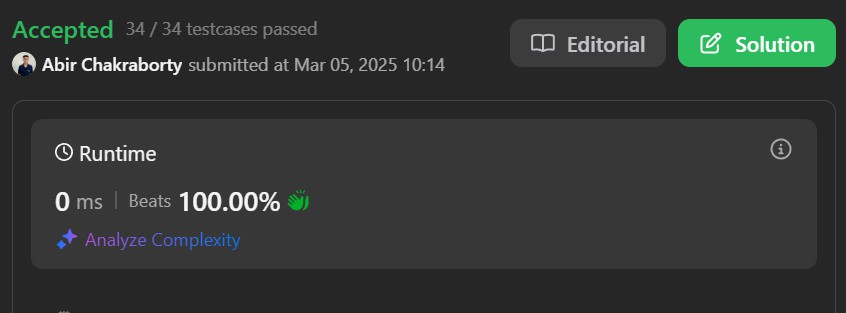
}

}

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

}

};



# SORT CHARACTER BY FREQUENCY

class Solution { public:

string frequencySort(string s) { unordered\_map<char,int> mp; multimap<int,char> r;

string ss="";

for(auto a : s) mp[a]++;

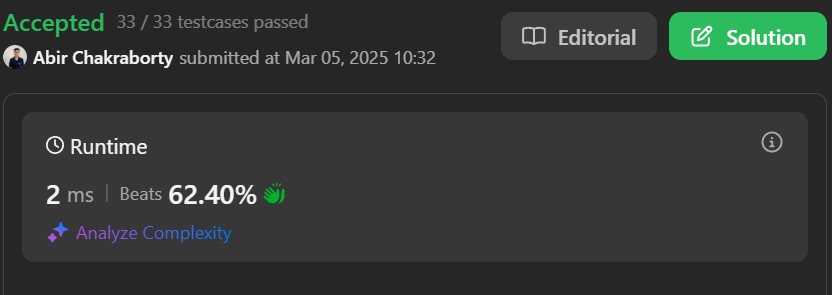
for(auto a : mp) r.insert({a.second, a.first});

for(auto it = r.rbegin(); it != r.rend(); ++it) ss += string(it->first, it->second);

return ss;

}

};



# MINIMUM NUMBER OF ARROWS TO BURST BALLOONS

class Solution { public:

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

return a[0] < b[0];

});

int arrows = 1;

int end = points[0][1];

for (size\_t i = 1; i < points.size(); ++i) { if (points[i][0] > end) {

arrows++;

end = points[i][1];

} else {

end = std::min(end, points[i][1]);

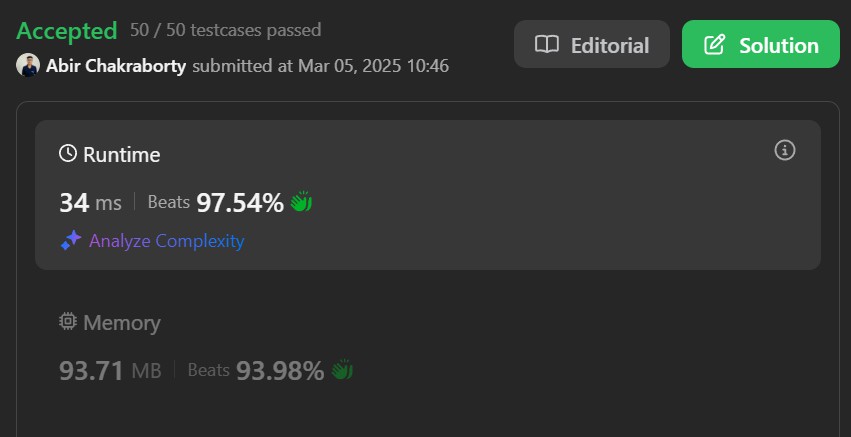
}

}

return arrows;

}

};



## BOATS TO SAVE PEOPLE

class Solution { public:

int numRescueBoats(vector<int>& people, int limit) { int boatCount = 0;

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

int left = 0;

int right = people.size() - 1;

while(left <= right){

int sum = people[left] + people[right]; if(sum <= limit){

boatCount++; left++;

right--;

}

else{

boatCount++; right--;

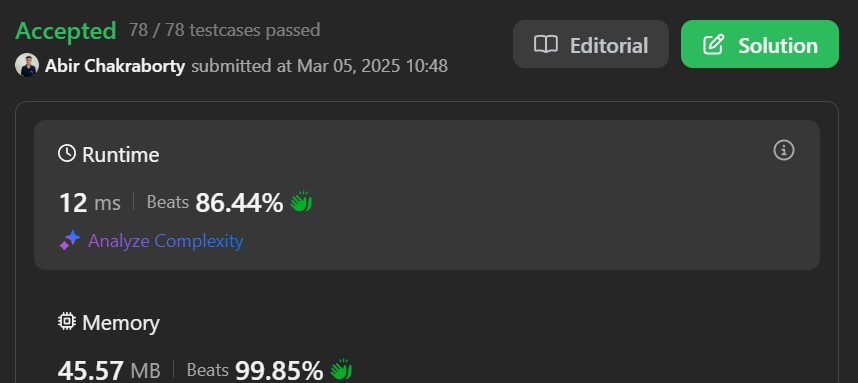
}

}

return boatCount;

}

};



## K CLOSEST POINT TO ORIGIN

class Solution { public:

vector<vector<int>> kClosest(std::vector<std::vector<int>>& points, int k) { auto k\_it = points.begin() + k;

std::nth\_element(points.begin(), k\_it, points.end(), comp);

return std::vector(std::make\_move\_iterator(points.begin()), std::make\_move\_iterator(k\_it));

}

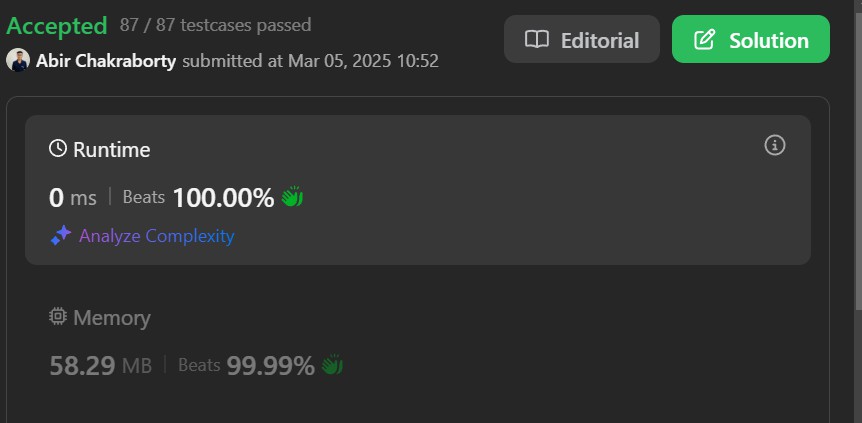
private:

static bool comp(const std::vector<int>& p, const std::vector<int>& q) { int dp = p[0] \* p[0] + p[1] \* p[1];

int dq = q[0] \* q[0] + q[1] \* q[1]; return dp < dq;

}

};



## REDUCE ARRAY SIZE TO HALF

class Solution { public:

int minSetSize(vector<int>& arr) { unordered\_map<int, int> counter; priority\_queue<int> q;

int res = 0, removed = 0;

for (auto a : arr) counter[a]++;

for (auto c : counter) q.push(c.second);

while (removed < arr.size() / 2) { removed += q.top();

q.pop(); res++;

}

return res;

}

};

