# ASSIGNMENT 1

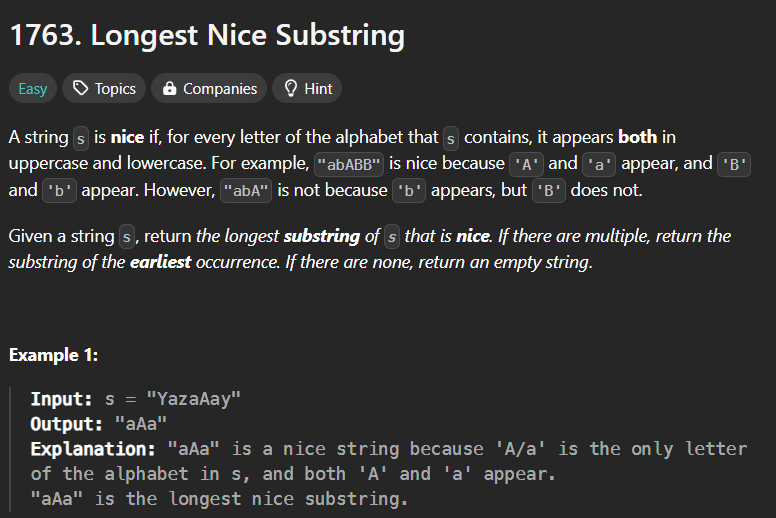
**Student Name: Mehak Chauhan UID: 22BCS12675**

**Branch: CSE Section: 22BCS\_IOT\_605 B**

**Semester: 6th DOP:05-02-2025**

**Subject: Advanced Programming Lab-II Subject Code: 22CSP-351**

**Question 1**

****

**Code:**

class Solution {

public:

   bool niceSubstr(string&s,int&i,int&j,int&n) {

       int l\_freq[26]={0} , u\_freq[26]={0};

       for(int x = i;x<=j;x++) {

           if(s[x]>='a' && s[x]<='z')

               l\_freq[s[x]-'a']++;

           else

               u\_freq[s[x]-'A']++;

       }

       for(int i=0;i<26;i++) {

           if((l\_freq[i]&&!u\_freq[i]) ||(!l\_freq[i]&&u\_freq[i]))

               return 0;

       }

       return true;

    }

    string longestNiceSubstring(string s) {

        int maxLen =0;

        string ans = "";

        int n = s.length();

        for(int i=0;i<n;i++) {

            for(int j=i+1;j<n;j++) {

                if(niceSubstr(s,i,j,n)) {

                    if(maxLen < j-i+1){

                        maxLen = j-i+1;

                        ans = s.substr(i,j-i+1);

                    }

                }

            }

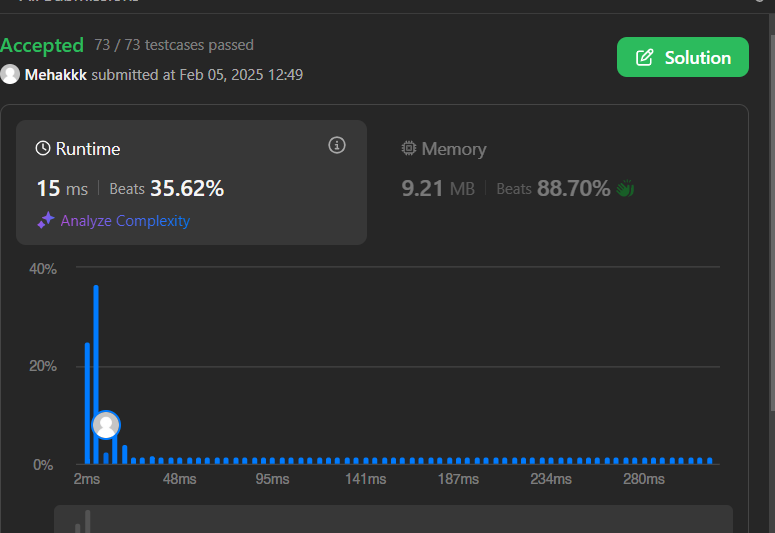
        }

        return ans;

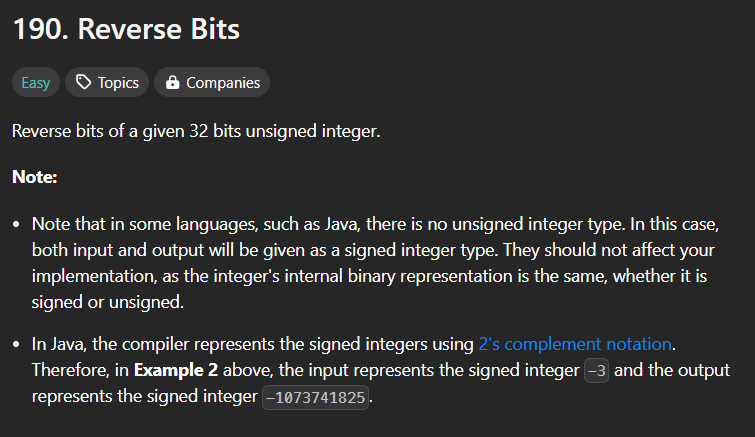
    }

};

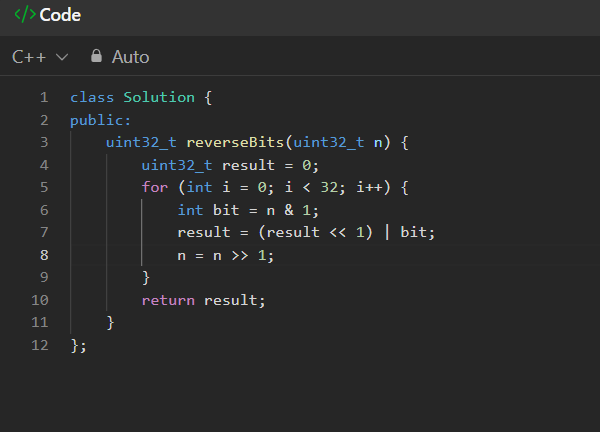
**Output:**

****

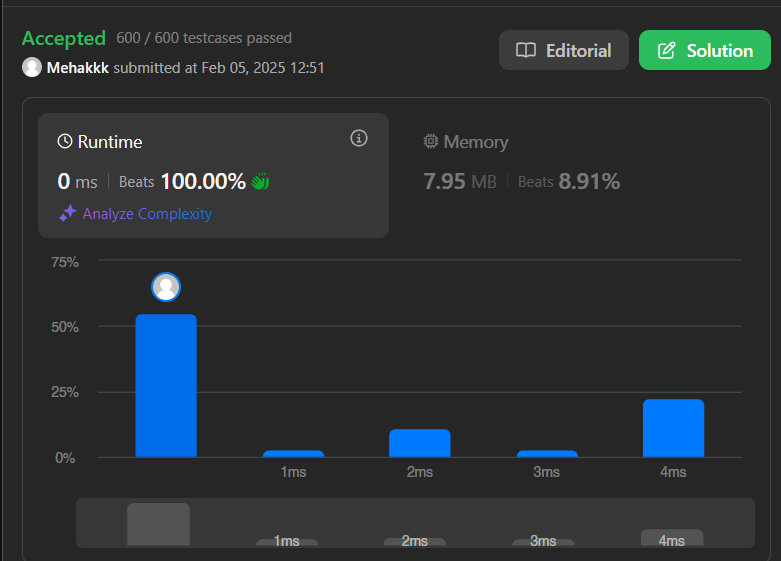
**Question 2**

****

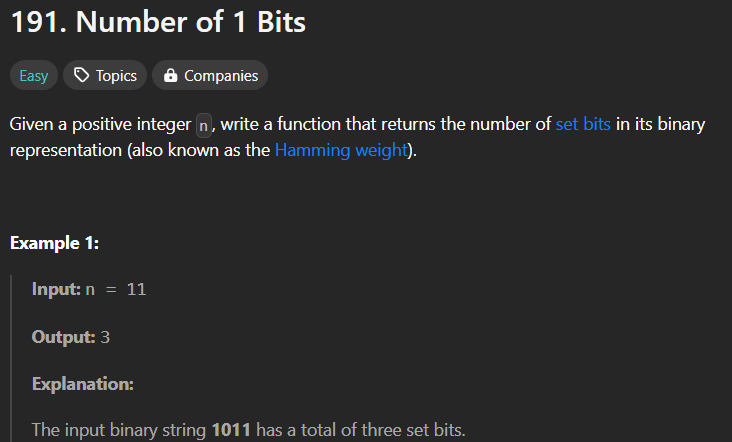
**Code:**

****

**Output:**

****

**Question 3**

****

**Code:**

class Solution {

public:

    int hammingWeight(uint32\_t n) {

        int res = 0;

        for (int i = 0; i < 32; i++) {

            if ((n >> i) & 1) {

                res += 1;

            }

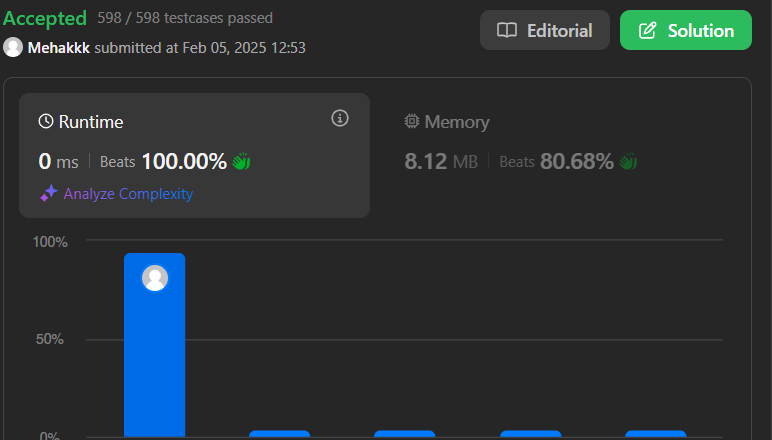
        }

        return res;

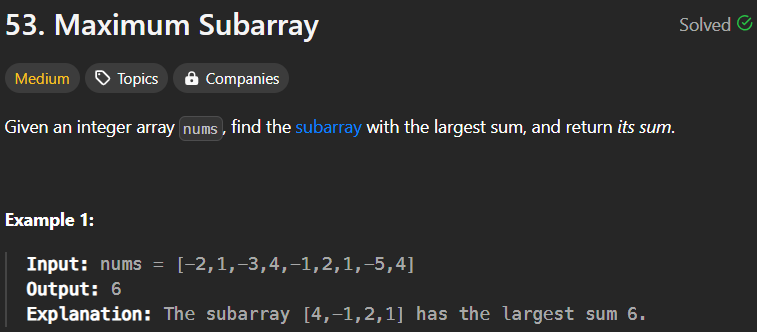
    }

};

**Output:**

****

**Question 4**

****

**Code:**

class Solution {

public:

    int maxSubArray(vector<int>& nums) {

        int result = nums[0];

        int curr\_sum = nums[0];

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

            if(nums.size()==1){

                return nums[i];

            }

            curr\_sum= max(nums[i],curr\_sum+nums[i]);

            result= max(result,curr\_sum);

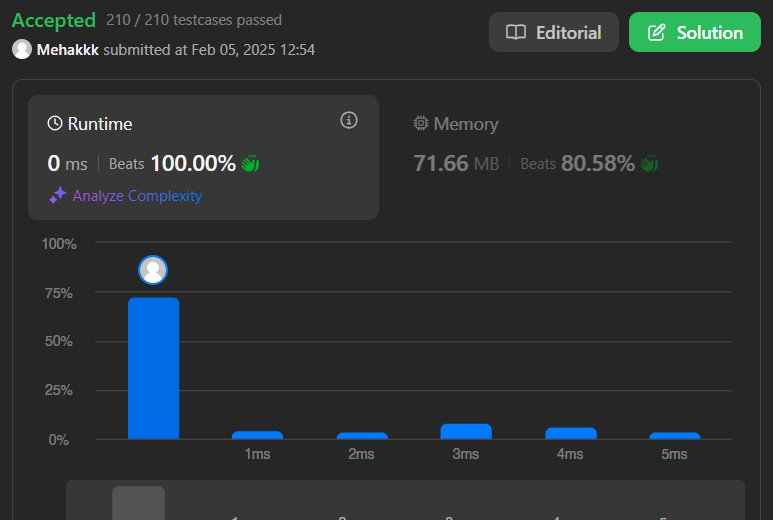
        }

        return result;

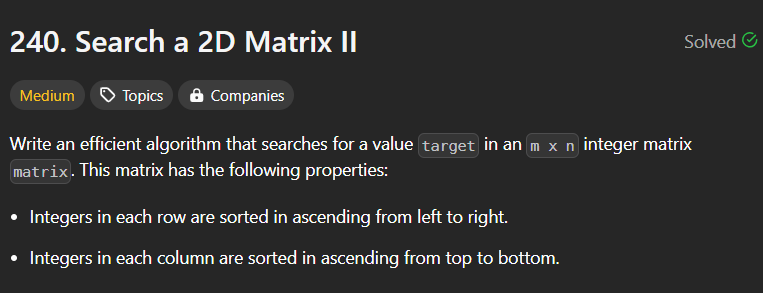
    }

};

**Output:**

****

**Question 5**

****

**Code:**

class Solution {

public:

    bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int n = matrix.size(), m = matrix[0].size();

        int row = 0, col = m - 1;

        while (row < n && col >= 0) {

            if (matrix[row][col] == target) return true;

            else if (matrix[row][col] < target) row++;

            else col--;

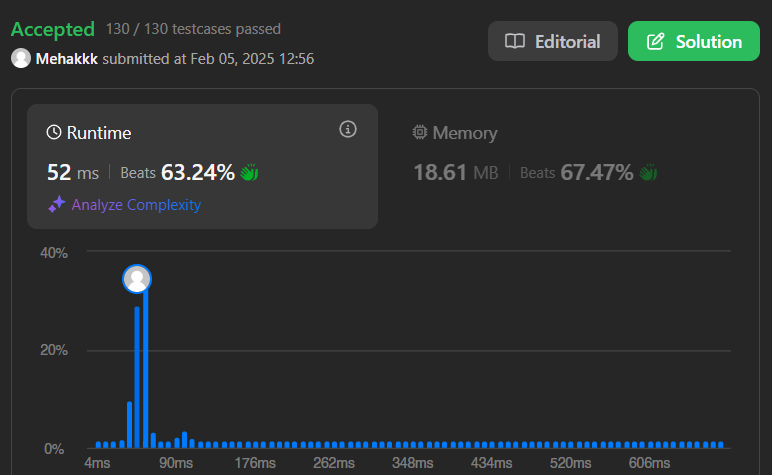
        }

        return false;

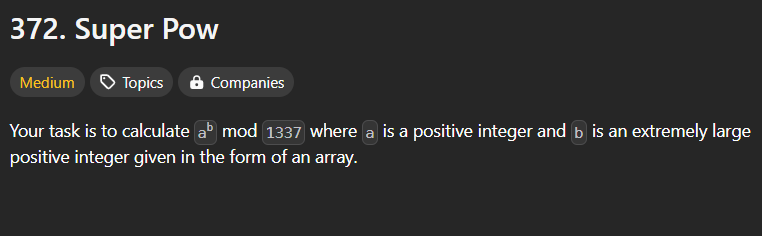
    }

};

**Output:**

****

**Question 6**

****

**Code**

int n=1337;

int phi=1140;

class Solution {

public:

    int Chinese\_Remainder(int a,  int x, vector<int>& b){

        if (x==n) return 0;

        int p=n/x;

        int M;//modInverse i.e. x\*M==1%p

        if (x==7) M=82;//can be computed by extended euclidean algorithm

        else M=4;

        int s=b.size();

        int exp=0;

        for(int i=0; i<s; i++)

            exp=(b[i]+10\*exp)%(p-1);

        bitset<12> e(exp);

        int y=1;

        a%=n;

        for(int i=11; i>=0; i--){

            y=y\*y%n;

            if (e[i]==1) y=y\*a%n;

        }

        int ans=y\*M\*x%n;

        while( ans<0)

            ans+=n;

    //    cout<<ans<<endl;

        return ans;//Chinese Remainder Theorem

    }

    int superPow(int a, vector<int>& b) {

        int g=gcd(a, n);

    //    cout<<"gcd="<<g<<endl;

        if (g!=1) return Chinese\_Remainder(a, g, b);

        int s=b.size();

        int exp=0;

        for(int i=0; i<s; i++)

            exp=(b[i]+10\*exp)%phi;

        bitset<12> e(exp);

        int y=1;

        a%=n;

        for(int&& i=11; i>=0; i--){

            y=y\*y%n;

            if (e[i]==1) y=y\*a%n;

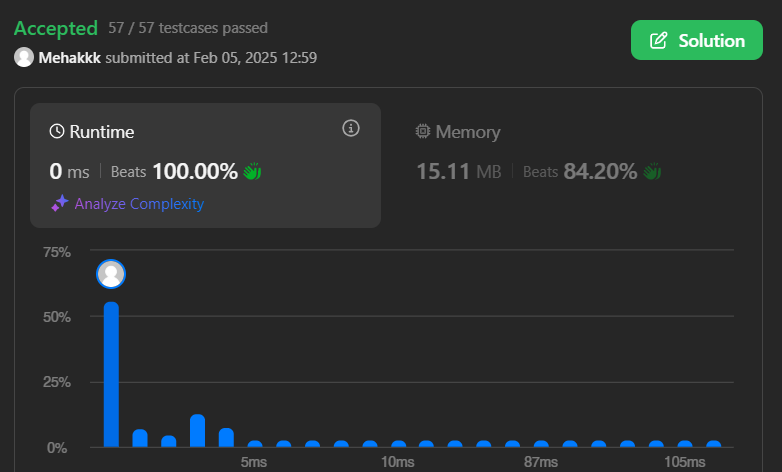
        }

        return y;

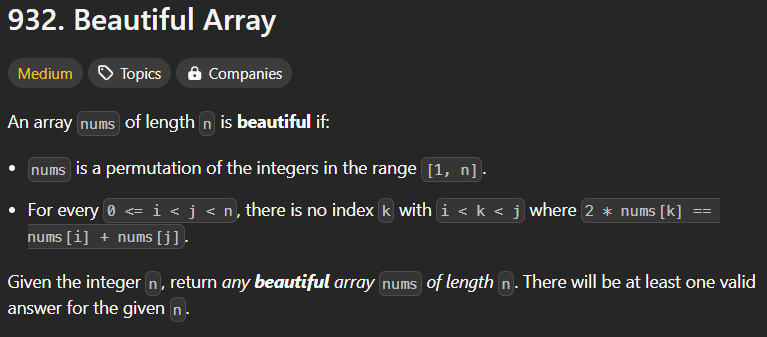
    }

};

**Output**

****

**Question 7**

****

**Code:**

class Solution {

public:

    int partition(vector<int> &v, int start, int end, int mask)

    {

        int j = start;

        for(int i = start; i <= end; i++)

        {

            if((v[i] & mask) != 0)

            {

                swap(v[i], v[j]);

                j++;

            }

        }

        return j;

    }

    void sort(vector<int> & v, int start, int end, int mask)

    {

        if(start >= end) return;

        int mid = partition(v, start, end, mask);

        sort(v, start, mid - 1, mask << 1);

        sort(v, mid, end, mask << 1);

    }

    vector<int> beautifulArray(int N) {

        vector<int> ans;

        for(int i = 0; i < N; i++) ans.push\_back(i + 1);

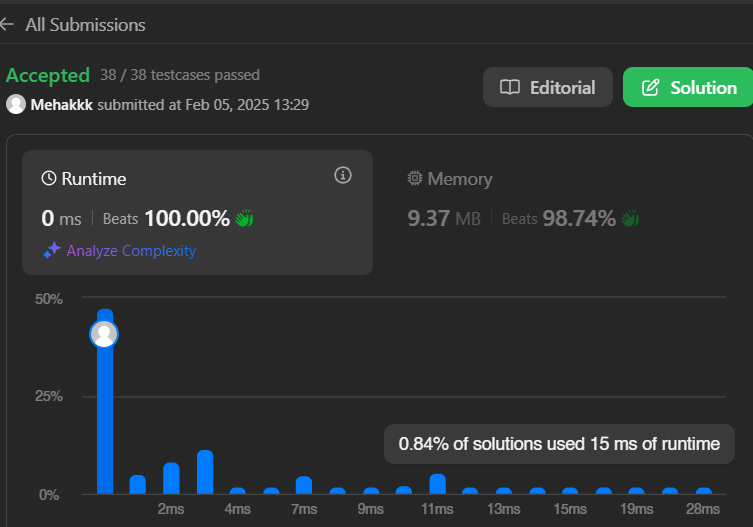
        sort(ans, 0, N - 1, 1);

        return ans;

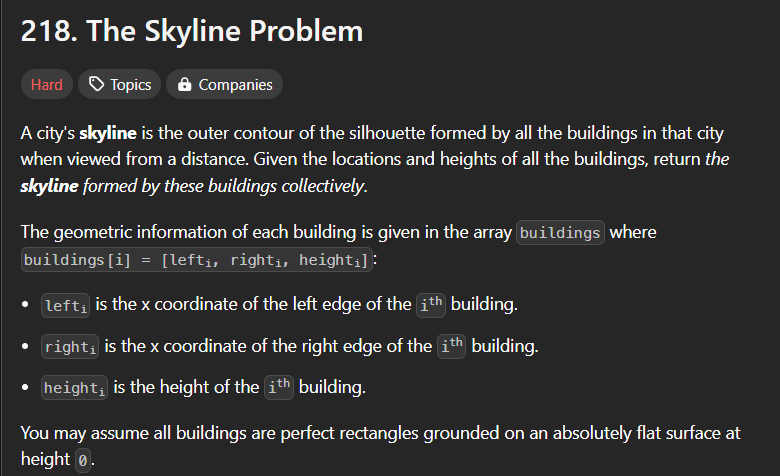
    }

};

**Output:**

****

**Question 8**

****

**Code:**

class Solution {

public:

    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

        vector<pair<int, int>> h;

        // Convert each building into two segments.

        for (auto b : buildings) {

            h.push\_back({b[0], -b[2]});

            h.push\_back({b[1], b[2]});

        }

        // Sort the segments.

        sort(h.begin(), h.end());

        int prev = 0, cur = 0;

        multiset<int> m;

        vector<vector<int>> res;

        m.insert(0);

        for (auto i:h) {

            // If i.second is less than zero, then it means it is left boundary.

            if (i.second < 0) {

                m.insert(-i.second);

            } else { // else it is right boundary.

                m.erase(m.find(i.second));

            }

            cur = \*m.rbegin();

            // If current maximum height is not equal to maximum previuous height, it is a key point.

            if (cur != prev) {

                res.push\_back({i.first, cur});

                prev = cur;

            }

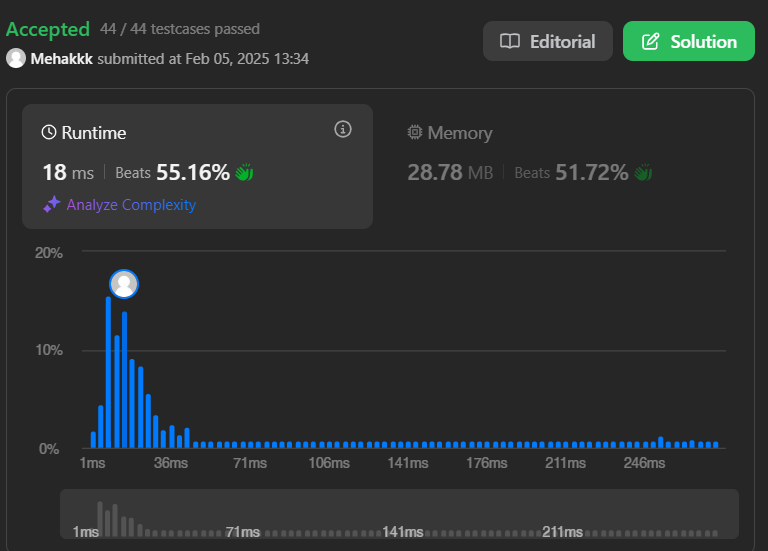
        }

        return res;

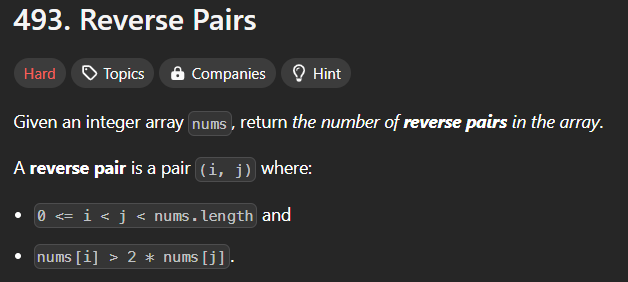
}

};

**Output:**

****

**Question 9**

****

**Code:**

class Solution {

public:

    void merge(vector<int>&arr, int low, int mid, int high){

        int left = low, right = mid+1; vector<int>temp;

        while(left <= mid && right <= high){

            if(arr[left] <= arr[right]){

                temp.push\_back(arr[left++]);

            }

            else{

                temp.push\_back(arr[right++]);

            }

        }

        while(left <= mid){

            temp.push\_back(arr[left++]);

        }

        while(right <= high){

            temp.push\_back(arr[right++]);

        }

        //now pushing back the temp array elmenets to original array

        for(int i=low; i<=high;i++){

            arr[i] = temp[i - low];

        }

    }

    int countPairs(vector<int>&arr, int low, int mid, int high){

        int count=0, right=mid+1;

        for(int i = low;i <= mid; i++){

            while (right <= high && (long long)arr[i] > 2LL \* arr[right]) right++;

            count += (right - (mid+1));

        }

        return count;

    }

    int mergeSort(vector<int>&arr, int low, int high){

        int count=0;

        if(low >= high) return count;

        int mid = (low + high)/2;

        count += mergeSort(arr, low, mid);

        count += mergeSort(arr, mid+1, high);

        count += countPairs(arr, low, mid, high);

        merge(arr, low, mid, high);

        return count;

    }

    int reversePairs(vector<int>& nums) {

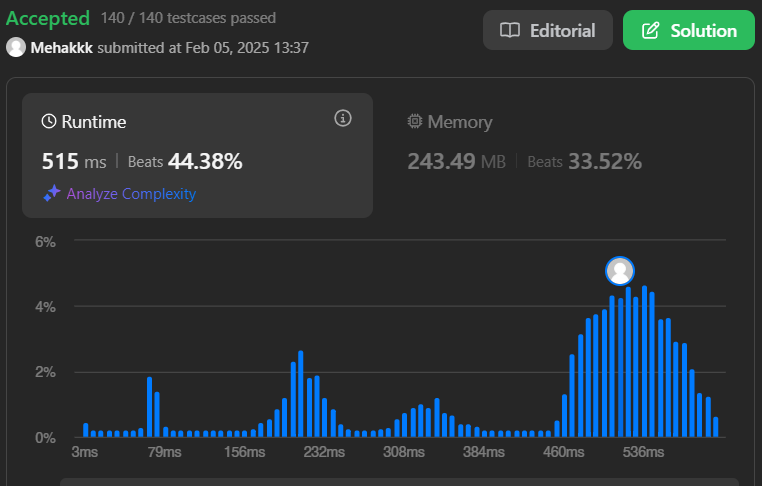
        int n = nums.size();

        return mergeSort(nums, 0, n-1);

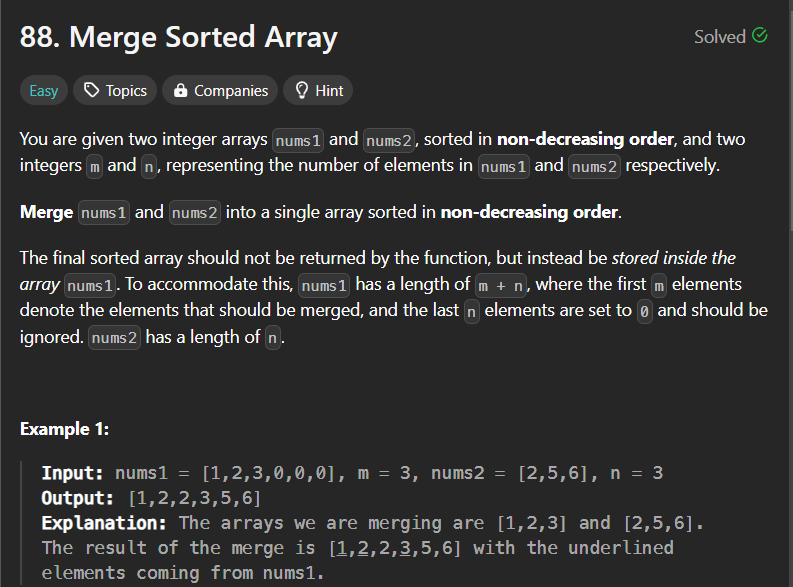
    }

};

**Output:**

****

**Question 10**

****

class Solution {

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        int i=n;

        int j=n;

        int k = m+n-1;

        while(m>0 && n>0){

            if(nums1[m-1]>nums2[n-1]){

                nums1[k]=nums1[m-1];

                m--;

            }

            else{

                nums1[k]=nums2[n-1];

                n--;

            }

            k--;

        }

        while (n > 0) {

        nums1[k] = nums2[n - 1];

        n--;

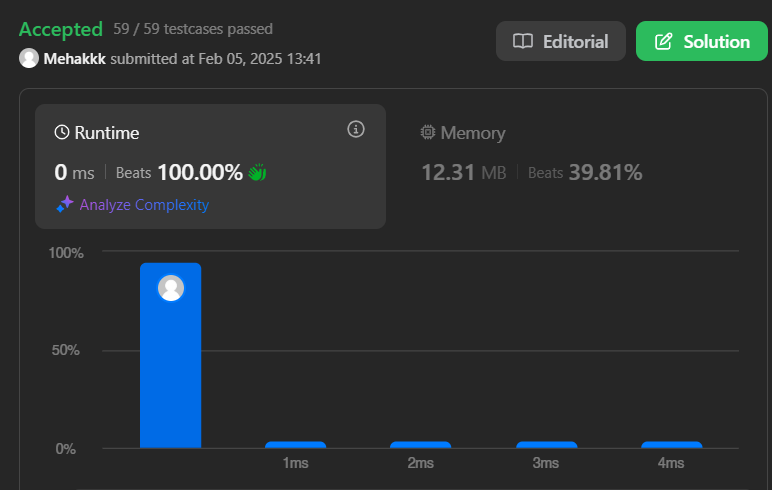
        k--;

    }

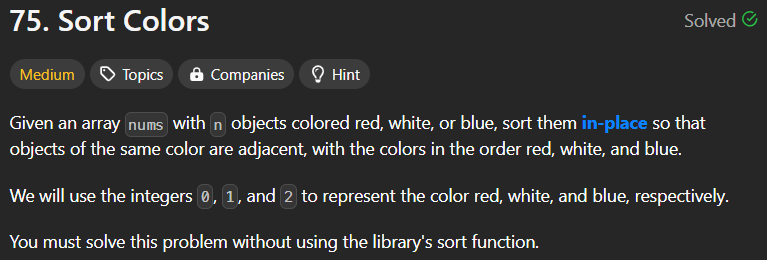
            }

};

**Output:**

****

**Question 11**

****

**Code:**

class Solution {

public:

     void sortColors(vector<int>& nums) {

        int low=0;

        int mid=0;

        int high= nums.size()-1;

        while(mid<=high){

            if(nums[mid]==0){

                swap(nums[low++],nums[mid++]);

            }

            else if(nums[mid]==1){

                mid++;

            }

            else{

                swap(nums[mid],nums[high--]);

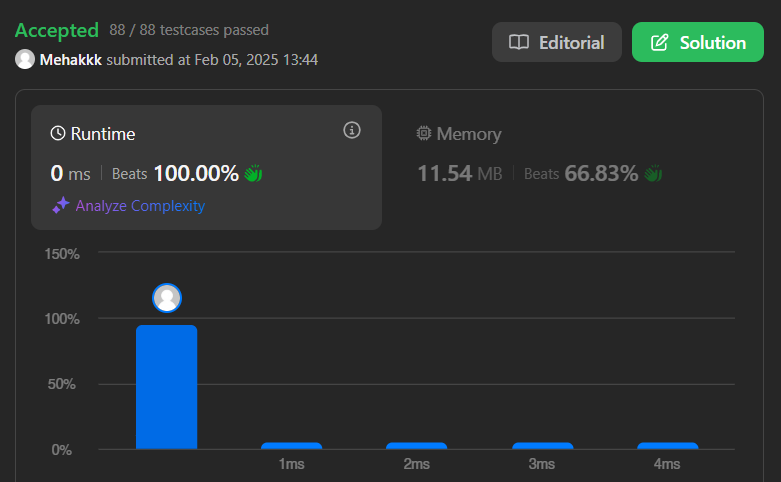
            }

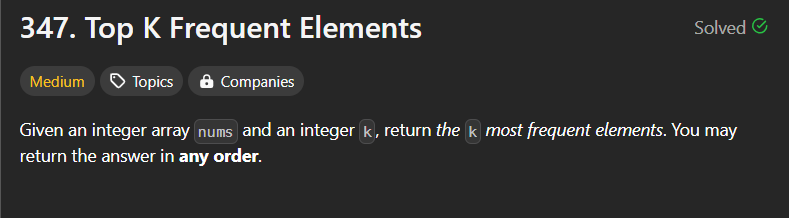
        }

    }

};

**Output:**

****

**Question 12**

**Code:**

class Solution {

public:

    vector<int> topKFrequent(vector<int>& nums, int k) {

        unordered\_map<int, int> freq;

        for (int num : nums) {

            freq[num]++;

        }

        priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> minHeap;

        for (auto it : freq) {

            minHeap.push({it.second, it.first});

            if (minHeap.size() > k) {

                minHeap.pop();

            }

        }

        vector<int> ans;

        while (!minHeap.empty()) {

            ans.push\_back(minHeap.top().second);

            minHeap.pop();

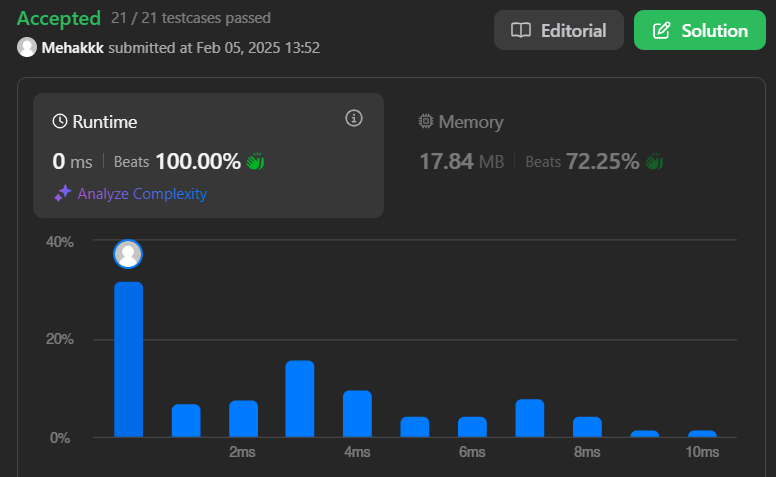
        }

        return ans;

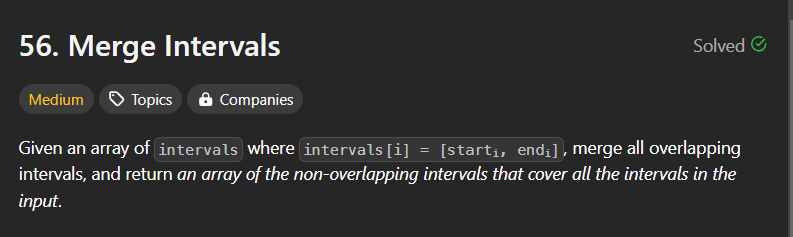
    }

};

**Output:**

****

**Question 13**

****

**Code:**

class Solution {

public:

    vector<vector<int>> merge(vector<vector<int>>& intervals) {

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

        vector<vector<int>> ans;

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

            if(ans.empty() || intervals[i][0]>ans.back()[1]){

                ans.push\_back(intervals[i]);

            }

            else{

                ans.back()[1]= max(ans.back()[1], intervals[i][1]);

            }

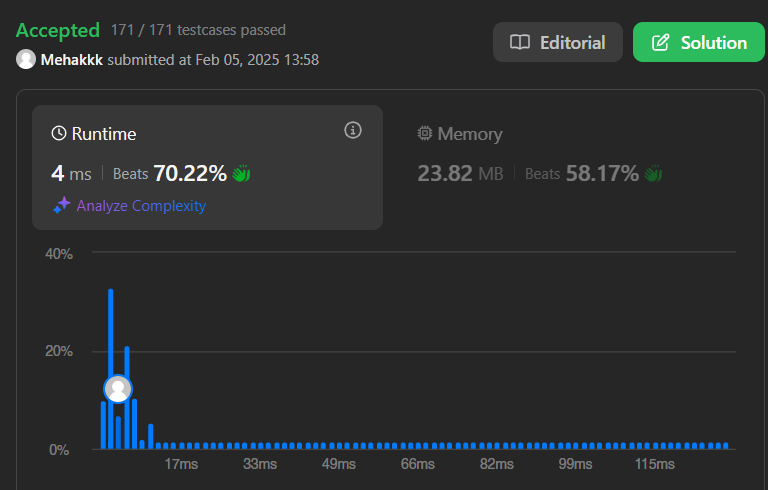
        }

return ans;

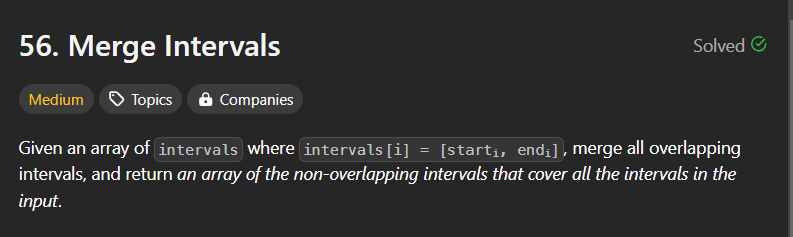
    }

};

**Output:**

****

**Question 14**

****

**Code:**

class Solution {

public:

    int search(vector<int>& nums, int target) {

        int n=nums.size();

        for(int i=0;i<n;i++){

            if(nums[i]==target){

                return i;

            }

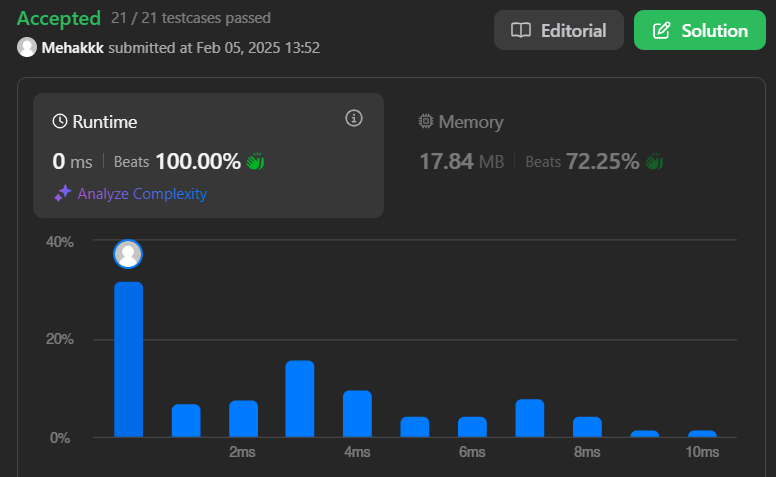
        }

        return -1;

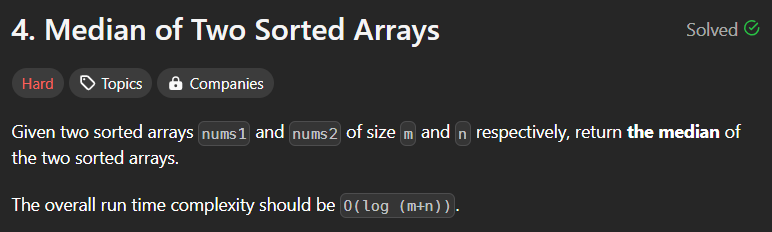
    }

};

**Output:**

****

**Question 15**

****

**Code:**

class Solution {

public:

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

        // Get the sizes of both input arrays.

        int n = nums1.size();

        int m = nums2.size();

        // Merge the arrays into a single sorted array.

        vector<int> merged;

        for (int i = 0; i < n; i++) {

            merged.push\_back(nums1[i]);

        }

        for (int i = 0; i < m; i++) {

            merged.push\_back(nums2[i]);

        }

        // Sort the merged array.

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

        // Calculate the total number of elements in the merged array.

        int total = merged.size();

        if (total % 2 == 1) {

            // If the total number of elements is odd, return the middle element as the median.

            return static\_cast<double>(merged[total / 2]);

        } else {

            // If the total number of elements is even, calculate the average of the two middle elements as the median.

            int middle1 = merged[total / 2 - 1];

            int middle2 = merged[total / 2];

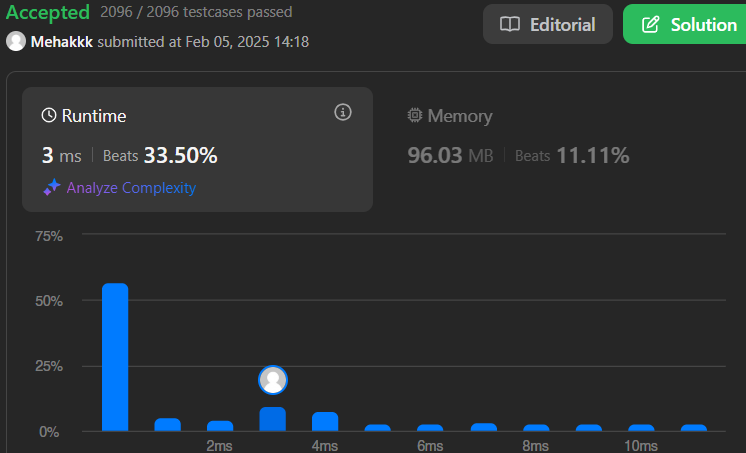
            return (static\_cast<double>(middle1) + static\_cast<double>(middle2)) / 2.0;

        }

    }

};

**Output:**

****