

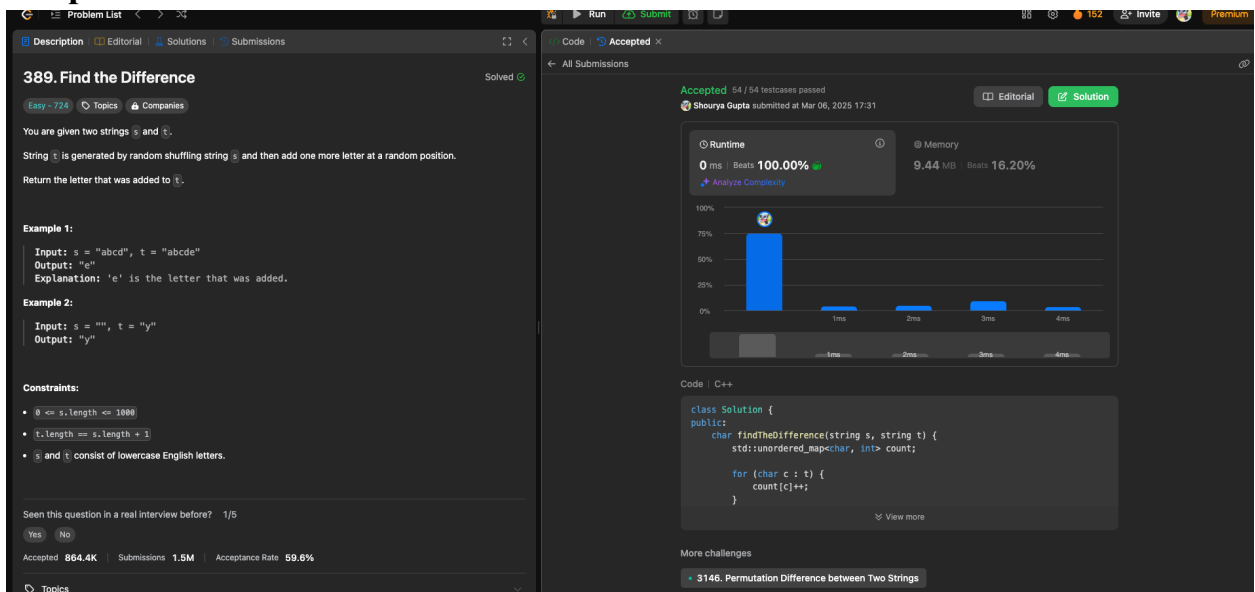
ASSIGNMENT -5 (ADVANCED PROGRAMMING) PISINI JOEL – 22BCS12690

1. Problem 1: Find the Difference.

2. Implementation/Code:

```
class Solution {
    public char findTheDifference(String s, String t) {
        int ssum = 0;
        int tsum = 0;
        for(int i=0;i<s.length();i++)
        {ssum = ssum + (int)s.charAt(i); }
        for(int i=0;i<t.length();i++) {
            tsum = tsum + (int)t.charAt(i); }
        int value = tsum - ssum;
        return (char)value;
    }
}
```

3. Output:

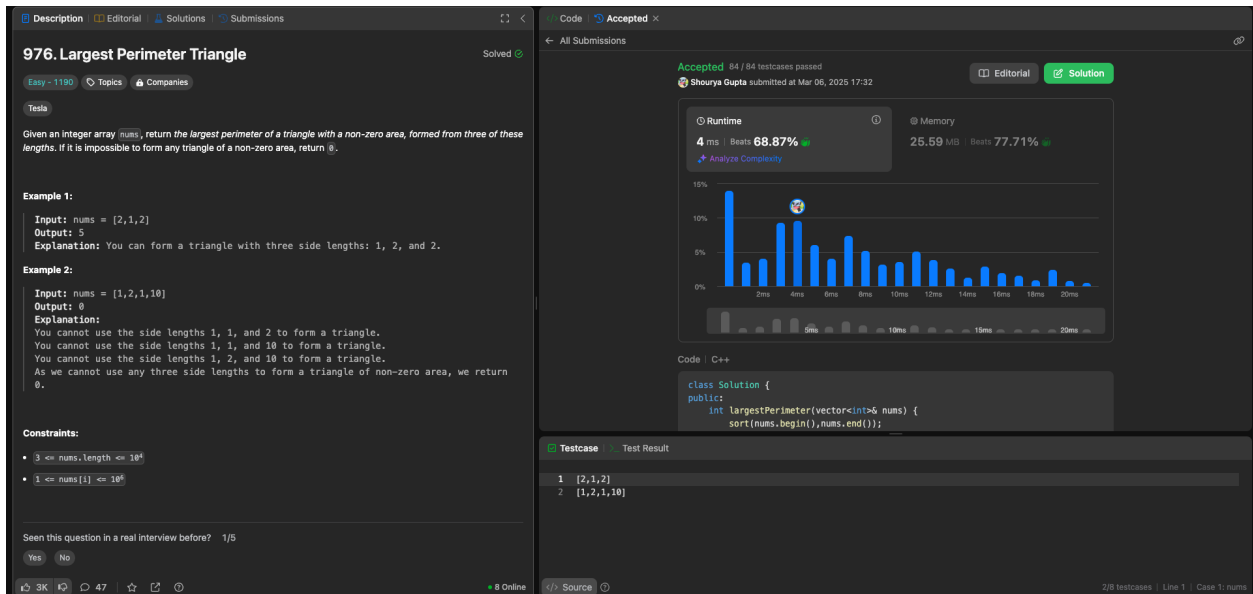


1. Problem 2: Largest Perimeter Triangle

2. Implementation/Code:

```
class Solution {
    public int largestPerimeter(int[] nums) {
        Arrays.sort(nums);
        for(int i = nums.length-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. Output:



The screenshot displays a coding platform interface for the problem "976. Largest Perimeter Triangle". The left panel shows the problem description, examples, and constraints. The right panel shows the solution status as "Accepted", a runtime performance graph, and the C++ code.

Problem Description: Given an integer array `nums`, return the largest perimeter of a triangle with a non-zero area, formed from three of these lengths. If it is impossible to form any triangle of a non-zero area, return 0.

Example 1:
Input: `nums = [2,1,2]`
Output: 5
Explanation: You can form a triangle with three side lengths: 1, 2, and 2.

Example 2:
Input: `nums = [1,2,1,10]`
Output: 0
Explanation: You cannot use the side lengths 1, 1, and 2 to form a triangle. You cannot use the side lengths 1, 1, and 10 to form a triangle. You cannot use the side lengths 1, 2, and 10 to form a triangle. As we cannot use any three side lengths to form a triangle of non-zero area, we return 0.

Constraints:

- $3 \leq \text{nums.length} \leq 10^4$
- $1 \leq \text{nums}[i] \leq 10^5$

Runtime Performance: 4 ms, Beats 68.87%, 25.59 MB, Beats 77.71%.

Code (C++):

```
class Solution {
public:
    int largestPerimeter(vector<int>& nums) {
        sort(nums.begin(), nums.end());
    }
};
```

Testcase: 1 [2,1,2], 2 [1,2,1,10]

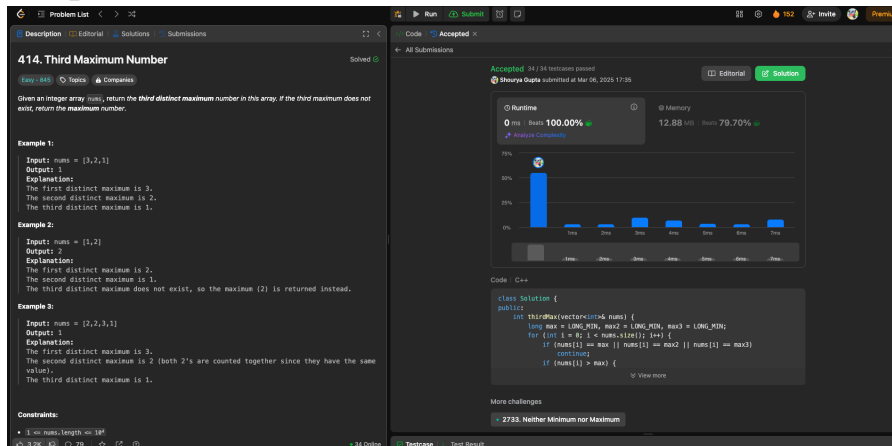
1. Problem 3: Third Maximum Number

2. Implementation/code:

```
class Solution {
public int thirdMax(int[] nums) {
    Integer max1 = null;
    Integer max2 = null;
    Integer max3 = null;
    for (Integer n : nums) {
        if (n.equals(max1) || n.equals(max2) || n.equals(max3)) continue;
        if (max1 == null || n > max1) {
            max3 = max2;
            max2 = max1;
            max1 = n;
        } else if (max2 == null || n > max2) {
            max3 = max2;
            max2 = n;
        } else if (max3 == null || n > max3) {
            max3 = n; } }
    return max3 == null ? max1 : max3; }}

```

3. Output:



The screenshot displays a coding platform interface. On the left, the problem description for '414. Third Maximum Number' is shown, including examples and constraints. On the right, the solution code in C++ is displayed, along with a runtime performance graph showing 100% success rate and a memory usage of 12.88 MB.

Problem Description:

Given an integer array `nums`, return the *third distinct maximum number* in this array. If the third maximum does not exist, return the *maximum number*.

Example 1:

Input: `nums = [3,2,1]`
Output: `1`
Explanation: The first distinct maximum is 3, the second distinct maximum is 2, the third distinct maximum is 1.

Example 2:

Input: `nums = [1,2]`
Output: `2`
Explanation: The first distinct maximum is 2, the second distinct maximum is 1, the third distinct maximum does not exist, so the maximum (2) is returned instead.

Example 3:

Input: `nums = [2,2,3,1]`
Output: `1`
Explanation: The first distinct maximum is 3, the second distinct maximum is 2 (both 2's are counted together since they have the same value), the third distinct maximum is 1.

Constraints:

- 1 ≤ `nums.length` ≤ 10⁴
- 2³¹ ≤ `nums[i]` ≤ 2³¹ - 1

Solution Code (C++):

```
class Solution {
public:
    int thirdMax(vector<int>& nums) {
        long max = LONG_MIN, max2 = LONG_MIN, max3 = LONG_MIN;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] == max || nums[i] == max2 || nums[i] == max3) continue;
            if (nums[i] > max) {
                max3 = max2;
                max2 = max;
                max = nums[i];
            } else if (nums[i] > max2) {
                max3 = max2;
                max2 = nums[i];
            } else if (nums[i] > max3) {
                max3 = nums[i];
            }
        }
        return max3 == LONG_MIN ? max : max3;
    }
};

```

Runtime Performance:

- Runtime: 0 ms, Beats 100.00%
- Memory: 12.88 MB, Beats 79.70%

More challenges:

- 2733. Neither Minimum nor Maximum

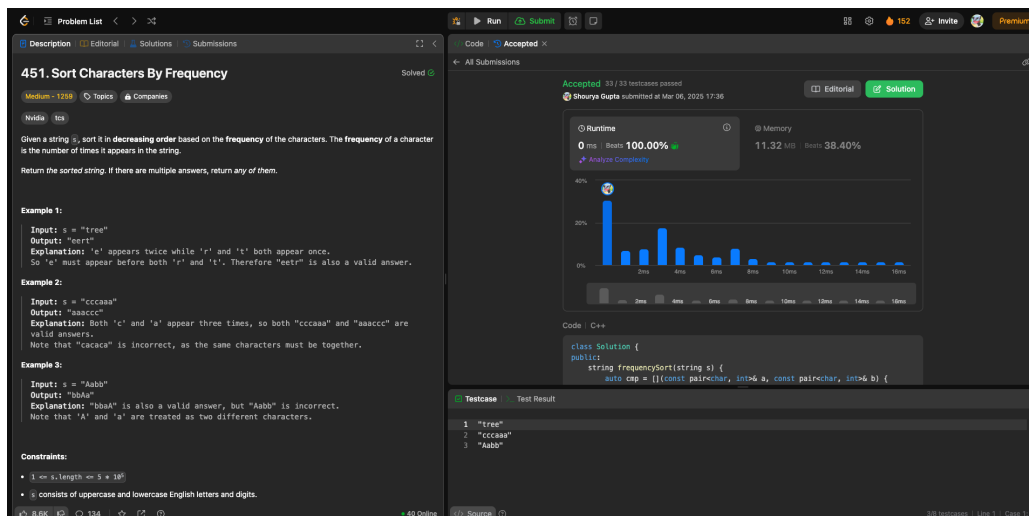
1. Problem 4: Sort Characters By Frequency

2. Implementation/code:

```
import java.util.*;

class Solution {
    public String frequencySort(String s) {
        Map<Character, Integer> frequencyMap = new HashMap<>();
        for (char c : s.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
        }
        PriorityQueue<Character> maxHeap = new PriorityQueue<>((a, b) -> frequencyMap.get(b) - frequencyMap.get(a));
        maxHeap.addAll(frequencyMap.keySet());
        StringBuilder result = new StringBuilder();
        while (!maxHeap.isEmpty()) {
            char c = maxHeap.poll();
            result.append(String.valueOf(c).repeat(frequencyMap.get(c)));
        }
        return result.toString();
    }
}
```

3. Output:



The screenshot displays a coding problem interface. On the left, the problem description for '451. Sort Characters By Frequency' is shown, including examples and constraints. On the right, the solution code is displayed, along with a runtime analysis graph and test case results.

Problem Description:

Given a string *s*, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string.

Return the sorted string. If there are multiple answers, return any of them.

Example 1:

Input: *s* = "tree"

Output: "eert"

Explanation: 'e' appears twice while 'r' and 't' both appear once. So 'e' must appear before both 'r' and 't'. Therefore "eert" is a valid answer.

Example 2:

Input: *s* = "cccaaa"

Output: "aaaccc"

Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers. Note that "cacaca" is incorrect, as the same characters must be together.

Example 3:

Input: *s* = "Aabb"

Output: "bbaA"

Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect. Note that 'A' and 'a' are treated as two different characters.

Constraints:

- 1 ≤ *s*.length ≤ 5 × 10⁵
- s* consists of uppercase and lowercase English letters and digits.

Runtime Analysis:

Accepted: 31 / 31 testcases passed

Shourya Gupta submitted at Mar 06, 2025 17:36

Runtime: 0 ms | Beats: 100.00% | Memory: 11.32 MB | Beats: 38.40%

Testcase Results:

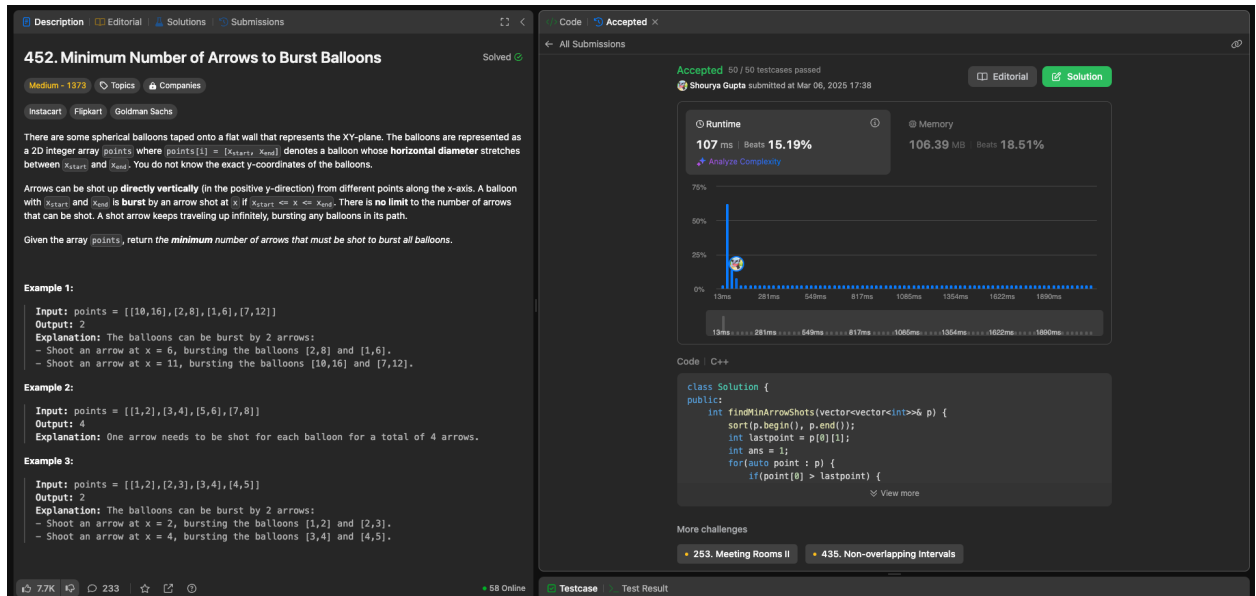
Testcase	Test Result
1 "tree"	Accepted
2 "cccaaa"	Accepted
3 "Aabb"	Accepted

1. Problem 5: Minimum Number of Arrows to Burst Balloons

2. Implementation/Code:

```
class Solution {
public:
    int findMinArrowShots(vector<vector<int>>> segments) {
        Arrays.sort(segments, (a, b) -> Integer.compare(a[1], b[1]));
        int ans = 0, arrow = 0;
        for (int i = 0; i < segments.length; i++) {
            if (ans == 0 || segments[i][0] > arrow) {
                ans++;
                arrow = segments[i][1];
            }
        }
        return ans;
    }
}
```

3. Output:



452. Minimum Number of Arrows to Burst Balloons Solved

Medium - 1373 Topics Companies

Instacart Flipkart Goldman Sachs

There are some spherical balloons taped onto a flat wall that represents the XY-plane. The balloons are represented as a 2D integer array `points` where `points[i] = [xstart, xend]` denotes a balloon whose horizontal diameter stretches between `xstart` and `xend`. You do not know the exact y-coordinates of the balloons.

Arrows can be shot up directly vertically (in the positive y-direction) from different points along the x-axis. A balloon with `xstart` and `xend` is burst by an arrow shot at `x` if `xstart ≤ x ≤ xend`. There is no limit to the number of arrows that can be shot. A shot arrow keeps traveling up infinitely, bursting any balloons in its path.

Given the array `points`, return the *minimum* number of arrows that must be shot to burst all balloons.

Example 1:
Input: `points = [[10,16],[2,8],[1,6],[7,12]]`
Output: 2
Explanation: The balloons can be burst by 2 arrows:
- Shoot an arrow at `x = 6`, bursting the balloons `[2,8]` and `[1,6]`.
- Shoot an arrow at `x = 11`, bursting the balloons `[10,16]` and `[7,12]`.

Example 2:
Input: `points = [[1,2],[3,4],[5,6],[7,8]]`
Output: 4
Explanation: One arrow needs to be shot for each balloon for a total of 4 arrows.

Example 3:
Input: `points = [[1,2],[2,3],[3,4],[4,5]]`
Output: 2
Explanation: The balloons can be burst by 2 arrows:
- Shoot an arrow at `x = 2`, bursting the balloons `[1,2]` and `[2,3]`.
- Shoot an arrow at `x = 4`, bursting the balloons `[3,4]` and `[4,5]`.

Accepted 50 / 50 testcases passed
Shourya Gupta submitted at Mar 06, 2025 17:38

Runtime: 107 ms | Beats 15.19%
Memory: 106.39 MB | Beats 18.51%

Code | C++

```
class Solution {
public:
    int findMinArrowShots(vector<vector<int>>> p) {
        sort(p.begin(), p.end());
        int lastpoint = p[0][1];
        int ans = 1;
        for(auto point : p) {
            if(point[0] > lastpoint) {
                ans++;
                lastpoint = point[1];
            }
        }
        return ans;
    }
}
```

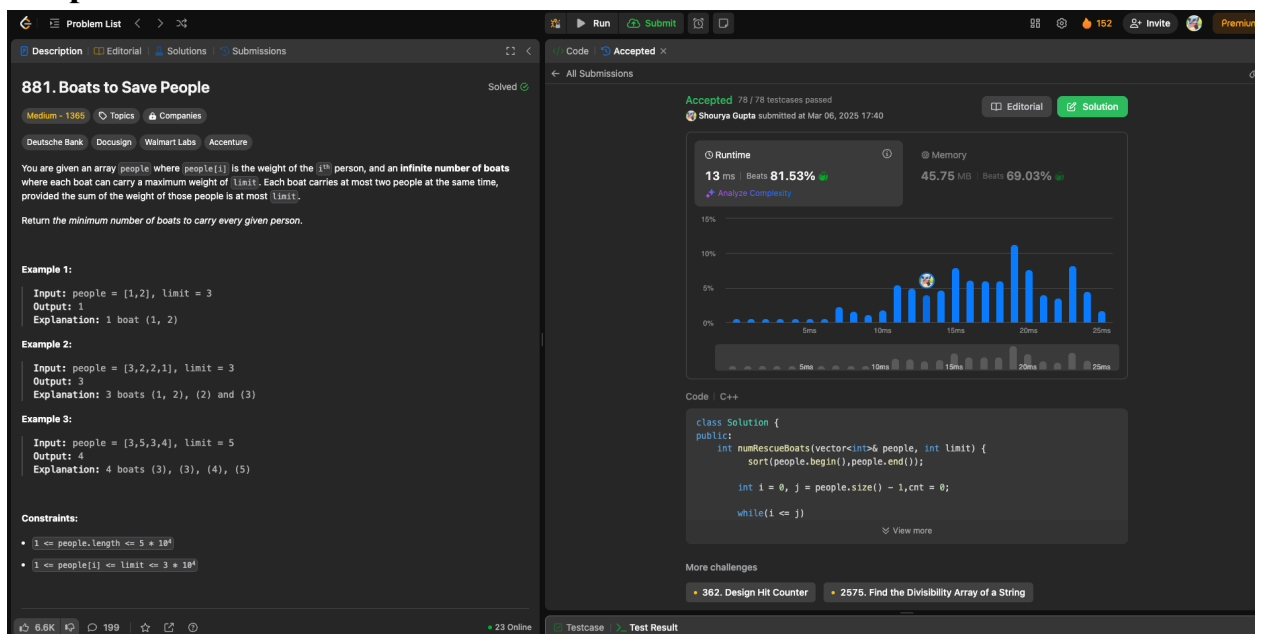
More challenges
253. Meeting Rooms II 435. Non-overlapping Intervals

1. Problem 6: Boats to Save People

2. Implementation/Code:

```
import java.util.Arrays;
class Solution {
    public int numRescueBoats(int[] people, int limit) {
        Arrays.sort(people);
        int left = 0, right = people.length - 1;
        int boats = 0;
        while (left <= right) {
            if (people[left] + people[right] <= limit) {
                left++;
            }
            right--;
            boats++;
        }
        return boats;
    }
}
```

3. Output:



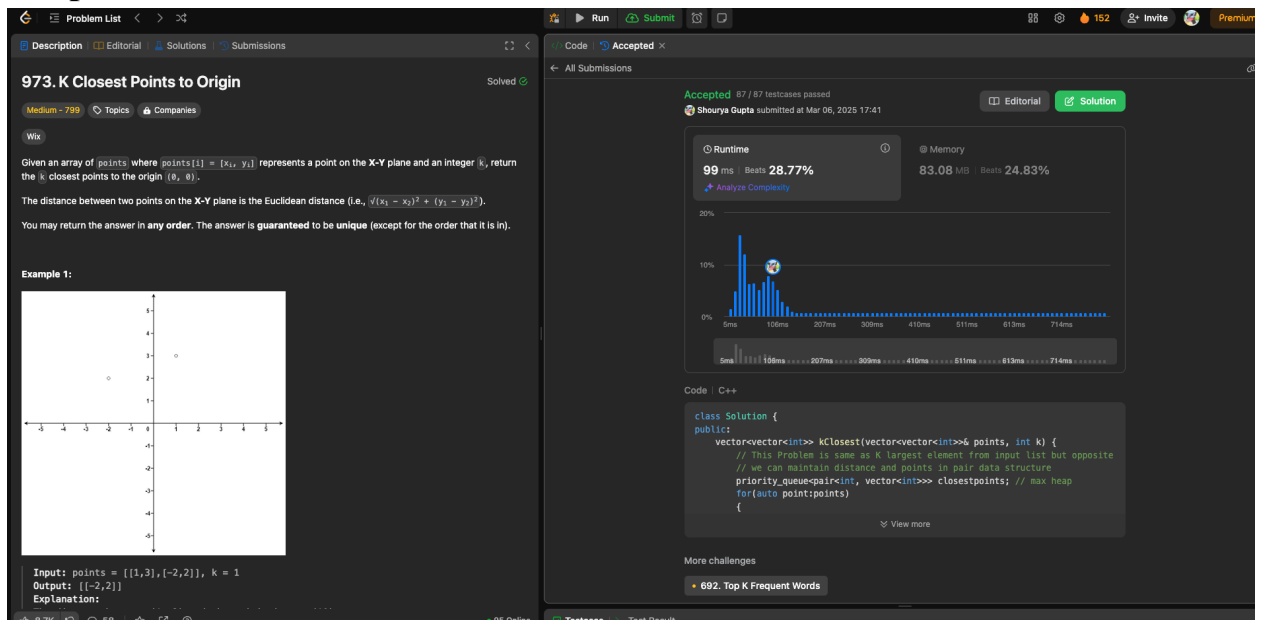
1. Problem 7: K Closest Points to Origin

2. Implementation/Code:

```
class Solution {
public int[][] kClosest(int[][] points, int k) {
    PriorityQueue<int[]> maxHeap = new PriorityQueue<>(
        (a, b) -> Integer.compare((b[0] * b[0] + b[1] * b[1]), (a[0] * a[0] +
a[1] * a[1])) );

    for (int[] point : points) {
        maxHeap.add(point);
        if (maxHeap.size() > k) {
            maxHeap.poll(); } }
    int[][] result = new int[k][2];
    for (int i = 0; i < k; i++) {
        result[i] = maxHeap.poll(); }
    return result; } }
```

3. Output:



973. K Closest Points to Origin Solved

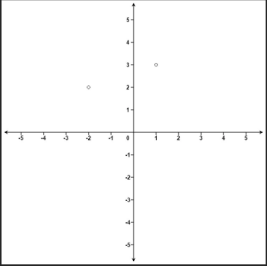
Medium - 799 Topics Companies

Given an array of points where $points[i] = [x_i, y_i]$ represents a point on the X-Y plane and an integer k , return the k closest points to the origin $(0, 0)$.

The distance between two points on the X-Y plane is the Euclidean distance (i.e., $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$).

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).

Example 1:



Input: $points = [[1, 3], [-2, 2]]$, $k = 1$
Output: $[[-2, 2]]$
Explanation:

Accepted 87 / 87 testcases passed
Shourya Gupta submitted at Mar 06, 2025 17:41

Runtime: 99 ms | Beats: 28.77% | @ Memory: 83.08 MB | Beats: 24.83%

Code C++

```
class Solution {
public:
    vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
        // This Problem is same as K largest element from input list but opposite
        // we can maintain distance and points in pair data structure
        priority_queue<pair<int, vector<int>>> closestpoints; // max heap
        for(auto point:points)
        {
```

More challenges
692. Top K Frequent Words

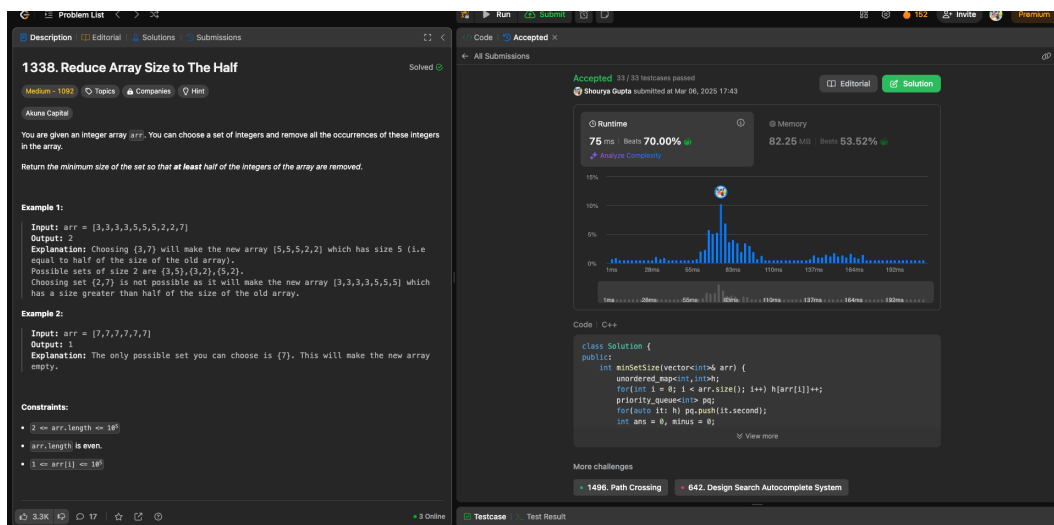
1. Problem 8: Reduce Array Size to The Half

2. Implementation/Code:

```
import java.util.*;

class Solution {
    public int minSetSize(int[] arr) {
        Map<Integer, Integer> freq = new HashMap<>();
        for (int num : arr) freq.put(num, freq.getOrDefault(num, 0) + 1);
        List<Integer> counts = new ArrayList<>(freq.values());
        counts.sort(Collections.reverseOrder());
        int res = 0, cnt = 0, half = arr.length / 2;
        for (int num : counts) {
            cnt += num;
            res++;
            if (cnt >= half) break;
        }
        return res;
    }
}
```

3. Output:



1338. Reduce Array Size to The Half

Medium - 1992 Topics Companies Hint

Alanna Capital

You are given an integer array `arr`. You can choose a set of integers and remove all the occurrences of these integers in the array.

Return the minimum size of the set so that at least half of the integers of the array are removed.

Example 1:

Input: `arr = [3,3,3,3,5,5,2,2,7]`
Output: 2
Explanation: Choosing `{3,7}` will make the new array `[5,5,2,2]` which has size 4 (i.e. equal to half of the size of the old array).
Possible sets of size 2 are `{3,5}`, `{3,2}`, `{5,2}`.
Choosing set `{2,7}` is not possible as it will make the new array `[3,3,3,5,5]` which has a size greater than half of the size of the old array.

Example 2:

Input: `arr = [7,7,7,7,7]`
Output: 1
Explanation: The only possible set you can choose is `{7}`. This will make the new array empty.

Constraints:

- `2 <= arr.length <= 105`
- `arr.length` is even.
- `1 <= arr[i] <= 105`

Accepted 33 / 33 testcases passed
Shourya Gupta submitted at Mar 06, 2025 17:43

Runtime 75 ms Beats 70.00% **Memory** 82.25 MB Beats 53.52%

Code C++

```
class Solution {
public:
    int minSetSize(vector<int>& arr) {
        unordered_map<int, int> h;
        for(int i = 0; i < arr.size(); i++) h[arr[i]]++;
        priority_queue<int> pq;
        for(auto it: h) pq.push(it.second);
        int ans = 0, minus = 0;
    }
```

Testcase **Test Result**

1496. Path Crossing 642. Design Search Autocomplete System