**Advanced Programming 2**

**Assignment 5**

1. **389.Find the diffrence**

**Code:**

class Solution {

public:

    char findTheDifference(string s, string t) {

        std::unordered\_map<char, int> count;

        for (char c : t) {

            count[c]++;

        }

        for (char c : s) {

            count[c]--;

            if (count[c] == 0) {

                count.erase(c);

            }

        }

        return count.begin()->first;

    }

};

**Output:**

****

1. **976.Largest Perimeter Triangle**

**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:**

****

1. **414.Third Maximum Number**

**Code:**

class Solution {

public:

    int thirdMax(vector<int>& nums) {

        sort(nums.begin(), nums.end(), greater<int>());

        int elemCounted = 1;

        int prevElem = nums[0];

        for (int index = 1; index < nums.size(); ++index) {

            if (nums[index] != prevElem) {

                elemCounted += 1;

                prevElem = nums[index];

            }

            if (elemCounted == 3) {

                return nums[index];

            }

        }

        return nums[0];

    }

};

**Output:**

****

1. **451.Sort Characters By Frequency**

**Code:**

class Solution {

public:

    string frequencySort(string s) {

        string ans= "";

        unordered\_map<char,int>mp;

        for(auto c:s){

            mp[c]++;

        }

        vector<pair<int,char>> x;

        for(auto c:mp){

            x.push\_back({c.second,c.first});

        }

        sort(x.rbegin(),x.rend());

        for(auto c:x){

            int i=c.first;

            while(i--){

                ans+=c.second;

            }

        }

        return ans;

    }

};

**Output:**

****

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

**Code:**

class Solution {

public:

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

        sort(points.begin(), points.end(), [](const vector<int>& a, const vector<int>& b) {

            return a[1] < b[1];

        });

        int arrows = 1;

        int prevEnd = points[0][1];

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

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

                arrows++;

                prevEnd = points[i][1];

            }

        }

        return arrows;

    }

};

**Output:**

****

1. **881.Boats to Save People**

**Code:**

class Solution {

public:

    int numRescueBoats(vector<int>& people, int limit) {

        int boatCount = 0;

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

        int left = 0;

        int right = people.size() - 1;

        while(left <= right){

            int sum = people[left] + people[right];

            if(sum <= limit){

                boatCount++;

                left++;

                right--;

            }

            else{

                boatCount++;

                right--;

            }

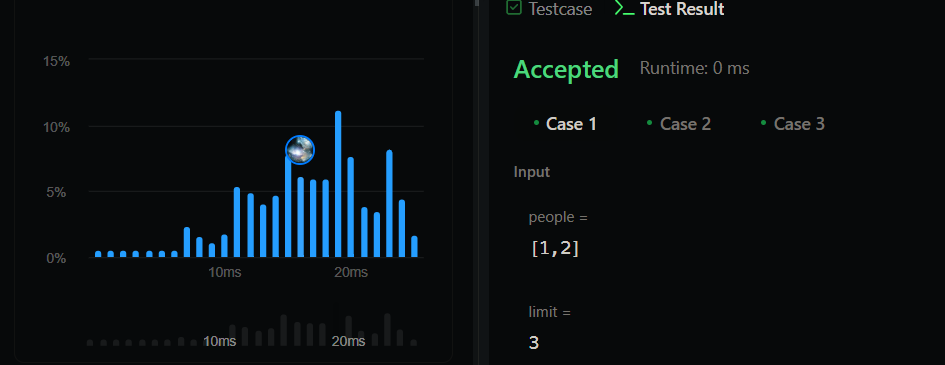
        }

        return boatCount;

    }

};

**Output:**

****

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

**Code:**

class Solution {

public:

    vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {

        priority\_queue<pair<int, vector<int>>> closestpoints;

        for(auto point:points)

        {

            int x\_cord = point[0];

            int y\_cord = point[1];

            int dist = x\_cord\*x\_cord+y\_cord\*y\_cord;

            if(closestpoints.size()<k)

            {

                closestpoints.push({dist,point});

            }

            else if(dist<closestpoints.top().first)

            {

                closestpoints.pop();

                closestpoints.push({dist,point});

            }

        }

        vector<vector<int>> resultClosePoints;

        while(k>0)

        {

            resultClosePoints.push\_back(closestpoints.top().second);

            closestpoints.pop();

            k--;

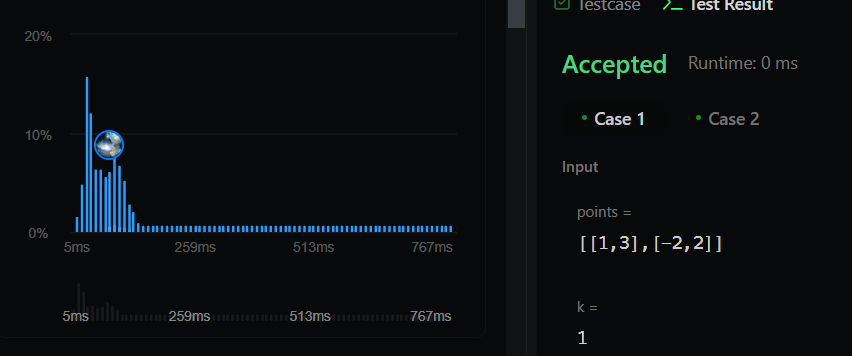
        }

        return resultClosePoints;

    }

};

**Output:**

****

1. **1338.Reduce Array Size to The Half**

**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];

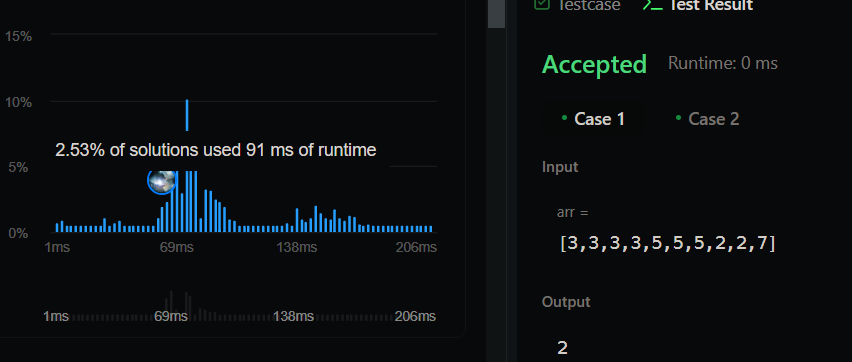
        }

        return ans;

    }

};

**Output:**

****