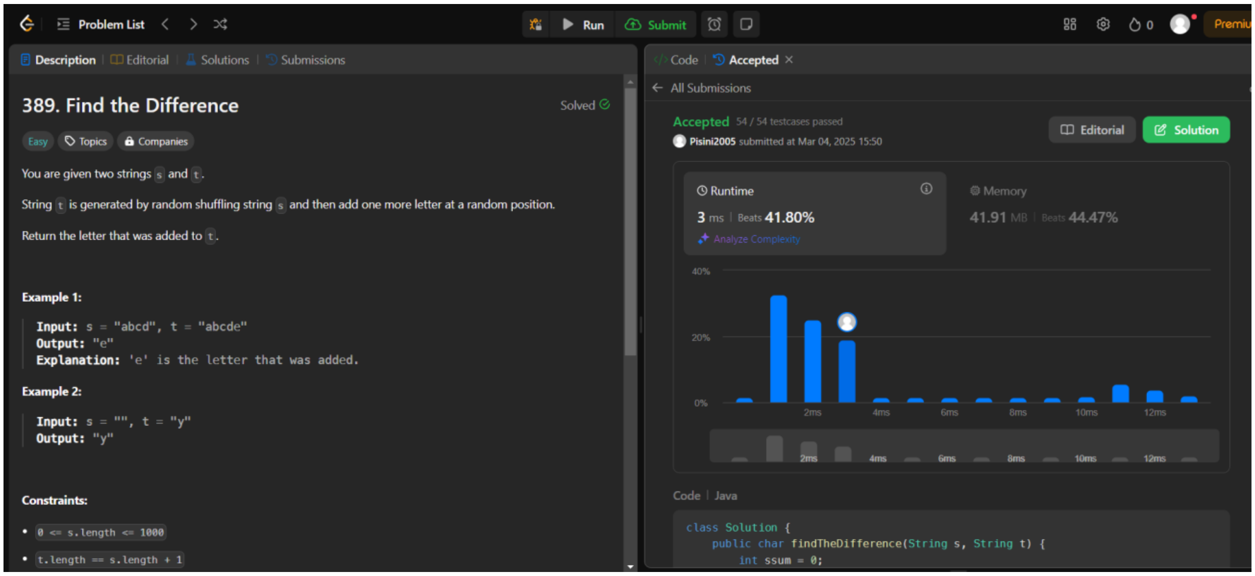
ASSIGNMENT -5 (ADVANCED PROGRAMMING)

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:**



389. Find the Difference

Solved

You are given two strings `s` and `t`.

String `t` is generated by random shuffling string `s` and then add one more letter at a random position.

Return the letter that was added to `t`.

Example 1:

```
Input: s = "abcd", t = "abcde"
Output: "e"
Explanation: 'e' is the letter that was added.
```

Example 2:

```
Input: s = "", t = "y"
Output: "y"
```

Constraints:

- `0 <= s.length <= 1000`
- `t.length == s.length + 1`

Accepted 54 / 54 testcases passed

Pisini2005 submitted at Mar 04, 2025 15:50

Runtime

3 ms | Beats 41.80%

Memory

41.91 MB | Beats 44.47%

Code | Java

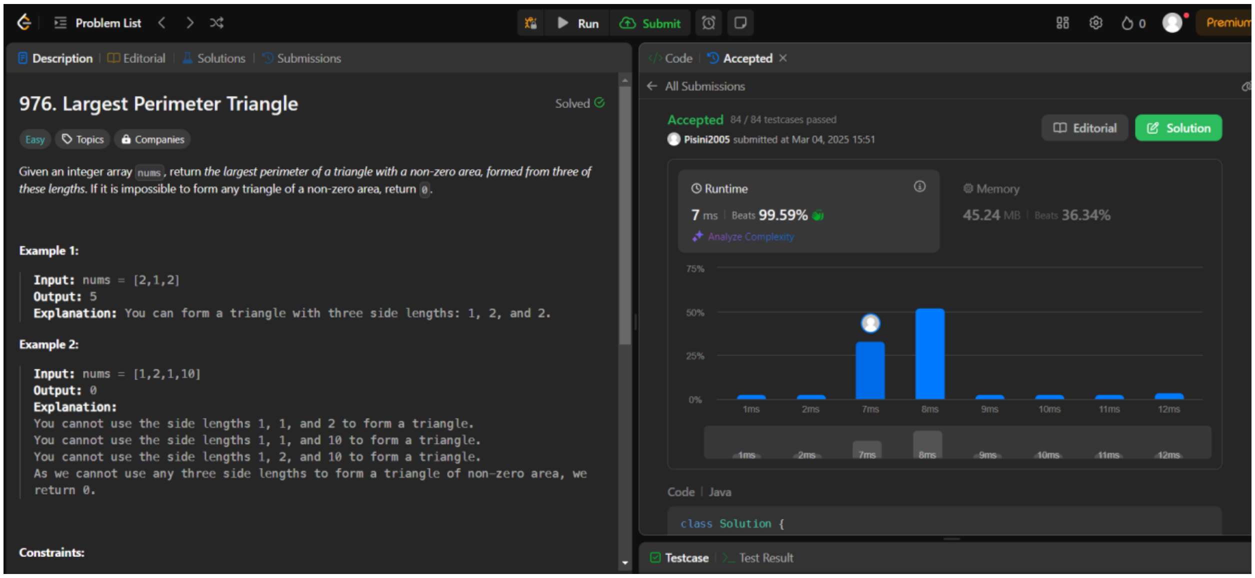
```
class Solution {
    public char findTheDifference(String s, String t) {
        int ssum = 0;
```

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:



976. Largest Perimeter Triangle

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:

Accepted 84 / 84 testcases passed

Pisini2005 submitted at Mar 04, 2025 15:51

Runtime 7 ms | Beats 99.59%

Memory 45.24 MB | Beats 36.34%

Code | Java

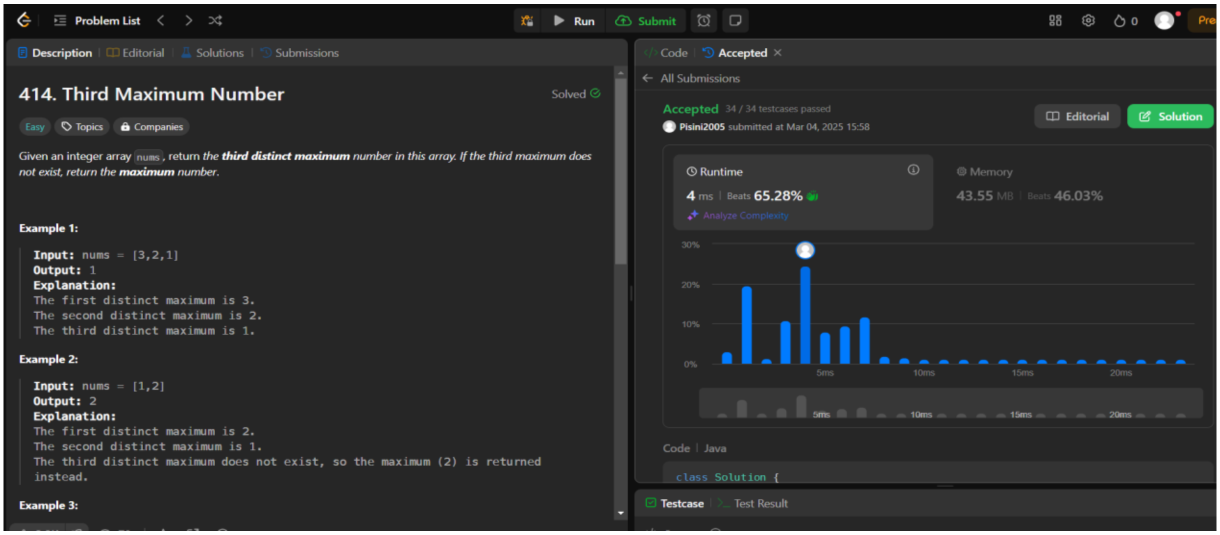
```
class Solution {
```

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:



414. Third Maximum Number

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:

Accepted 34 / 34 testcases passed

Pisini2005 submitted at Mar 04, 2025 15:58

Runtime
4 ms | Beats 65.28%

Memory
43.55 MB | Beats 46.03%

Code | Java

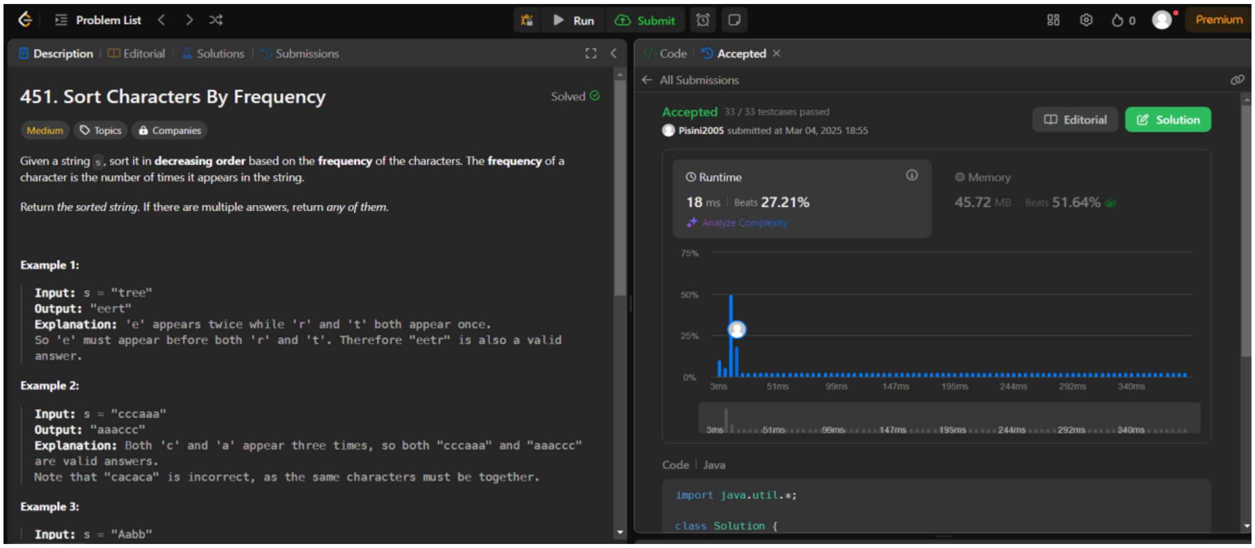
```
class Solution {
```

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:



451. Sort Characters By Frequency

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 "eetr" is also 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"
```

Accepted 33 / 33 testcases passed

Pisini2005 submitted at Mar 04, 2025 18:55

Runtime 18 ms | Beats 27.21%

Memory 45.72 MB | Beats 51.64%

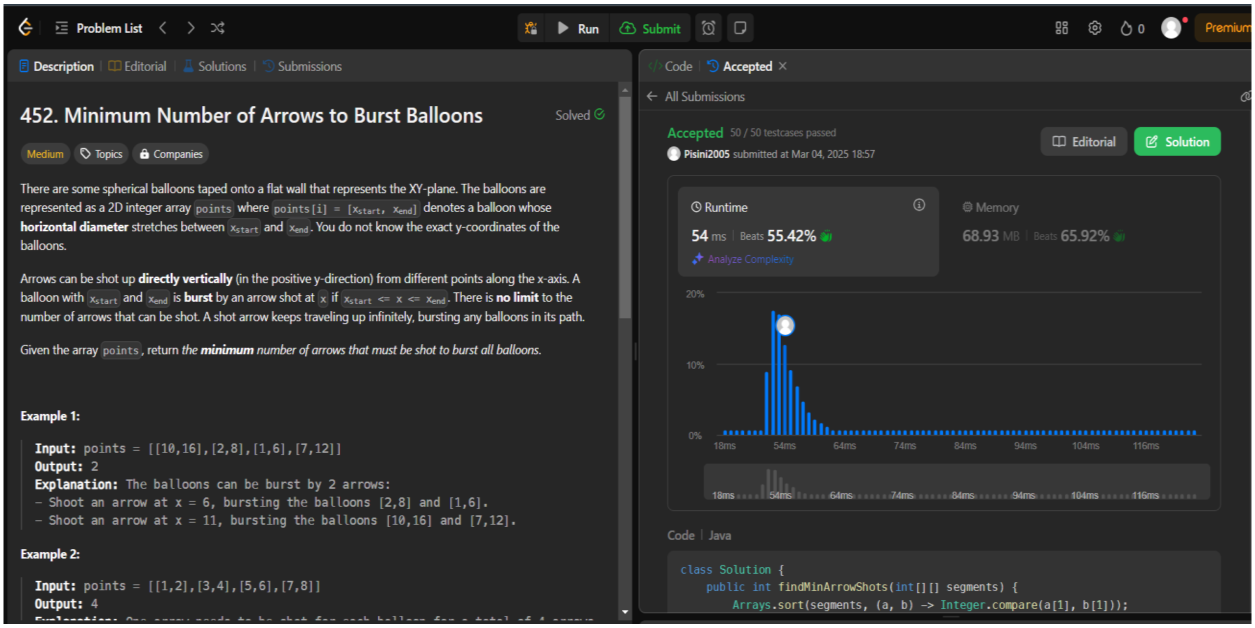
```
import java.util.*;

class Solution {
```

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

2. **Implementation/Code:**

```
class Solution {
    public int findMinArrowShots(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

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
```

Accepted 50 / 50 testcases passed

Pisini2005 submitted at Mar 04, 2025 18:57

54 ms | Beats 55.42%

68.93 MB | Beats 65.92%

Code | Java

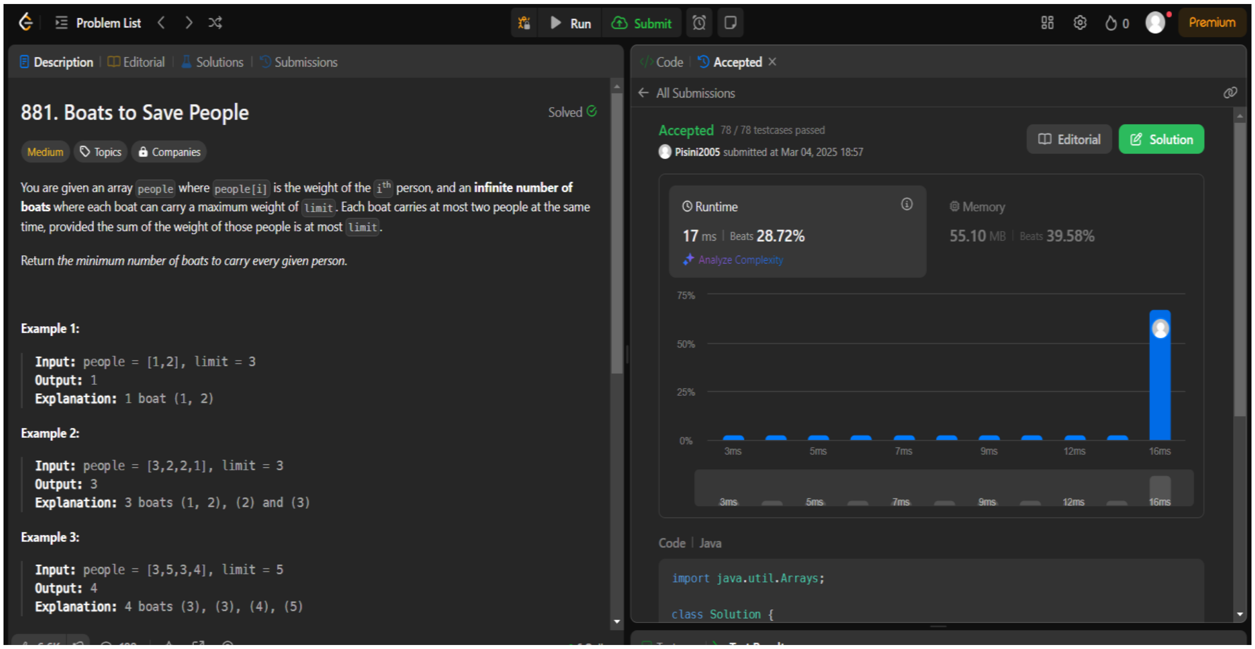
```
class Solution {
    public int findMinArrowShots(int[][] segments) {
        Arrays.sort(segments, (a, b) -> Integer.compare(a[1], b[1]));
```

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:**



881. Boats to Save People

You are given an array `people` where `people[i]` is the weight of the i^{th} person, and an **infinite number of boats** where each boat can carry a maximum weight of `limit`. Each boat carries at most two people at the same time, provided the sum of the weight of those people is at most `limit`.

Return *the minimum number of boats to carry every given person.*

Example 1:

```
Input: people = [1,2], limit = 3
Output: 1
Explanation: 1 boat (1, 2)
```

Example 2:

```
Input: people = [3,2,2,1], limit = 3
Output: 3
Explanation: 3 boats (1, 2), (2) and (3)
```

Example 3:

```
Input: people = [3,5,3,4], limit = 5
Output: 4
Explanation: 4 boats (3), (3), (4), (5)
```

Accepted 78 / 78 testcases passed

Pisini2005 submitted at Mar 04, 2025 18:57

Runtime
17 ms | Beats 28.72%

Memory
55.10 MB | Beats 39.58%

Code | Java

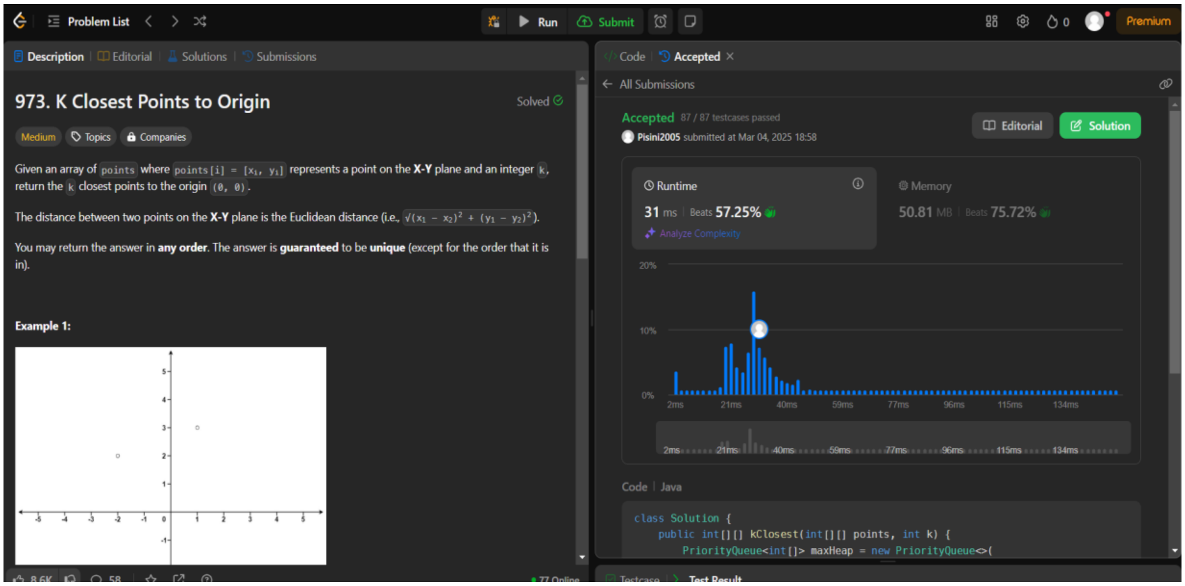
```
import java.util.Arrays;

class Solution {
```

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 · Topics · Companies

Given an array of `points` where `points[i] = [xᵢ, yᵢ]` 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:

Accepted 87 / 87 testcases passed

Pisini2005 submitted at Mar 04, 2025 18:58

Runtime

31 ms | Beats 57.25%

Memory

50.81 MB | Beats 75.72%

Code | Java

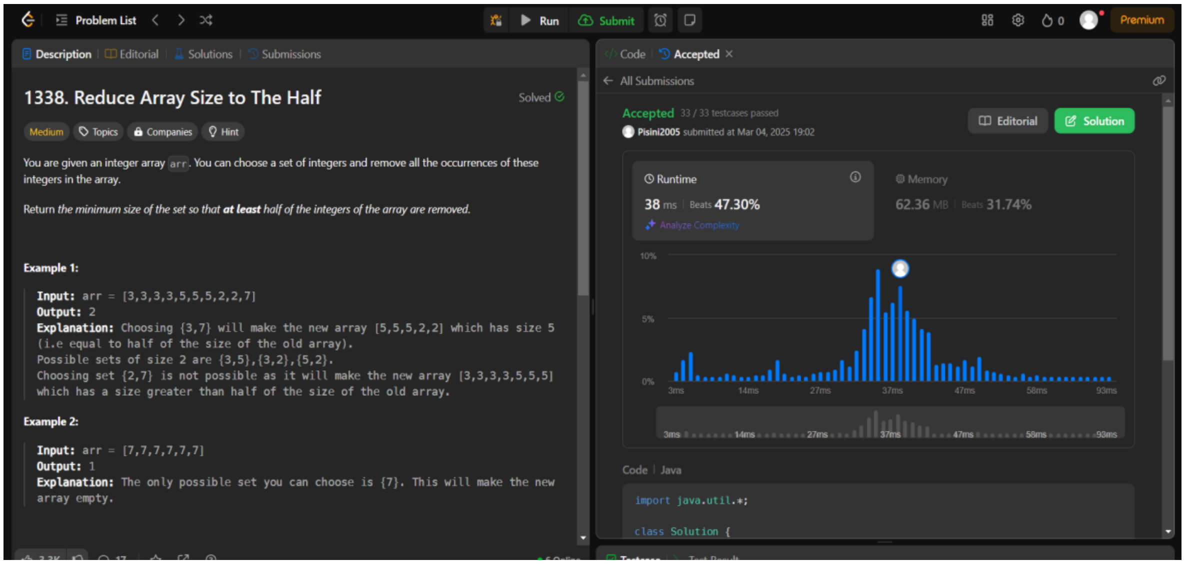
```
class Solution {
    public int[][] kClosest(int[][] points, int k) {
        PriorityQueue<int[]> maxHeap = new PriorityQueue<>(
```

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

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 = [3,3,3,3,5,5,5,2,2,7]
Output: 2
Explanation: Choosing {3,7} will make the new array [5,5,5,2,2] which has size 5
(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 [3,3,3,3,5,5,5]
which has a size greater than half of the size of the old array.
```

Example 2:

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

Accepted 33 / 33 testcases passed
Pisini2005 submitted at Mar 04, 2025 19:02

Runtime
38 ms | Beats 47.30%

Memory
62.36 MB | Beats 31.74%

Code | Java

```
import java.util.*;

class Solution {
```