**ASSIGNMENT – 1**

**Name: Sachin dadhwal section:605/B**

**UID: 22BCS12149 Subject: CG**

1. **1763.**[**Longest Nice Substring**](https://leetcode.com/problems/longest-nice-substring/description/)**.**

A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

Given a string s, return *the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string*.

**Code:**

class Solution

{

public:

string longestNiceSubstring(string s) {

if (s.size() < 2) return "";

unordered\_set<char> st(begin(s), end(s));

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

if (st.find((char) toupper(s[i])) == end(st) || st.find((char) tolower(s[i])) == end(st)) {

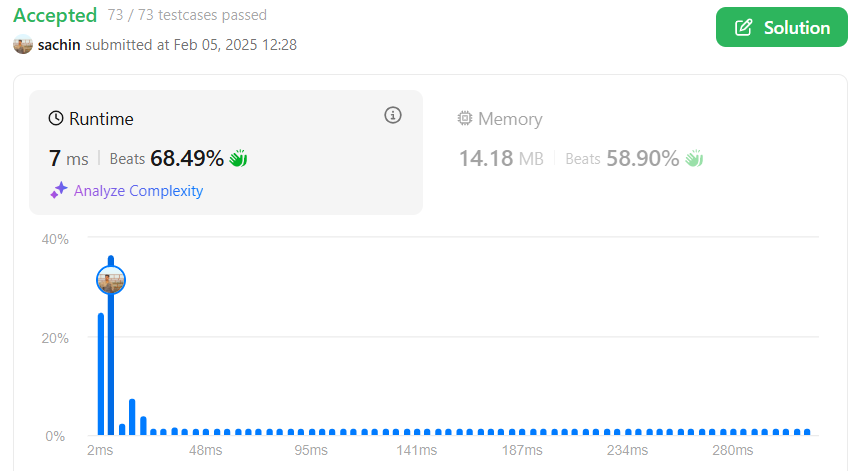
string s1 = longestNiceSubstring(s.substr(0, i));

string s2 = longestNiceSubstring(s.substr(i + 1));

return s1.size() >= s2.size() ? s1 : s2; } }

return s; } };

**Output:**

****

1. **190.**[**Reverse Bits**](https://leetcode.com/problems/reverse-bits/description/)**.**

Reverse bits of a given 32 bits unsigned integer.

**Code:**

 uint32\_t reverseBits(uint32\_t n) {

        uint32\_t result = 0;

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

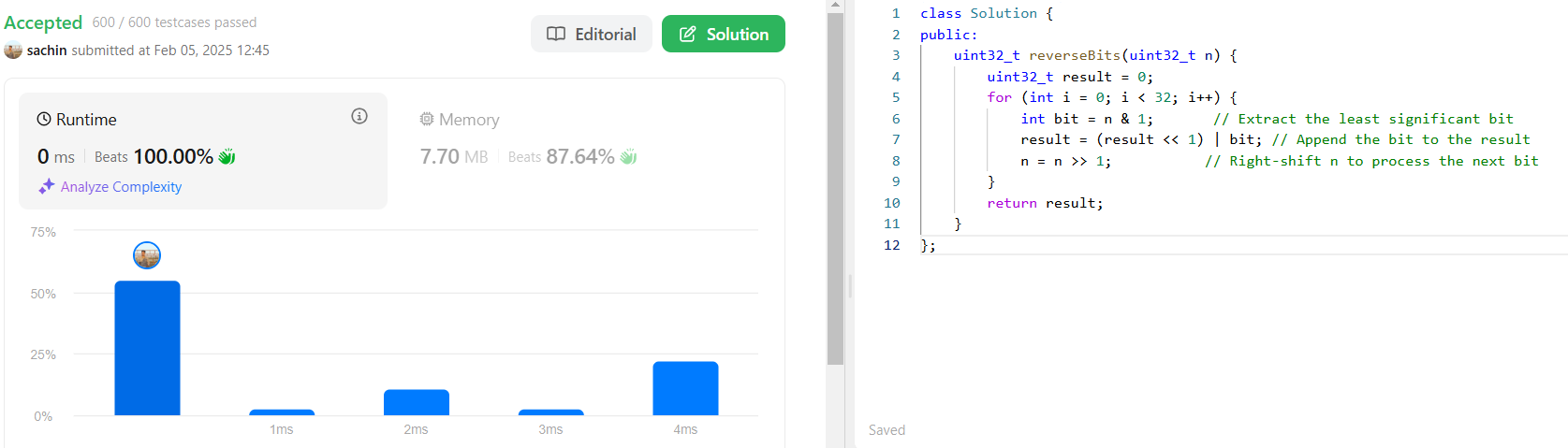
            int bit = n & 1;       // Extract the least significant bit

            result = (result << 1) | bit; // Append the bit to the result

            n = n >> 1;           // Right-shift n to process the next bit }

        return result; }

**OUTPUT:**



1. 191.[Number of 1 Bits](https://leetcode.com/problems/number-of-1-bits/description/).

Given a positive integer n, write a function that returns the number of

set bits in its binary representation (also known as the Hamming weight).

**CODE:**

int hammingWeight(uint32\_t n) {

int c=0;

while(n!=0)

{

if(n&1)

{

c++;

}

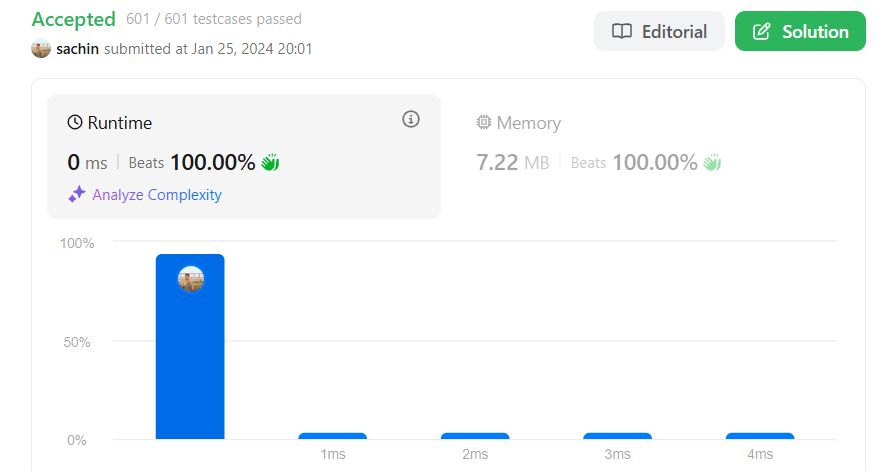
n=n>>1;

}

return c;

}

**OUTPUT:**

****

1. 53.[Maximum Subarray](https://leetcode.com/problems/maximum-subarray/description/).

Given an integer array nums, find the  Subarray with the largest sum, and return *its sum*.

**CODE:**

int maxSubArray(vector<int>& nums) {

int sum=0;

int max=nums[0];

int n=nums.size();

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

{

sum=sum+nums[i];

if(sum>max)

max=sum;

if(sum<0)

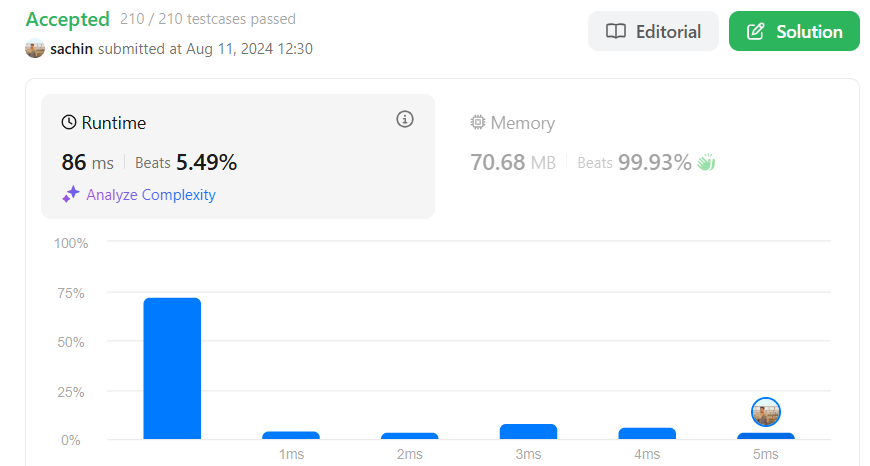
sum=0;

}

return max;

}

**OUTPUT:**

****

1. **240.**[**Search a 2D Matrix II**](https://leetcode.com/problems/search-a-2d-matrix-ii/description/)**.**

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

* Integers in each row are sorted in ascending from left to right.
* Integers in each column are sorted in ascending from top to bottom

**CODE:**

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

int row=matrix.size();

int col=matrix[0].size();

int r=0;

int c=col-1;

while(r<row && c>=0 ){

int key=matrix[r][c];

if(key==target {

return true; }

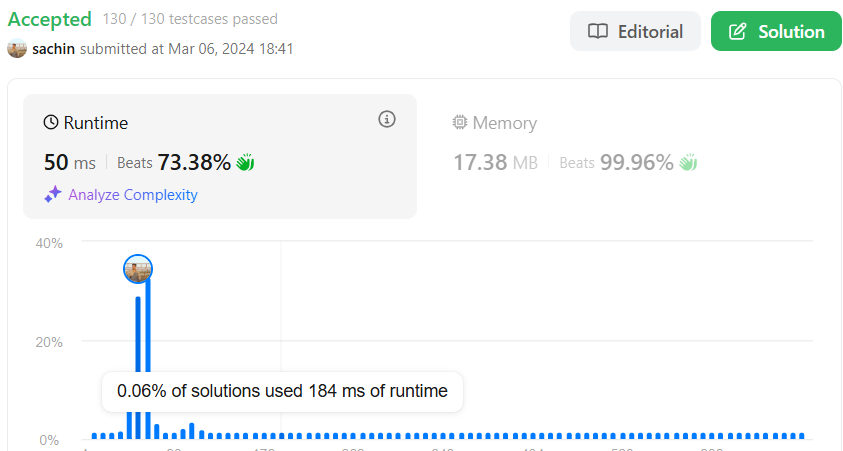
if(key<target){

r++; }

Else c--; } }

return false; }

**OUTPUT:**



1. 372.[Super Pow](https://leetcode.com/problems/super-pow/description/).

Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

**CODE:**

class Solution {

public:

    const int MOD = 1337;

    int pow(int a, int b) {

        int result = 1;

        a %= MOD;  // Taking mod to prevent overflow

        for (int i = 0; i < b; i++) {

            result = (result \* a) % MOD;

        }

        return result;

    }

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

        int result = 1;

        for (int i = b.size() - 1; i >= 0; i--) {

            result = (result \* pow(a, b[i])) % MOD;

            a = pow(a, 10);  // Power up for the next iteration

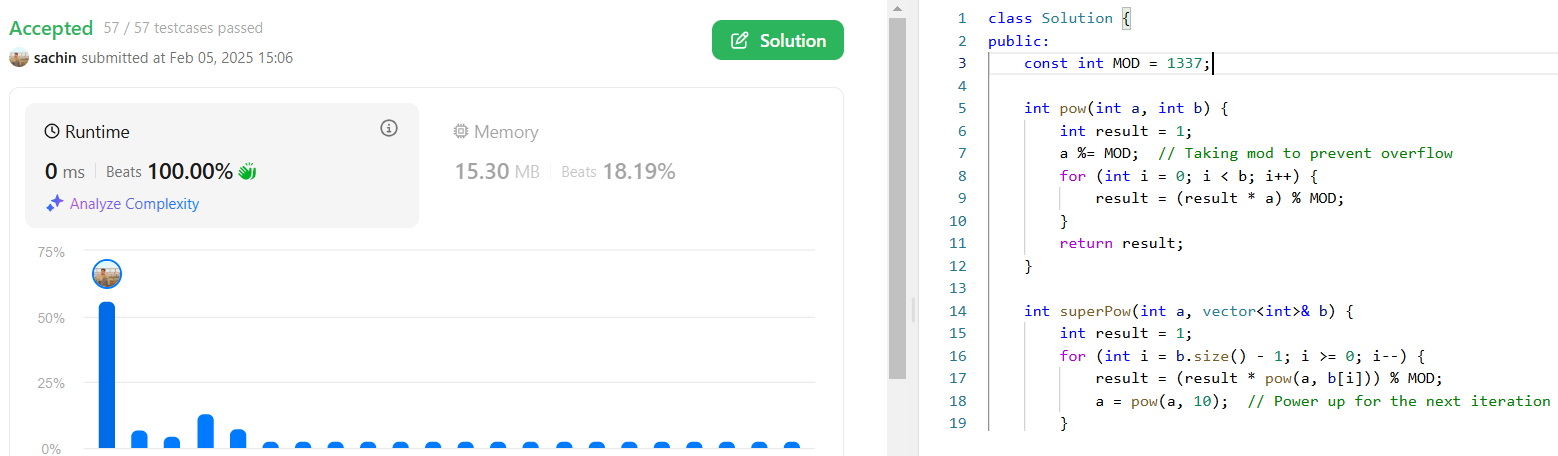
        }

        return result;

    }

};

**OUTPUT:**



1. **932.**[**Beautiful Array**](https://leetcode.com/problems/beautiful-array/description/)**.**

An array nums of length n is beautiful if:

* nums is a permutation of the integers in the range [1, n].
* For every 0 <= i < j < n, there is no index k with i < k < j where 2 \* nums[k] == nums[i] + nums[j].

Given the integer n, return *any beautiful array*nums*of length*n. There will be at least one valid answer for the given n.

**CODE**:

class Solution {

public:

       vector<int> beautifulArray(int N) {

        vector<int> res = {1};

        while (res.size() < N) {

            vector<int> tmp;

            for (int i : res) if (i \* 2 - 1 <= N) tmp.push\_back(i \* 2 - 1);

            for (int i : res) if (i \* 2 <= N) tmp.push\_back(i \* 2);

            res = tmp;

        }

        return res;

    }

};

**OUTPUT:**

****

1. **218.**[**The Skyline Problem**](https://leetcode.com/problems/the-skyline-problem/description/)**.**

A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return *the skyline formed by these buildings collectively*.

**CODE:**

class Solution {

public:

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

vector<vector<int>> ans;

multiset<int> pq{0};

vector<pair<int, int>> points;

for(auto b: buildings){

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

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

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

int ongoingHeight = 0;

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

int currentPoint = points[i].first;

int heightAtCurrentPoint = points[i].second;

if(heightAtCurrentPoint < 0){

pq.insert(-heightAtCurrentPoint);

} else {

pq.erase(pq.find(heightAtCurrentPoint)); }

auto pqTop = \*pq.rbegin();

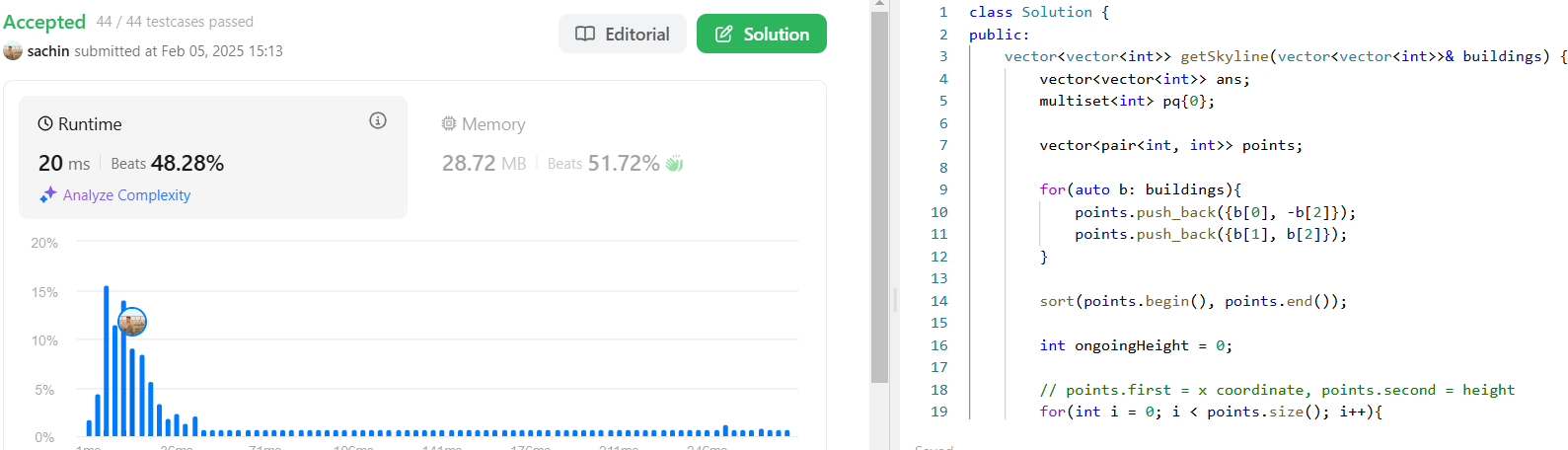
if(ongoingHeight != pqTop){

ongoingHeight = pqTop;

ans.push\_back({currentPoint, ongoingHeight});}}

return ans;}};

**OUTPUT:**

****

1. **493.**[**Reverse Pairs**](https://leetcode.com/problems/reverse-pairs/description/)**.**

Given an integer array nums, return *the number of reverse pairs in the array*.

**CODE:**

class Solution {

private:

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

int j = mid+1;

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

while(j<=high && nums[i] > 2\*(long long)nums[j]){

j++;

}

reversePairsCount += j-(mid+1);

}

int size = high-low+1;

vector<int> temp(size, 0);

int left = low, right = mid+1, k=0;

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

if(nums[left] < nums[right]){

temp[k++] = nums[left++];}

else{

temp[k++] = nums[right++];}

}

while(left<=mid){

temp[k++] = nums[left++]; }

while(right<=high){

temp[k++] = nums[right++];

}

int m=0;

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

nums[i] = temp[m++]; }}

void mergeSort(vector<int>& nums, int low, int high, int& reversePairsCount){

if(low >= high){

return;

}

int mid = (low + high) >> 1;

mergeSort(nums, low, mid, reversePairsCount);

mergeSort(nums, mid+1, high, reversePairsCount);

merge(nums, low, mid, high, reversePairsCount);

public:

int reversePairs(vector<int>& nums) {

int reversePairsCount = 0;

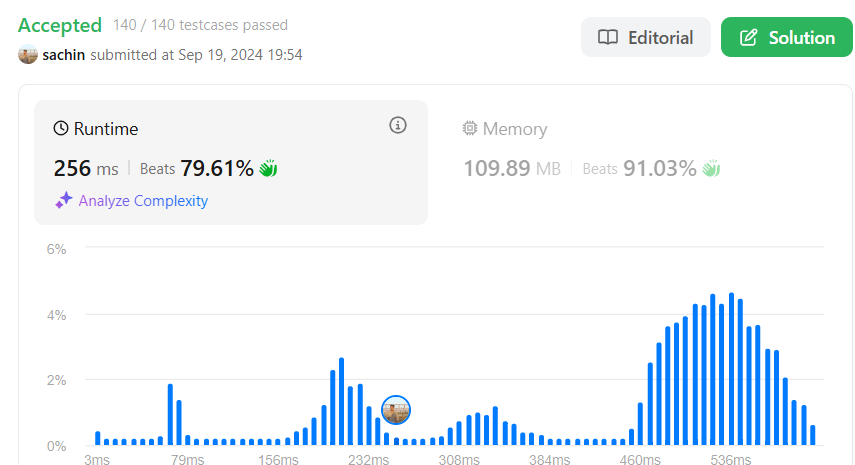
mergeSort(nums, 0, nums.size()-1, reversePairsCount);

return reversePairsCount;

}

};

**OUTPUT**:



1. 2407.[Longest Increasing Subsequence II](https://leetcode.com/problems/longest-increasing-subsequence-ii/description/).

You are given an integer array nums and an integer k.

Find the longest subsequence of nums that meets the following requirements:

**CODE:**

vector<int> seg;

    void upd(int ind, int val, int x, int lx, int rx) {

        if(lx == rx) {

            seg[x] = val;

            return;

        int mid = lx + (rx - lx) / 2;

        if(ind <= mid)

            upd(ind, val, 2 \* x + 1, lx, mid);

        else

            upd(ind, val, 2 \* x + 2, mid + 1, rx);

        seg[x] = max(seg[2 \* x + 1], seg[2 \* x + 2]);

    int query(int l, int r, int x, int lx, int rx) {

        if(lx > r or rx < l) return 0;

        if(lx >= l and rx <= r) return seg[x];

        int mid = lx + (rx - lx) / 2;

        return max(query(l, r, 2 \* x + 1, lx, mid), query(l, r, 2 \* x + 2, mid + 1, rx));

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

        int x = 1;

        while(x <= 200000) x \*= 2;

        seg.resize(2 \* x, 0);

        int res = 1;

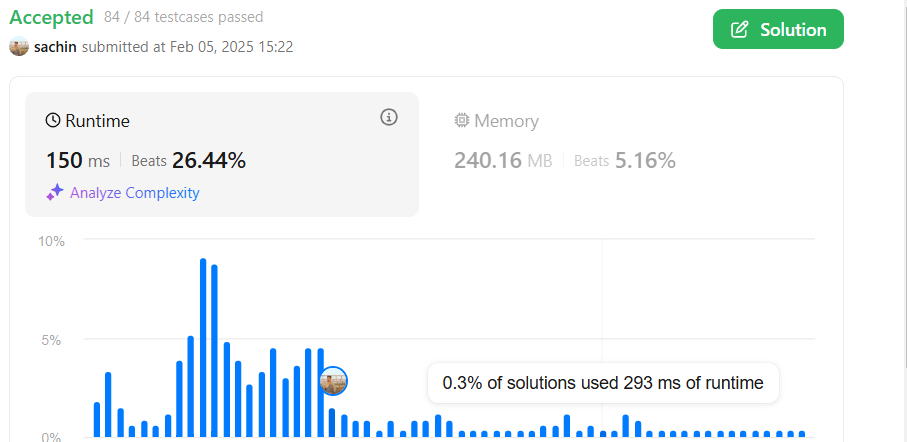
        for(int i = 0; i < nums.size(); ++i) {

            int left = max(1, nums[i] - k), right = nums[i] - 1;

            int q = query(left, right, 0, 0, x - 1); // check for the element in the range of [nums[i] - k, nums[i] - 1] wi

            res = max(res, q + 1);         return res;}};

**OUTPUT:**

****

1. **88.**[**Merge Sorted Array**](https://leetcode.com/problems/merge-sorted-array/description/)**.**

You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

**CODE:**class Solution {

public:

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

int i = m - 1;

int j = n - 1;

int k = m + n - 1;

while (i >= 0 && j >= 0) {

if (nums1[i] > nums2[j]) {

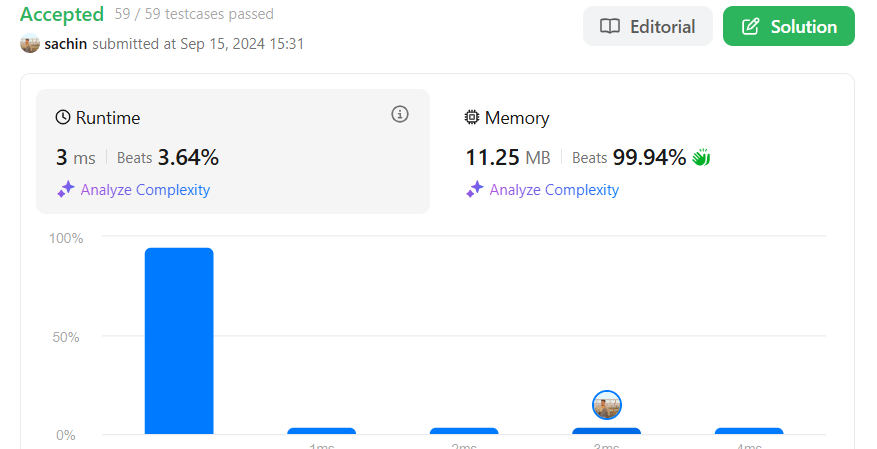
nums1[k--] = nums1[i--];

} else {

nums1[k--] = nums2[j--]; // Place the larger element from nums2 at the end of nums1 } }

while (j >= 0) {

nums1[k--] = nums2[j--]; }}};

**OUTPUT:** ****

1. **278.**[**First Bad Version**](https://leetcode.com/problems/first-bad-version/description/)**.**

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

**CODE:**

int firstBadVersion(int n) {

int first = 1;

int last = n;

while (first < last) {

int mid = first + (last - first) / 2;

if (isBadVersion(mid)) {

last = mid; // Mid could be the first bad version, so narrow the

// range to the left half.

} else {

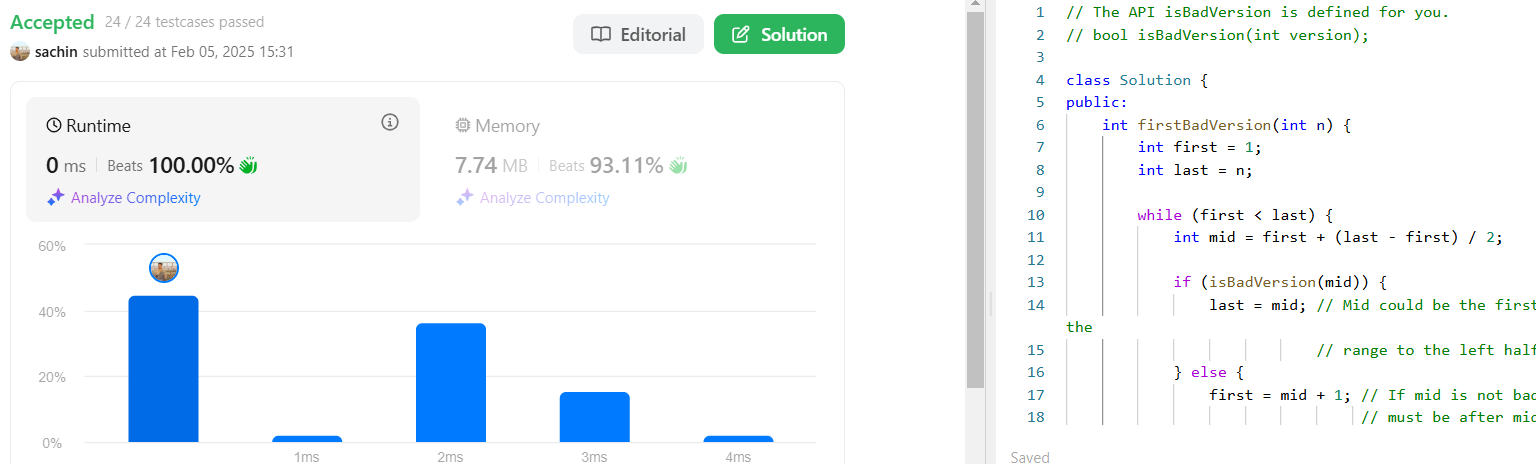
first = mid + 1; // If mid is not bad, the first bad version

} }

return first; // At the end, first will be the first bad version.

}

**OUTPUT:**

****

1. **75.**[**Sort Colors**](https://leetcode.com/problems/sort-colors/description/)**.**

Given an array nums with n objects colored red, white, or blue, sort them [in-place](https://en.wikipedia.org/wiki/In-place_algorithm) so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

**CODE:**

void sortColors(vector<int>& nums) {

int arr[3]={0};

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

{

arr[nums[i]]++;

}

nums.clear();

for(int i=0;i<3;i++)

{

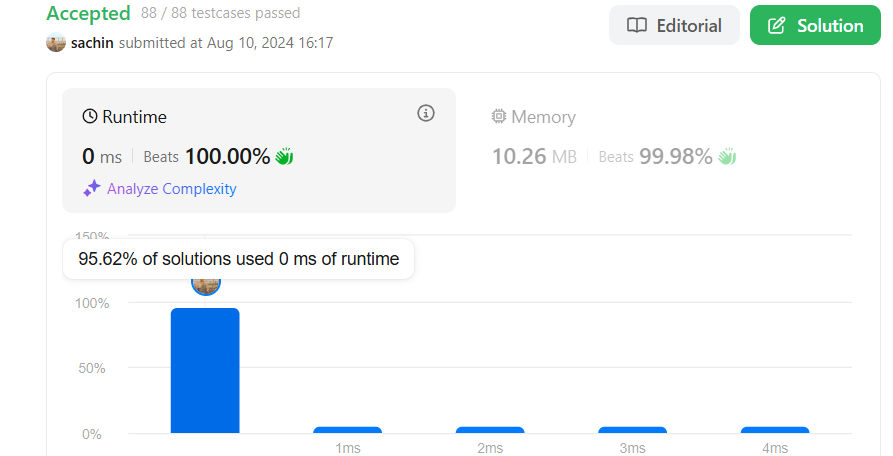
while(arr[i])

{

nums.push\_back(i);

arr[i]--;}}}

**OUTPUT:**



1. 162.[Find Peak Element](https://leetcode.com/problems/find-peak-element/description/)

A peak element is an element that is strictly greater than its neighbors.

Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.

You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in O(log n) time.

**CODE:**

class Solution {

public:

int findPeakElement(vector<int>& nums) {

int n=nums.size();

if(n==1) return 0;

if(nums[0]>nums[1]) return 0;

if(nums[n-2]<nums[n-1]) return n-1;

int l=n-2;

int s=1;

while(s<=l){

int mid=s+(l-s)/2;

if(nums[mid-1]<nums[mid] && nums[mid]>nums[mid+1]){

return mid;

}

else if(nums[mid]<nums[mid+1]){

s=mid+1;

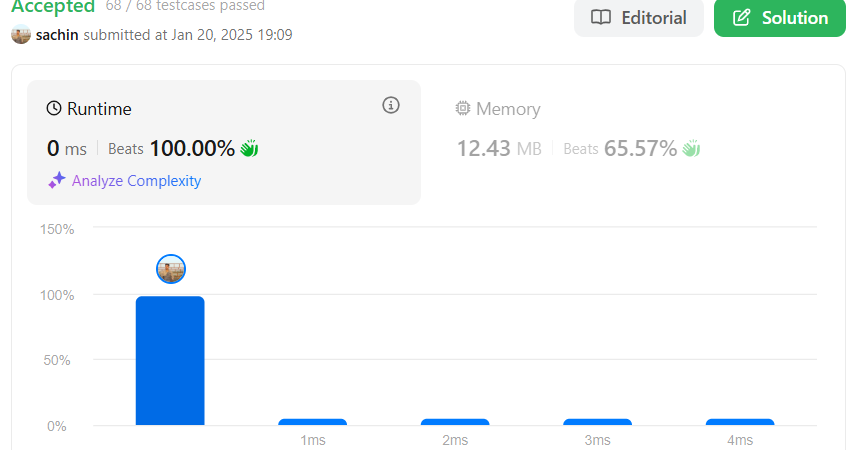
}

else{

l=mid-1;}

RETURN -1}};

**OUTPUT:**

****

1. **56.**[**Merge Intervals**](https://leetcode.com/problems/merge-intervals/description/)**.**

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

**CODE:**

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

vector<vector<int>>ans;

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

int r=intervals.size();

for(int i=0;i<r;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]);

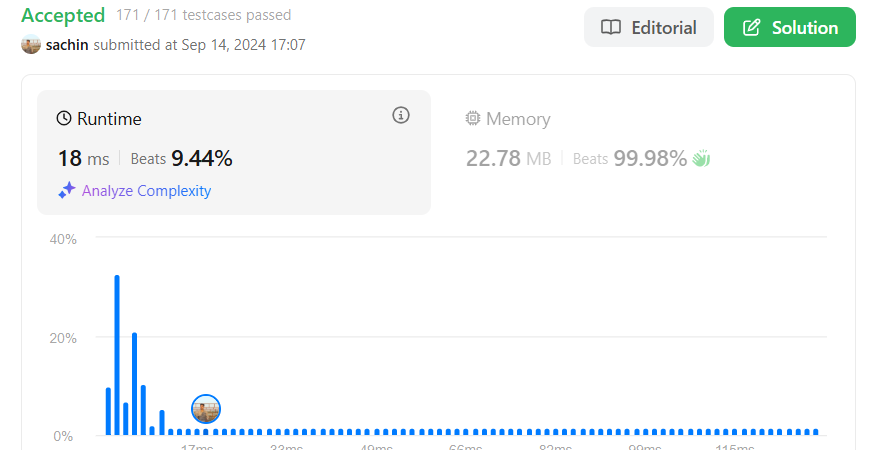
}

}

return ans;

}

**OUTPUT:**

****