**Assignment -5**

**Name- Gunjan Mahur**

**UID-22BCS10484**

**Q Find the Difference**

**Code:**

class Solution {

public char findTheDifference(String s, String t) {

int[] freq = new int[26];

for (char ch : s.toCharArray()) {

freq[ch - 'a']++;

}

for (char ch : t.toCharArray()) {

freq[ch - 'a']--;

if (freq[ch - 'a'] < 0) {

return ch;

}

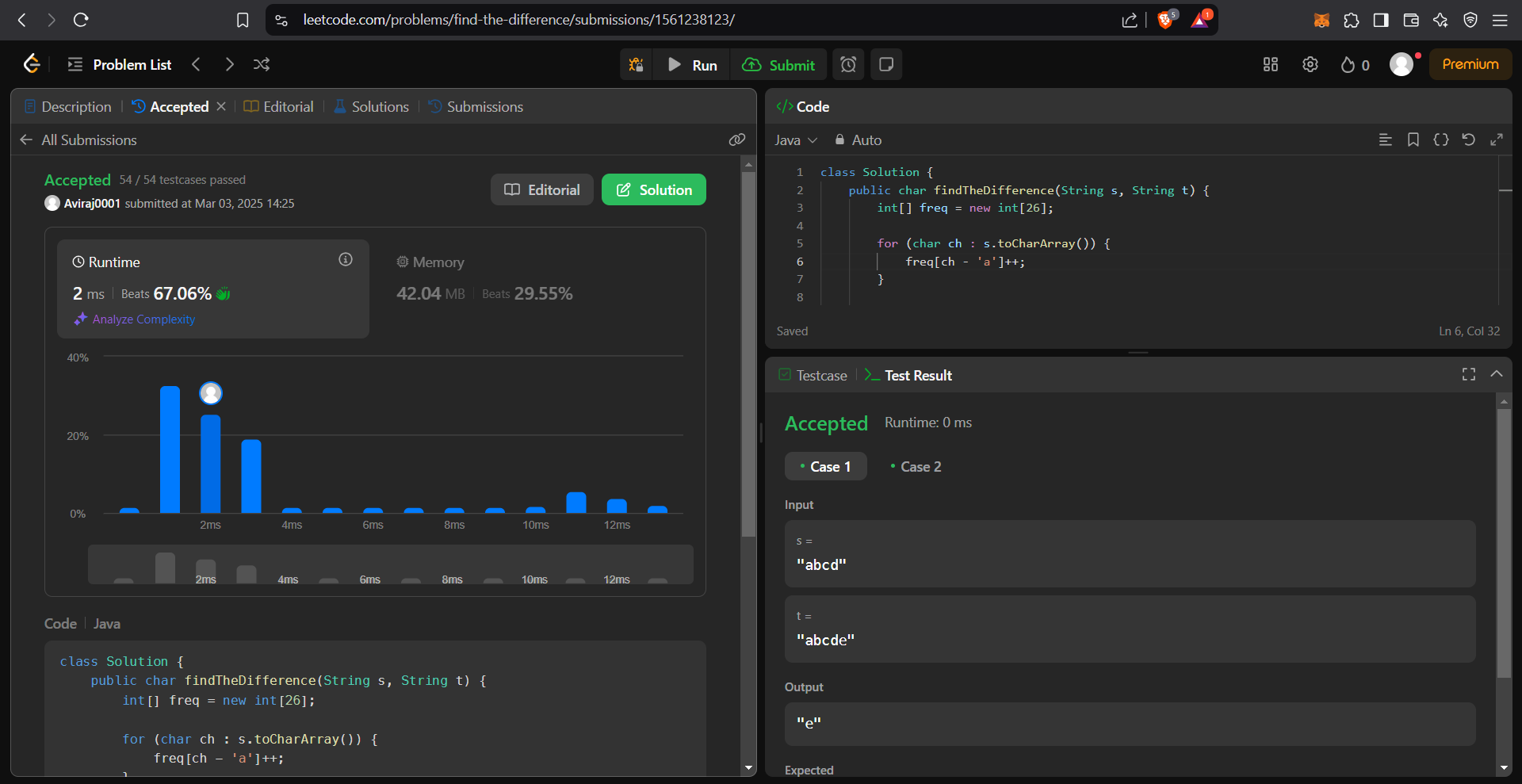
}

return ' ';

}

}

Output:



**Q Third Maximum Number**

Code:

import java.util.Arrays;

class Solution {

public int thirdMax(int[] nums) {

Arrays.sort(nums);

int count = 0;

int n = nums.length;

for (int i = n - 1; i > 0; i--) {

if (nums[i] != nums[i - 1]) {

count++;

}

if (count == 2) return nums[i - 1];

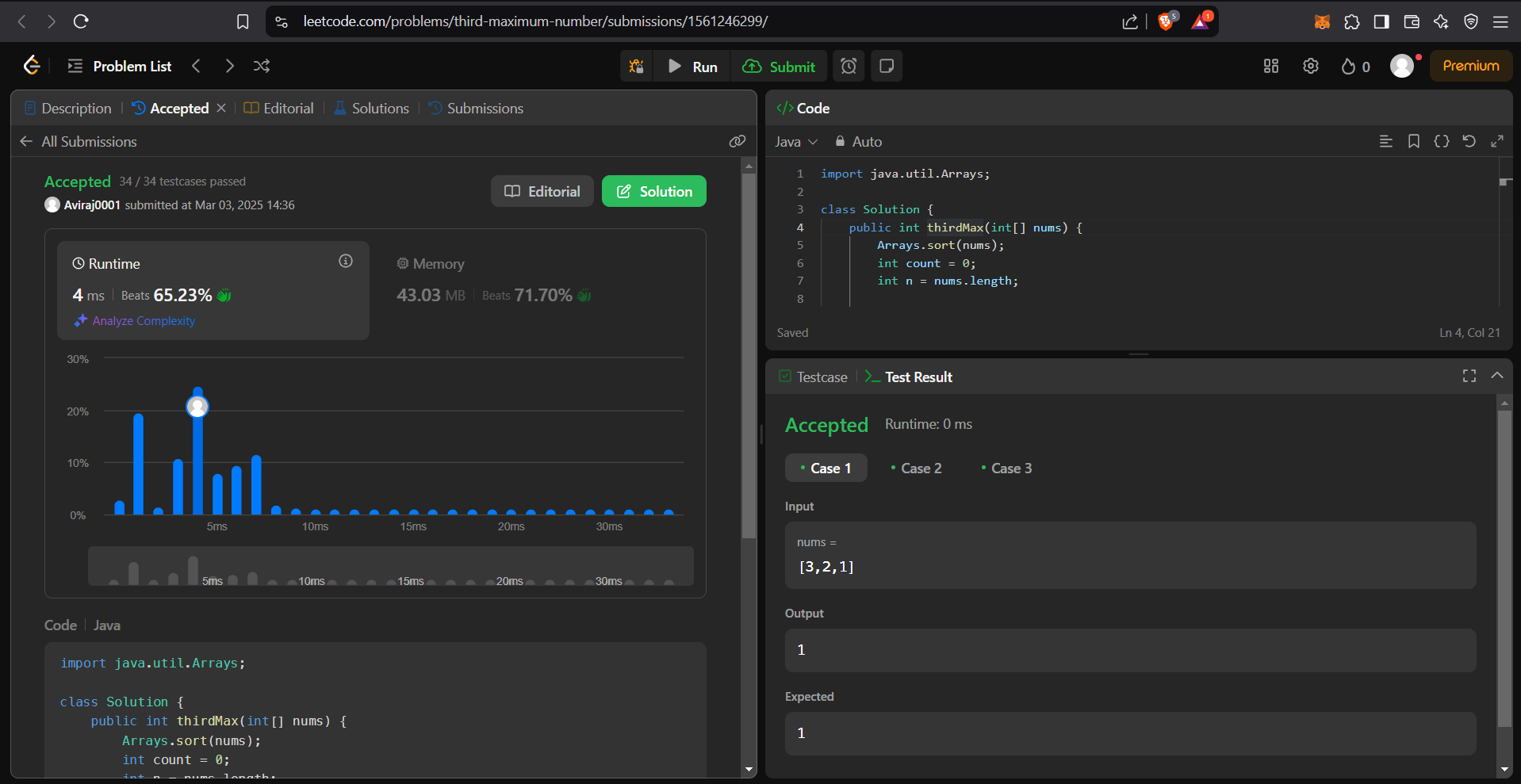
}

return nums[n - 1];

}

}

Output:



**Q Largest Perimeter Triangle.**

Code:

import java.util.Arrays;

class Solution {

public int largestPerimeter(int[] nums) {

Arrays.sort(nums);

for (int i = nums.length - 1; i >= 2; i--) {

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

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

}

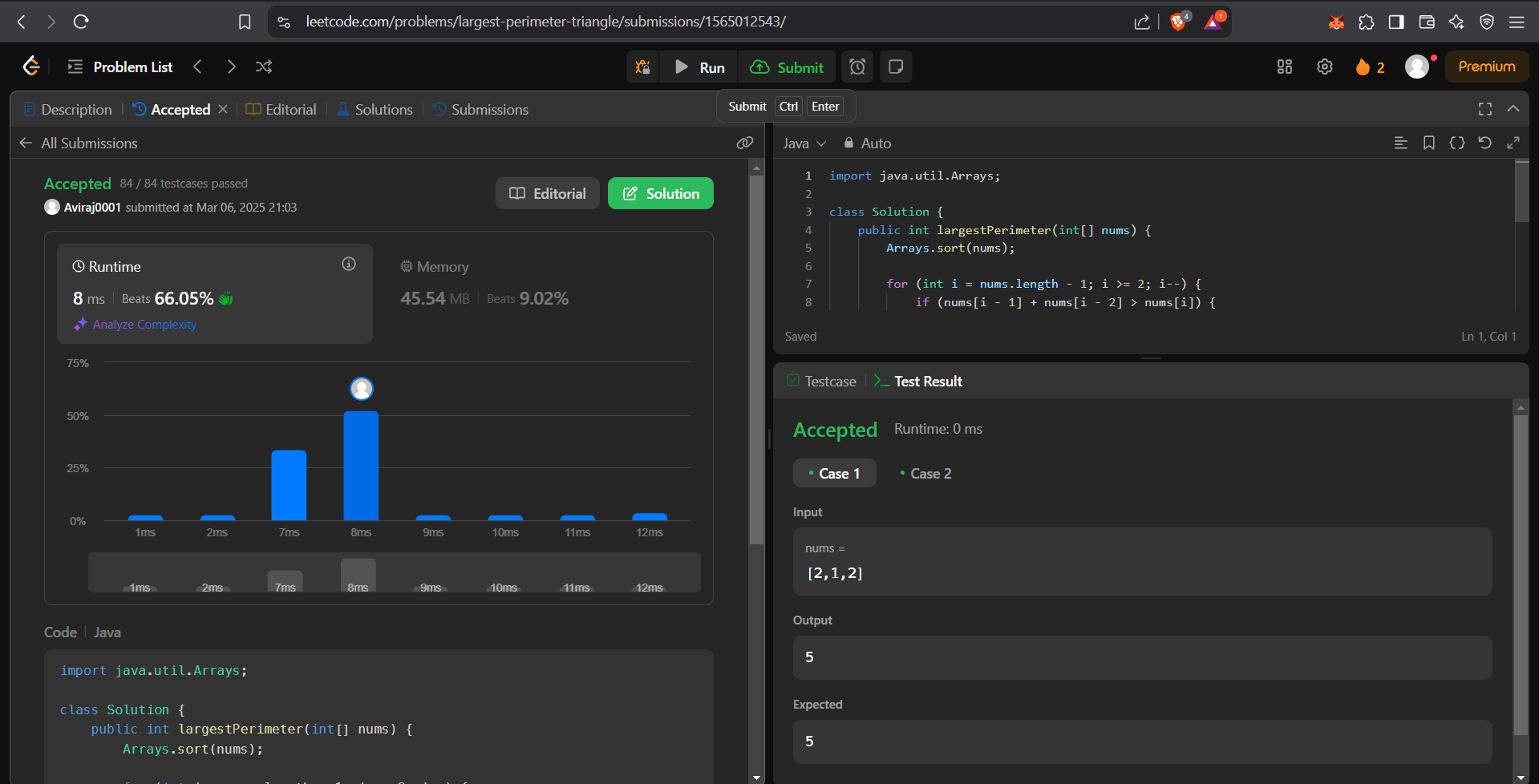
}

return 0;

}

}

Output:



**Q Sort Characters By Frequency**

Code:

class Solution

{

public String frequencySort(String s)

{

StringBuilder sb = new StringBuilder();

int[] freq = new int[128];

for(char c : s.toCharArray())

{

freq[c-'0']++;

}

for(int k = 0; k < s.length(); k++)

{

int max = 0;

int ind = 0;

for(int j = 0; j < freq.length; j++)

{

if(freq[j] != 0 && freq[j] > max)

{

max = freq[j];

ind = j;

}

}

for(int i = 0; i < max; i++)

{

sb.append((char)('0'+ind));

}

freq[ind] = 0;

k += --max;

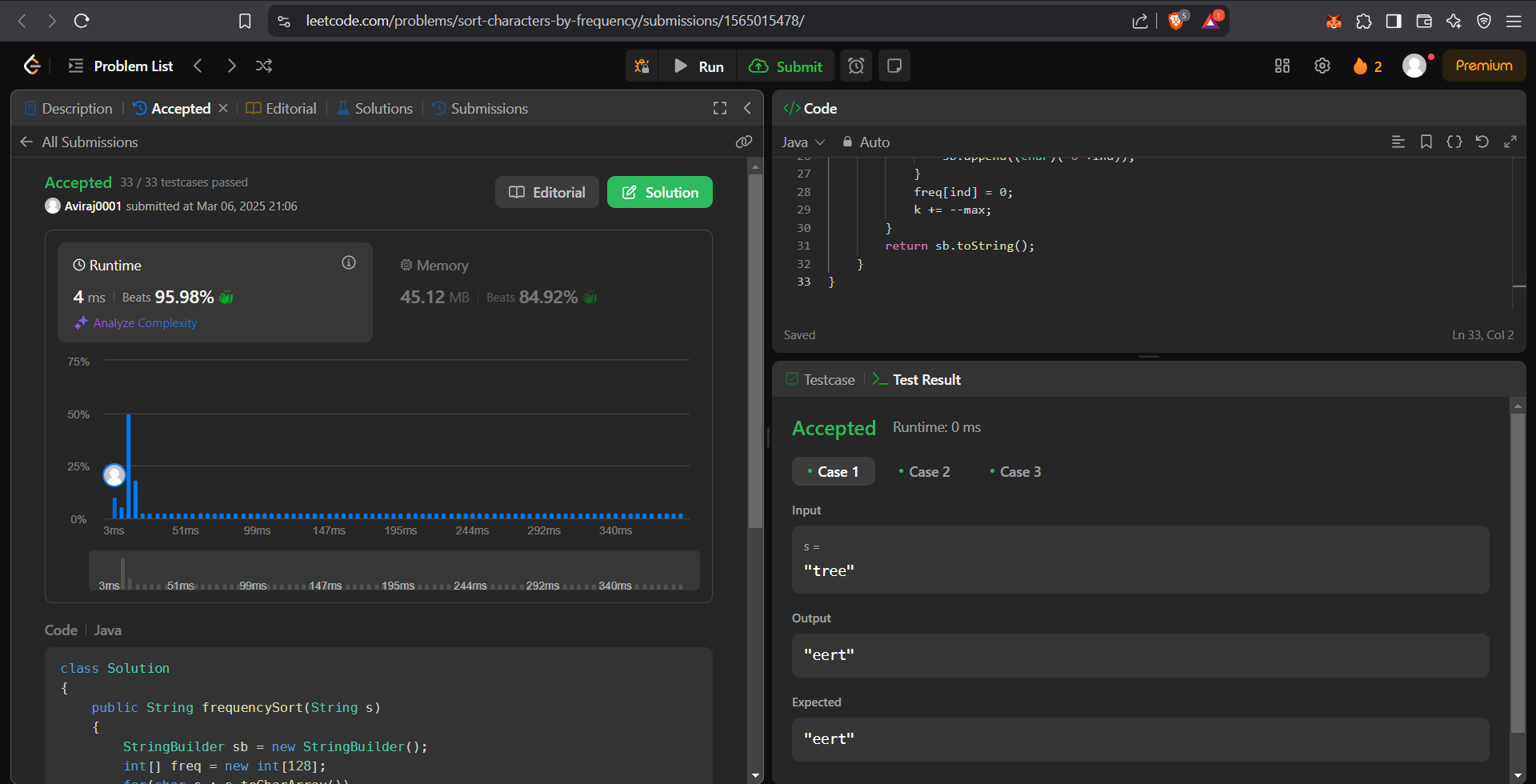
}

return sb.toString();

}

}

Output:



**Q Minimum Number of Arrows to Burst Balloons**

Code:

class Solution {

public int findMinArrowShots(int[][] points) {

// Sort the balloons based on their end coordinates

Arrays.sort(points, (a, b) -> Integer.compare(a[1], b[1]));

int arrows = 1;

int prevEnd = points[0][1];

// Count the number of non-overlapping intervals

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

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

arrows++;

prevEnd = points[i][1];

}

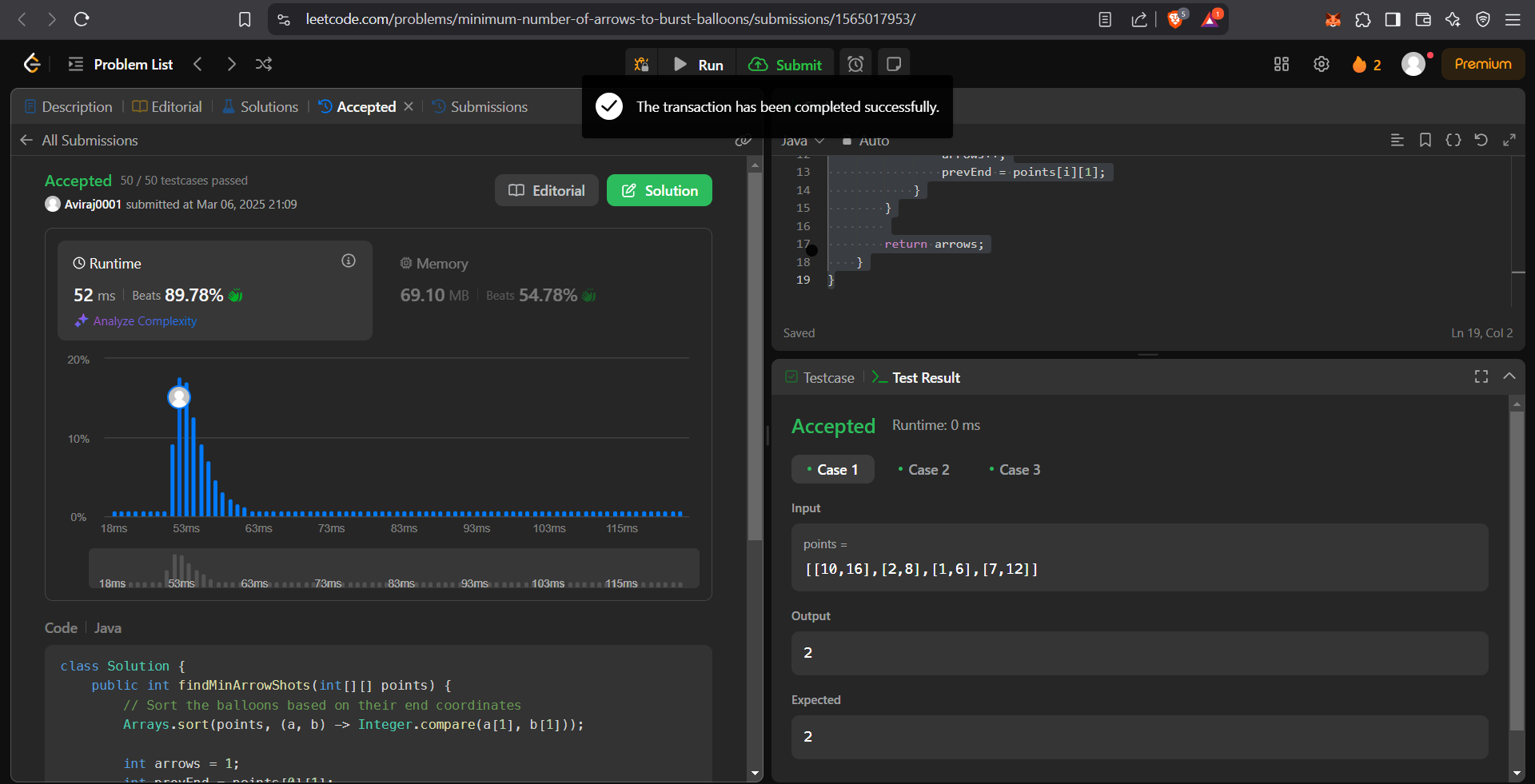
}

return arrows;

}

}

Output:



**Q Boats to Save People**

Code:

class Solution {

public int numRescueBoats(int[] people, int limit) {

int boats = 0;

Arrays.sort(people);

int i=0,j=people.length-1;

while(i<=j){

if((people[j]+people[i])<=limit){

i++;

}

j--;

boats++;

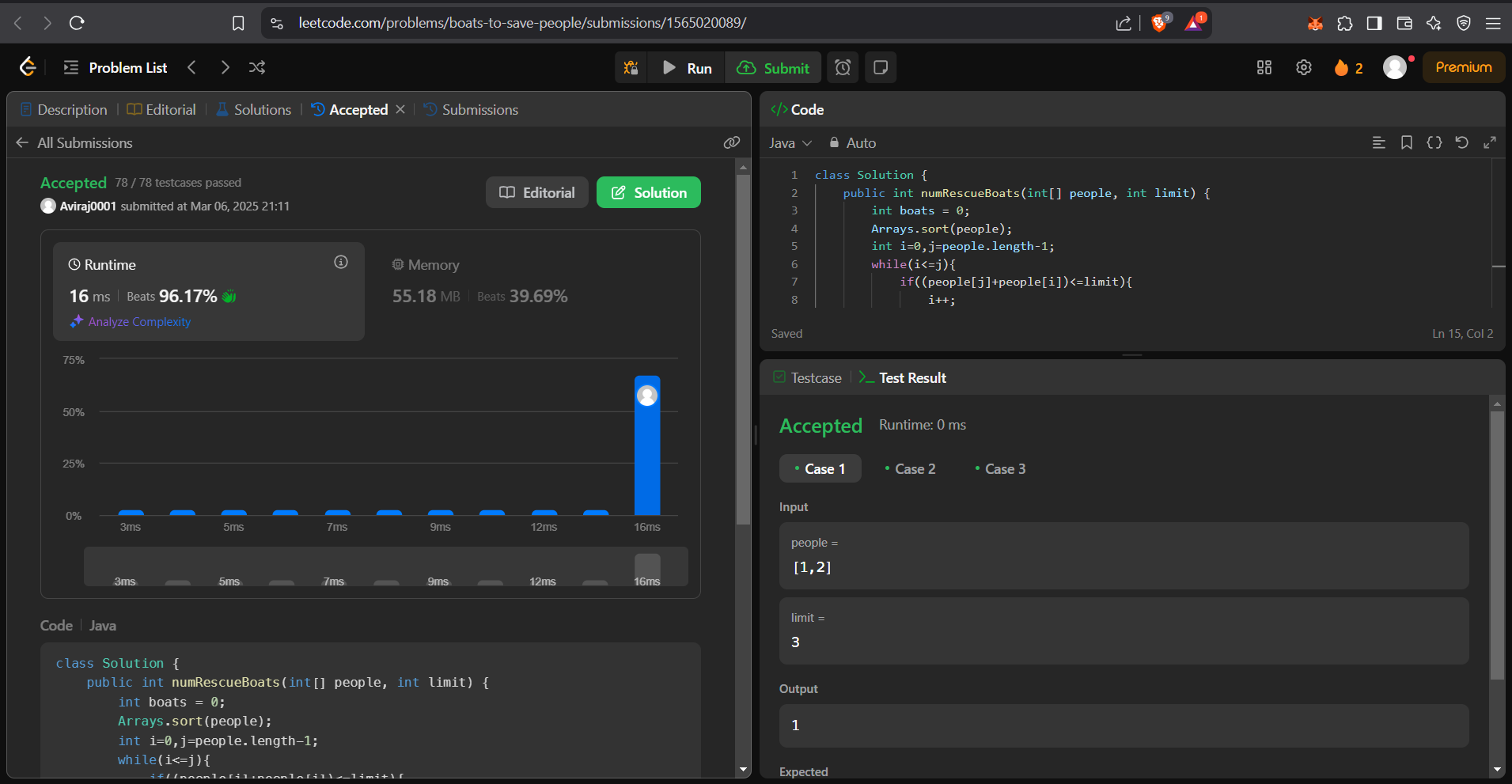
}

return boats;

}

}

Output:



**Q K Closest Points to Origin**

Code:

class Solution {

static class Point implements Comparable<Point> {

int x;

int y;

int disSq;

int idx;

public Point(int x, int y, int disSq, int idx) {

this.x = x;

this.y = y;

this.disSq = disSq;

this.idx = idx;

}

@Override

public int compareTo(Point p2) {

return this.disSq - p2.disSq; // Ascending order

}

}

public int[][] kClosest(int[][] points, int k) {

PriorityQueue<Point> pq = new PriorityQueue<>();

for (int i = 0; i < points.length; i++) {

int disSq = points[i][0] \* points[i][0] + points[i][1] \* points[i][1];

pq.add(new Point(points[i][0], points[i][1], disSq, i));

}

int[][] result = new int[k][2];

for (int i = 0; i < k; i++) {

Point p = pq.remove();

result[i][0] = p.x;

result[i][1] = p.y;

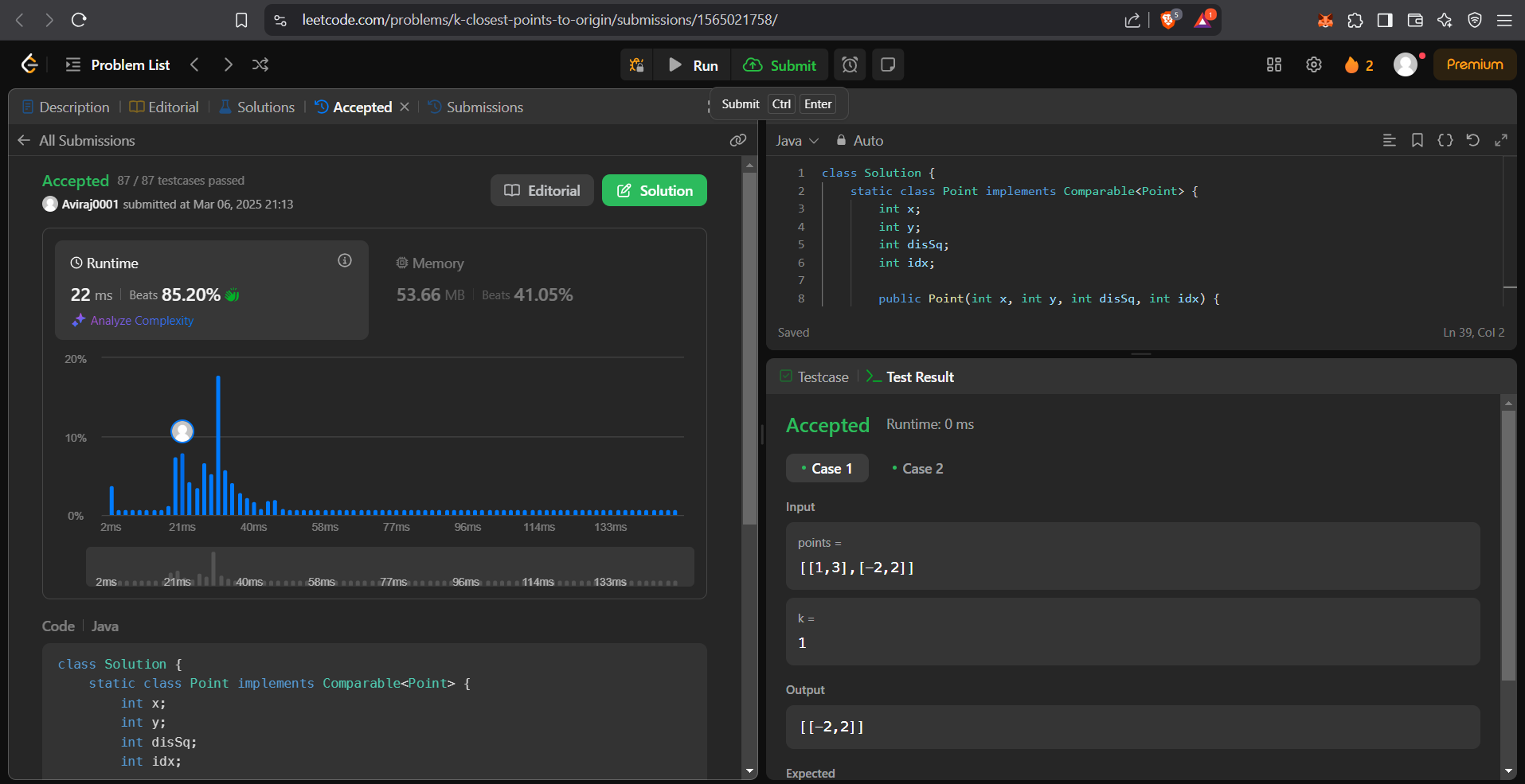
}

return result;

}

}

Output:



**Q Reduce Array Size to The Half**

Code:

class Solution {

public int minSetSize(int[] arr) {

HashMap<Integer, Integer> cnt = new HashMap<>();

for (int x : arr) cnt.put(x, cnt.getOrDefault(x, 0) + 1);

int[] frequencies = new int[cnt.values().size()];

int i = 0;

for (int freq : cnt.values()) frequencies[i++] = freq;

Arrays.sort(frequencies);

int ans = 0, removed = 0, half = arr.length / 2;

i = frequencies.length - 1;

while (removed < half) {

ans += 1;

removed += frequencies[i--];

}

return ans;

}

}

Output:

