ASSIGNMENT-5

NAME: SACHIN DADHWAL **SEC: 605 /B**

UID: 22BCS12149

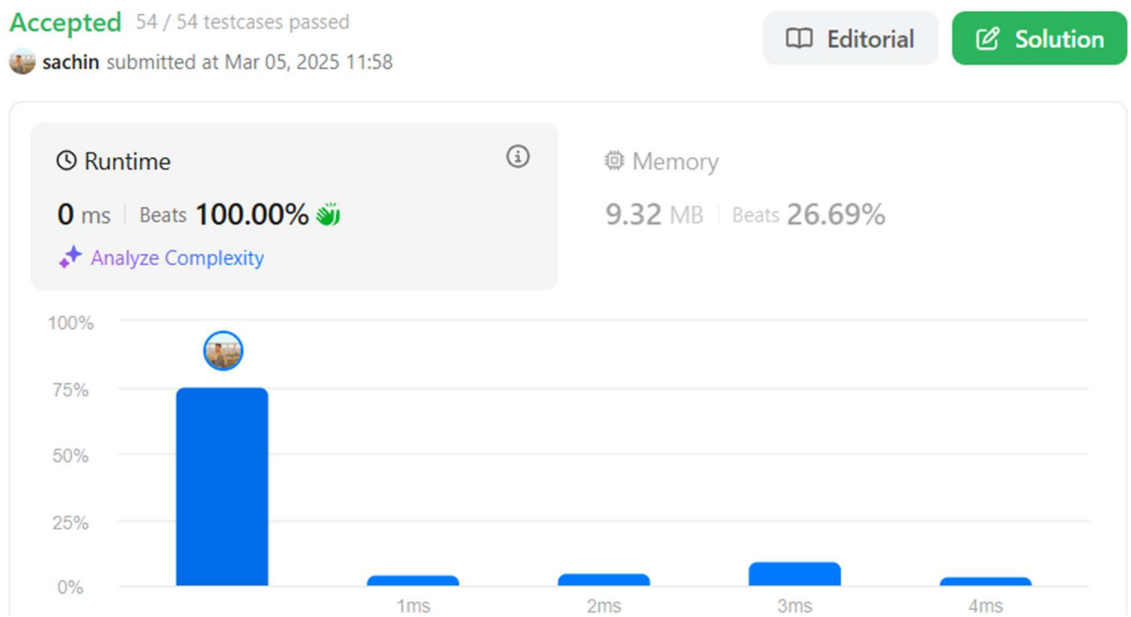
1. **389.Find the diffrence.**

   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.

   CODE:

```
class Solution {
public:
   char findTheDifference(string s, string t) {
int arr[26]={0};
for(auto a:s){
   arr[a-'a']++;
}
for(auto i:t){
   if(!arr[i-'a']) return i;
   arr[i-'a']--;   }
return 'e';    }};
```

   OUTPUT:



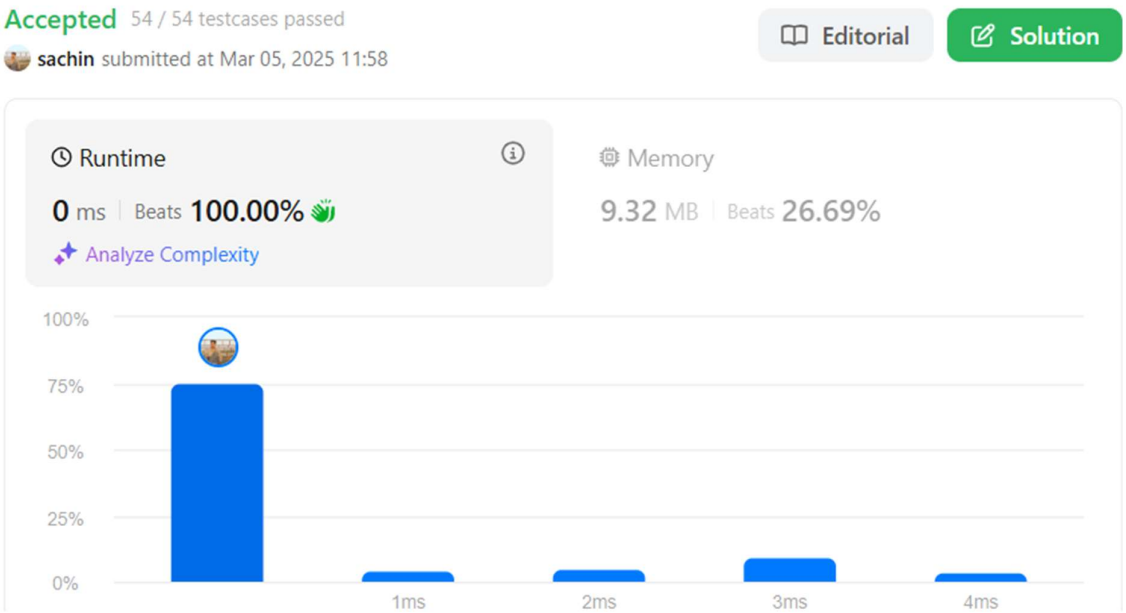
2. **976.Largest Perimeter Triangle.**

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.

CODE:

```
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;
   }
};
```

OUTPUT:



3. **414.[Third Maximum Number](#).**

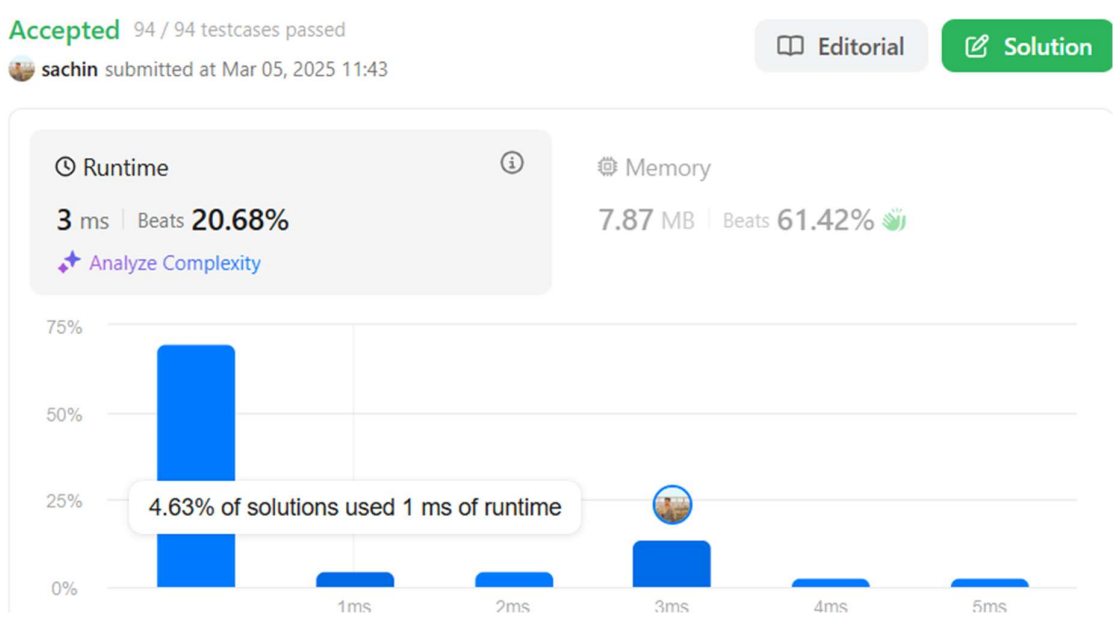
Given an integer array nums, return *the third distinct maximum number in this array. If the third maximum does not exist, return the maximum number*.

CODE:

```
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;
  }
}
```

OUTPUT:



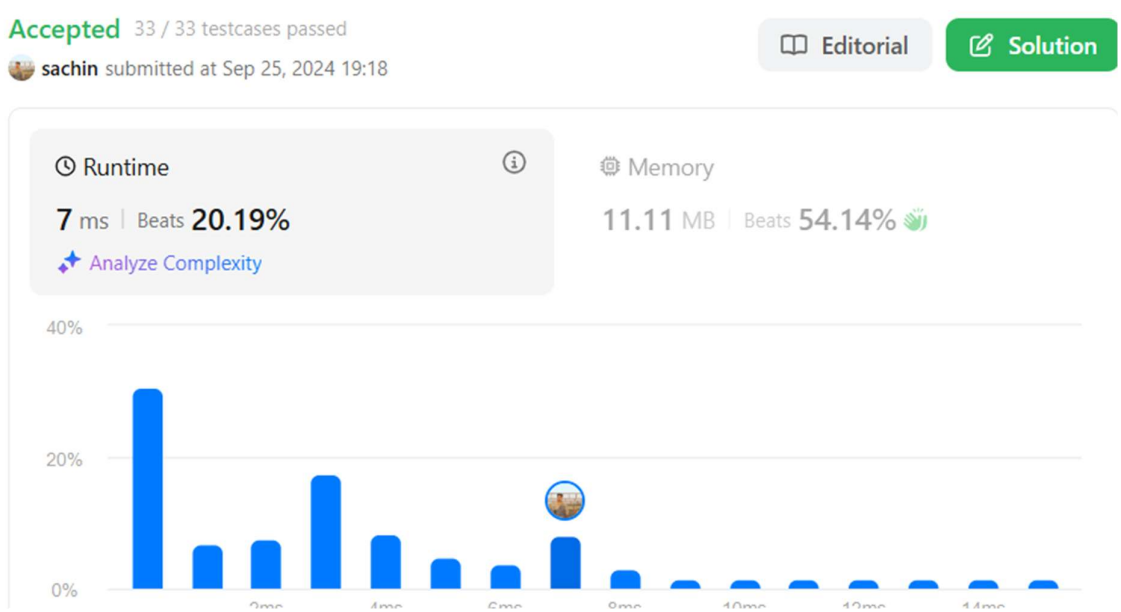
4. **451.[Sort Characters By Frequency]**

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*.

CODE:

```
class Solution {
public:
   string frequencySort(string s) {
      unordered_map<char, int> mp;
      vector<pair<int, char>> v;
      string ans = "";
      // count character frequency
      for(auto ch: s){
         mp[ch]++;
      }
      // push from map to vector
      for(auto i: mp){
         v.push_back({i.second, i.first});
      }
      // sort the vector in decreasing order
      sort(v.begin(), v.end(), greater<pair<int, char>>());
      // add to final answer string
      for(auto i: v){
         while(i.first--) ans += i.second;
      }
      return ans;
   }
};
```

OUTPUT:



5. **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] = [x_{start}, x_{end}] denotes a balloon whose **horizontal diameter** stretches between x_{start} and x_{end}. You do not know the exact y-coordinates of the balloons.

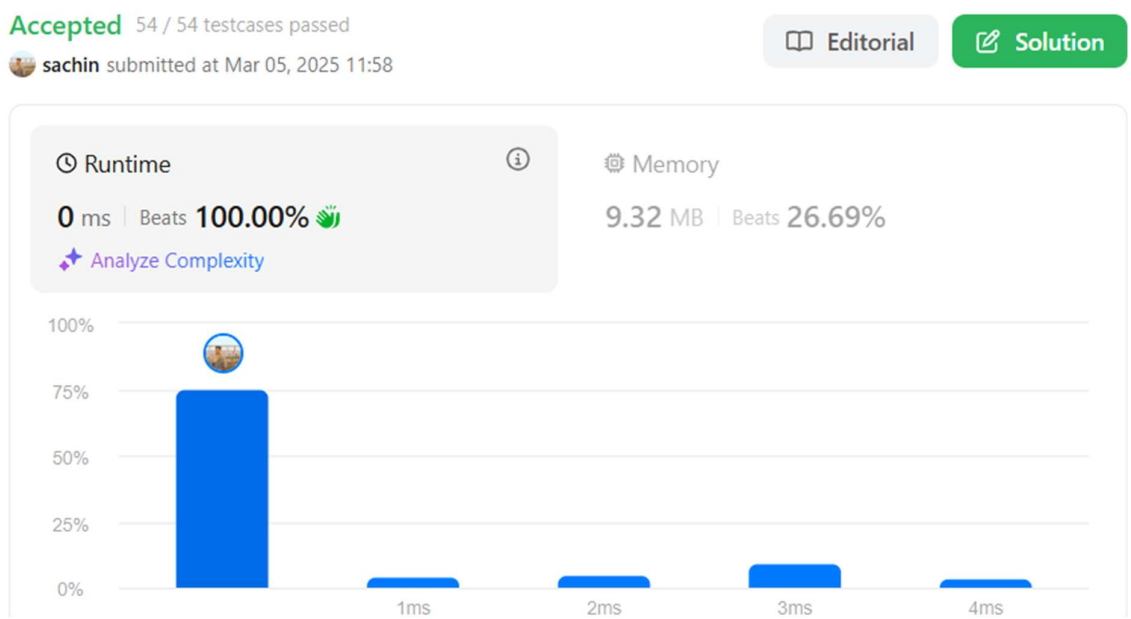
CODE:

```
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;
  }
};
```

OUTPUT:



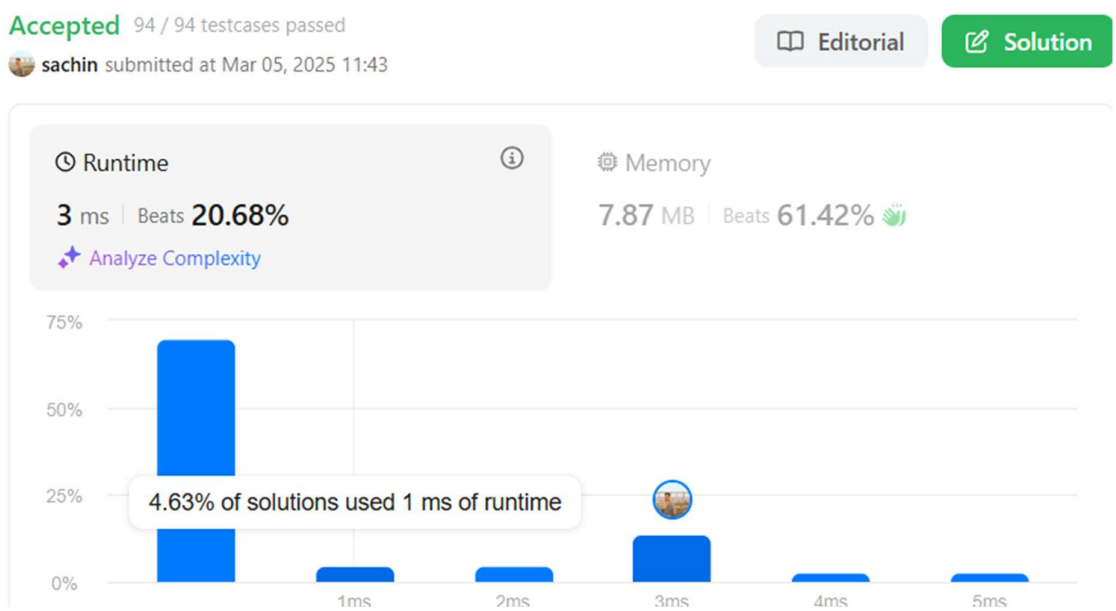
6. **881.[Boats to Save People](#).**

You are given an array people where people[i] is the weight of the ith 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.

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:



7. **973.[K Closest Points to Origin](#).**

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).

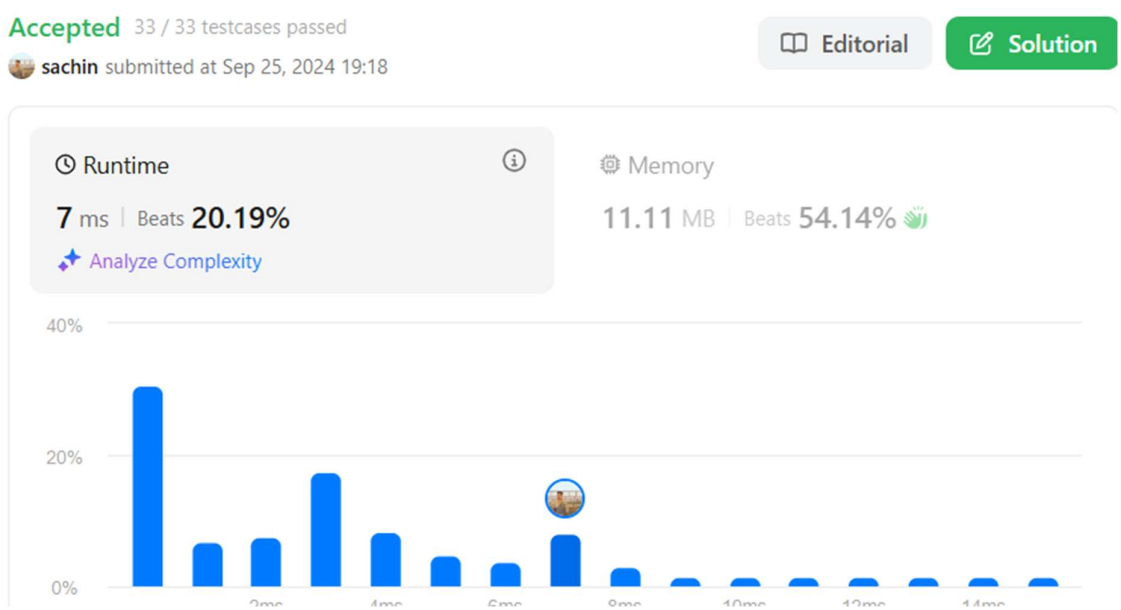
CODE:

```
class Solution {
public:
  vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
    // Max heap to store distances and corresponding points
    priority_queue<pair<int, vector<int>>> maxHeap;

    for (auto& point : points) {
      int distance = point[0] * point[0] + point[1] * point[1];
      maxHeap.push({distance, point});
      if (maxHeap.size() > k) maxHeap.pop(); // Remove farthest point if size > k
    }

    vector<vector<int>> ans;
    while (!maxHeap.empty()) {
      ans.push_back(maxHeap.top().second);
      maxHeap.pop();
    }
    return ans;
  }
};
```

OUTPUT:



8. **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.

CODE:

```
class Solution {
public:
   int minSetSize(vector<int>& arr) {
      int n = arr.size();
      unordered_map<int, int> cnt;
      for (int x : arr) ++cnt[x];

      vector<int> counting(n + 1);
      for (auto [_, freq] : cnt) ++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;
   }
};
```

OUTPUT: