**Name:** Aditya Singh

**UID:** 22BCS15063

**Section:** FL\_IOT\_601 - A

**Assignment – 5 Solutions:-**

1. **Find the Difference:**

class Solution {

public:

char findTheDifference(string s, string t) {

vector<char> s1(26,0);

for(int i=0;i<s.length();i++) s1[s[i]-'a']++;

for(int i=0;i<t.length();i++) s1[t[i]-'a']++;

for(int i=0;i<s1.size();i++){

if(s1[i]%2) return char('a'+i);

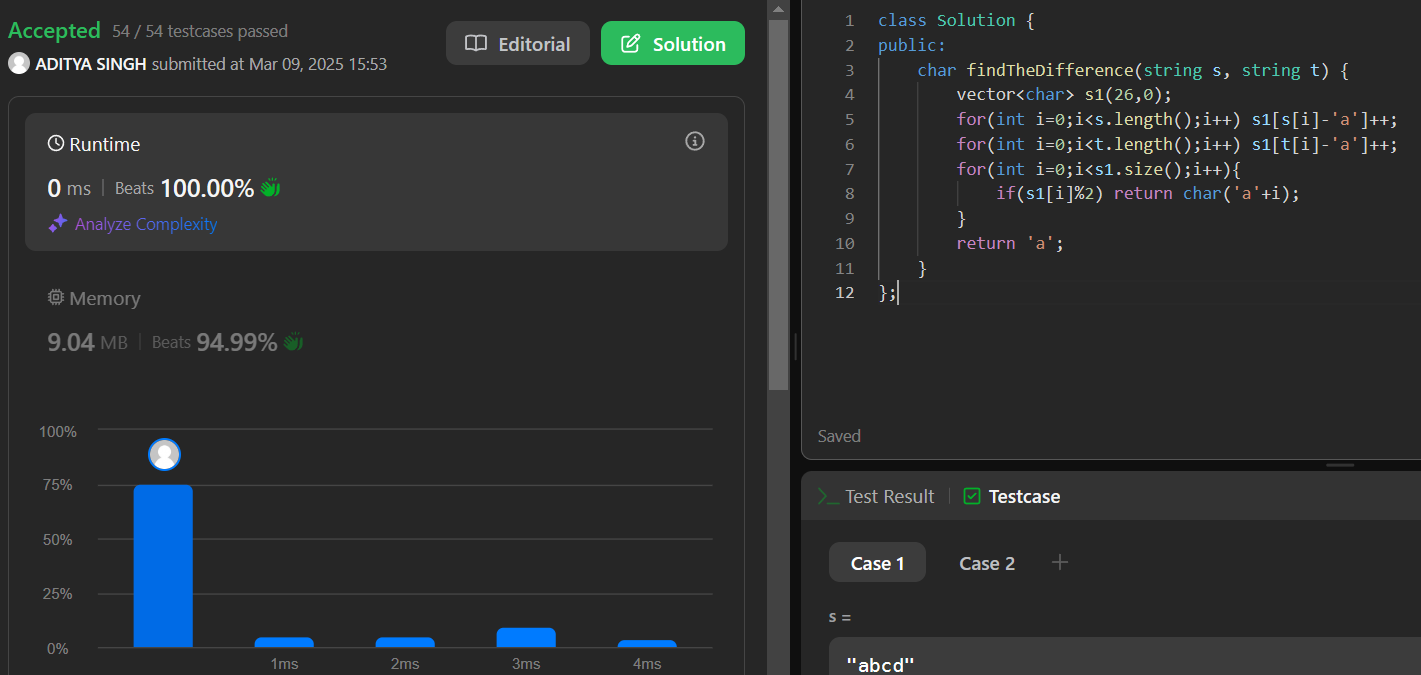
}

return 'a';

}

};

Result:-



1. **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];

}

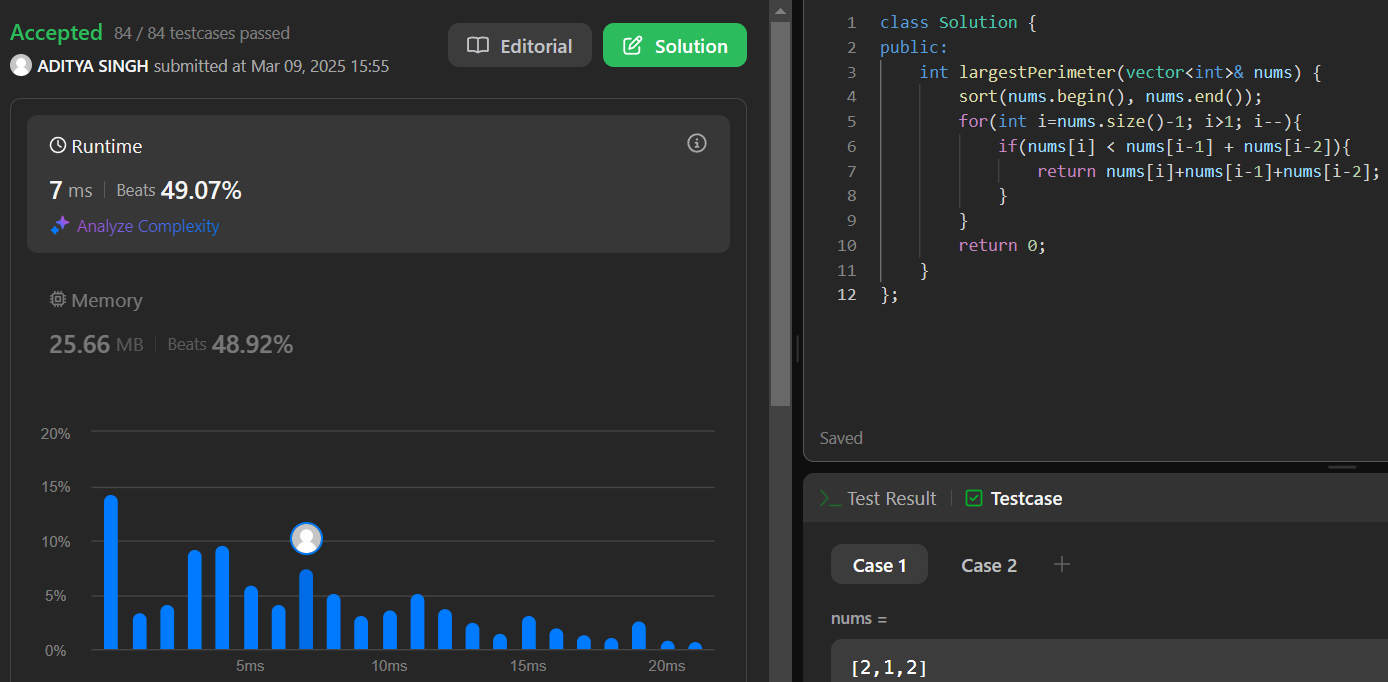
}

return 0;

}

};

Result:



1. **Third Maximum Number:**

class Solution {

public int thirdMax(int[] nums) {

long max1 = Long.MIN\_VALUE;

long max2 = Long.MIN\_VALUE;

long max3 = Long.MIN\_VALUE;

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

if (nums[i] > max1) {

max3 = max2;

max2 = max1;

max1 = nums[i];

} else if (nums[i] > max2 && nums[i] != max1) {

max3 = max2;

max2 = nums[i];

} else if (nums[i] > max3 && nums[i] != max1 && nums[i] != max2) {

max3 = nums[i];

}

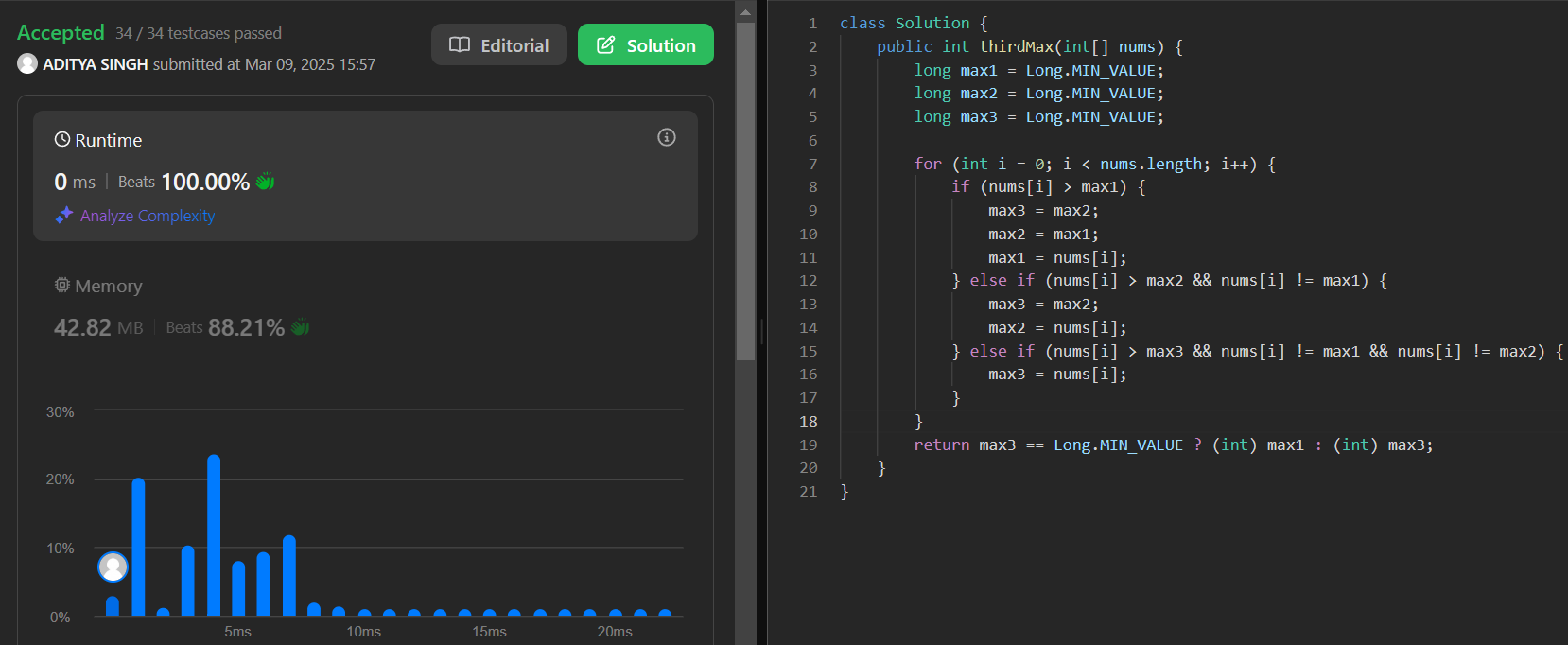
}

return max3 == Long.MIN\_VALUE ? (int) max1 : (int) max3;

}

}

Result:



1. **Maximum Subarray:**

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;

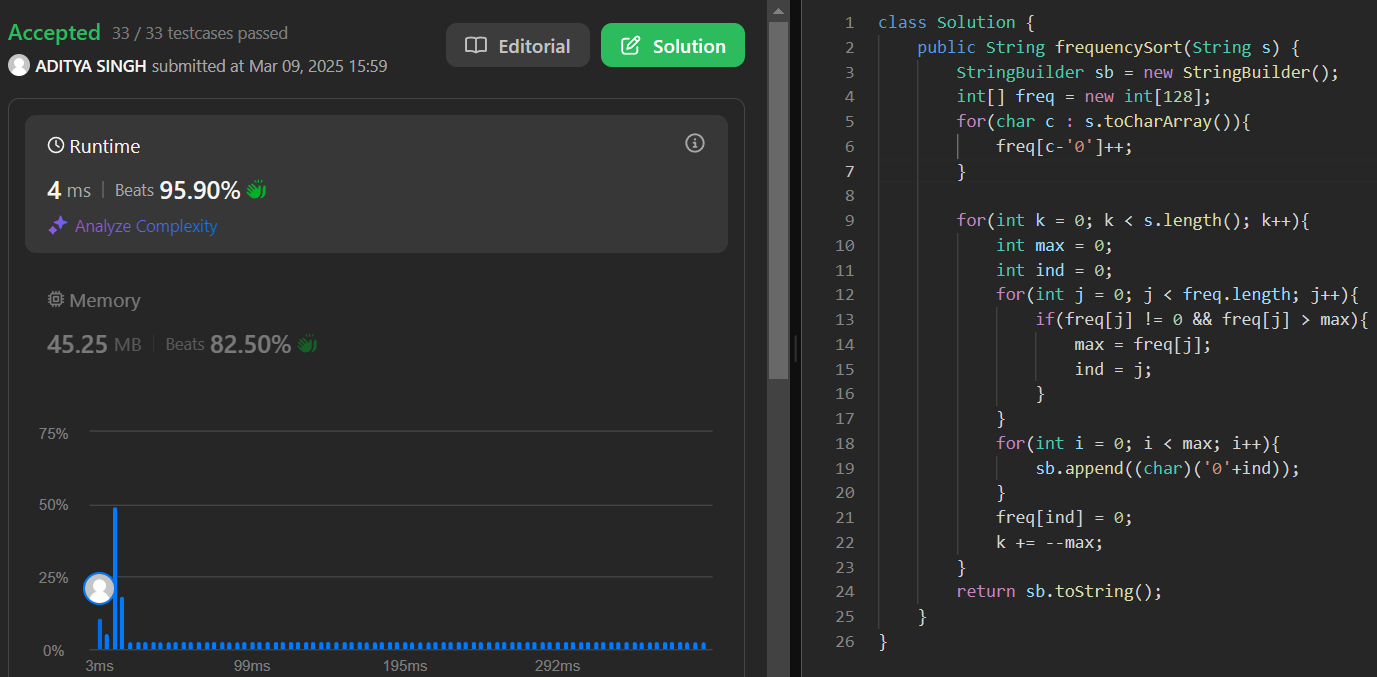
}

return sb.toString();

}

}

Result:



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

class Solution {

public:

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

int n = points.size();

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

int cnt = 1;

int x = points[0][0], y = points[0][1];

for(int i = 1; i < n; i++){

if((points[i][0] >= x && points[i][0] <= y) ||

(points[i][1] >= x && points[i][1] <= y)){

x = max(x, points[i][0]);

y = min(y, points[i][1]);

}

else{

x = points[i][0];

y = points[i][1];

cnt++;

}

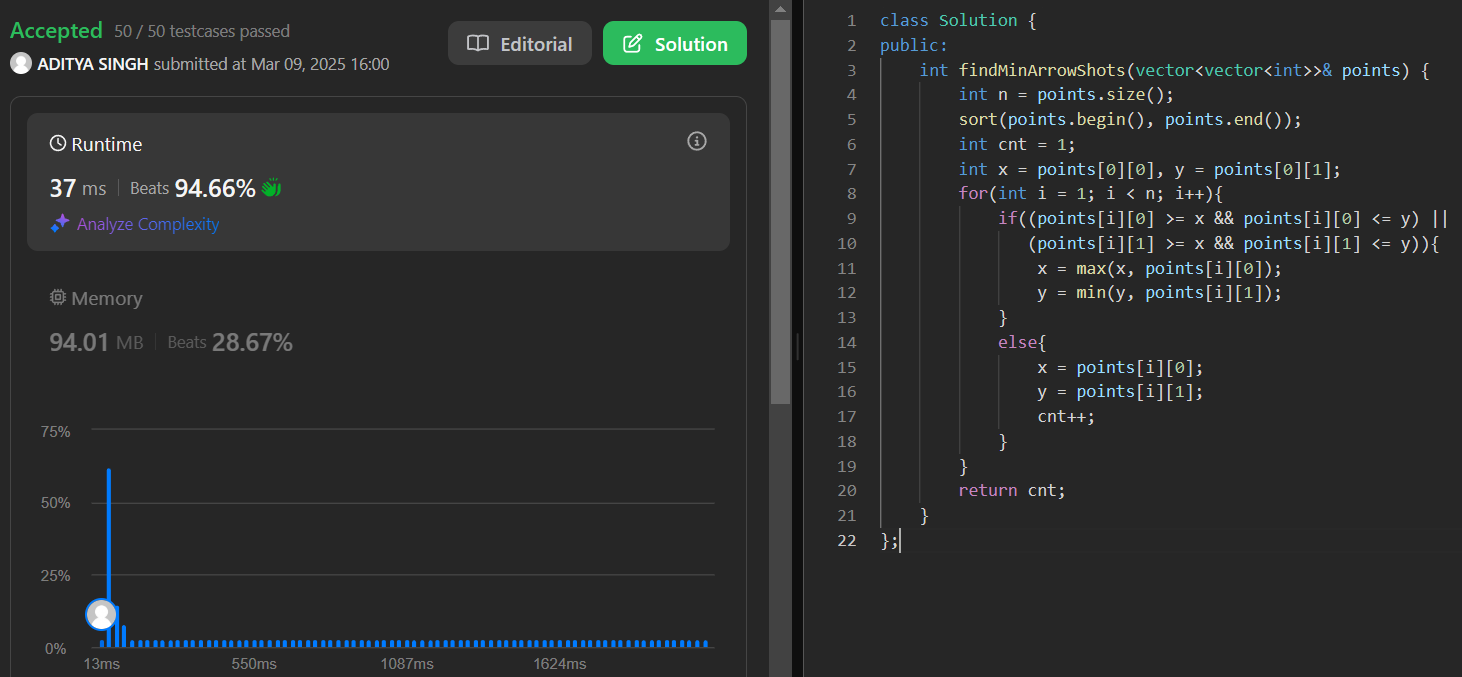
}

return cnt;

}

};

Result:



1. **Boats to Save People:**

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++;

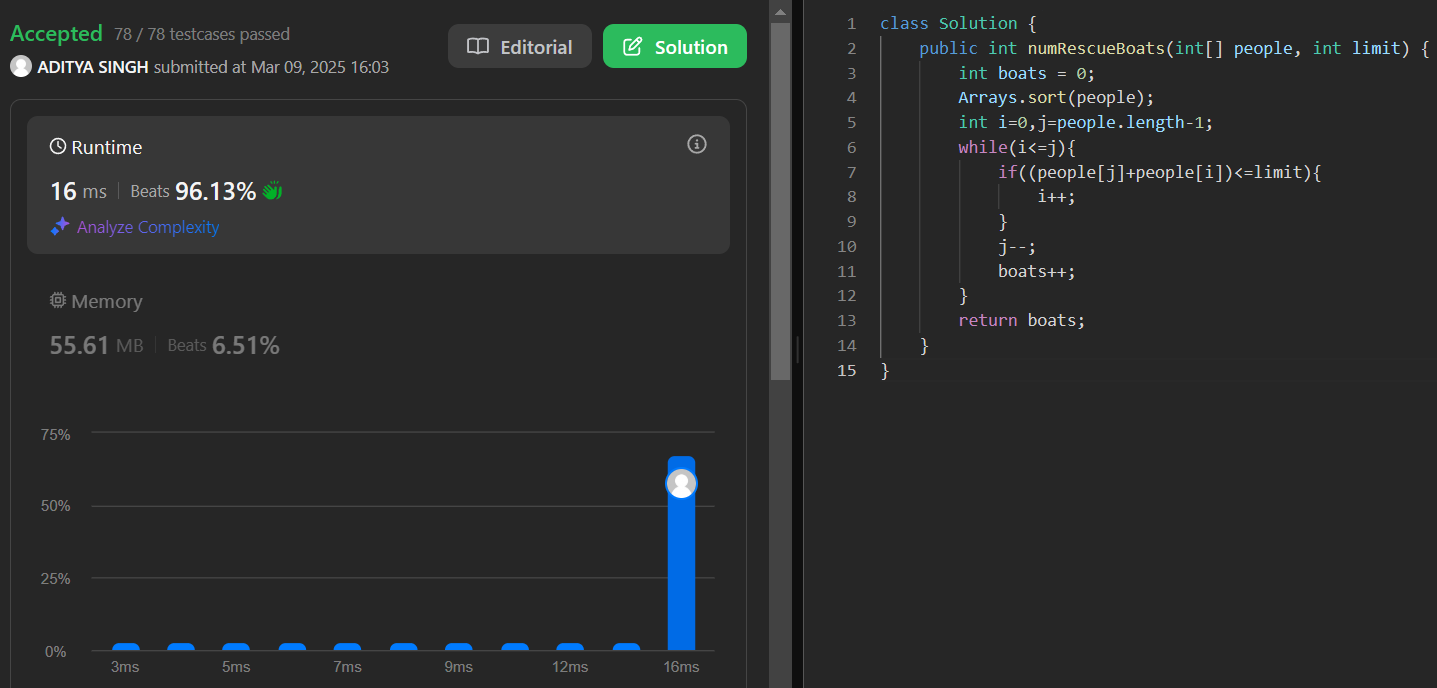
}

return boats;

}

}

Result:



1. **K Closest Points to Origin:**

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();

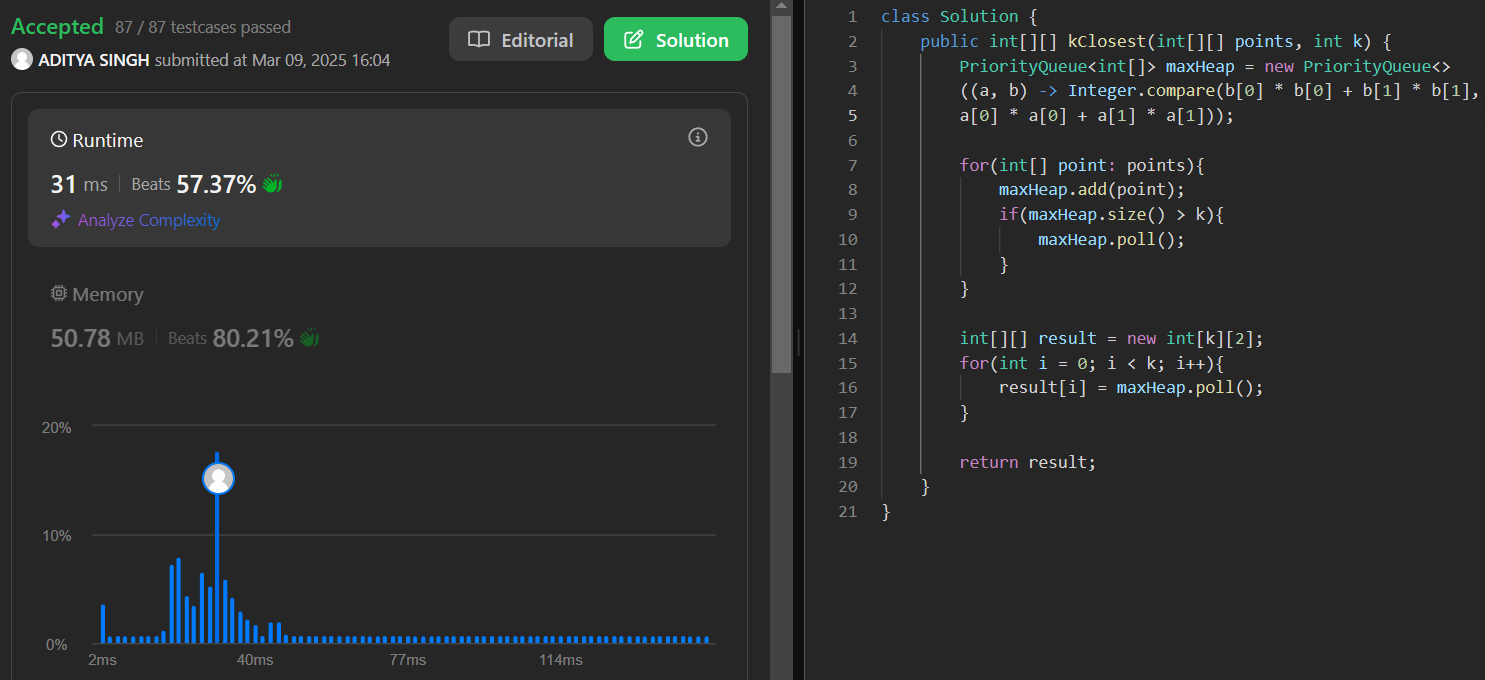
}

return result;

}

}

Result:



1. **Reduce Array Size to The Half:**

class Solution {

public int minSetSize(int[] arr) {

int n = arr.length;

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

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

int[] counting = new int[n + 1];

for (int freq : cnt.values()) ++counting[freq];

int ans = 0, removed = 0, half = n / 2, freq = n;

while (removed < half) {

ans += 1;

while (counting[freq] == 0) --freq;

removed += freq;

--counting[freq];

}

return ans;

}

}

Result:

