AP LAB EXPERIMENT -2

1. **Longest Nice string:**

**Problem Statement:** 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:

    bool check(string s){

        vector<int> t1(26, 0), t2(26, 0);

        for(int i=0; i<s.length(); i++) if(s[i] >= 'a' && s[i] <= 'z') t1[s[i] - 'a']++;

        for(int i=0; i<s.length(); i++) if(s[i] >= 'A' && s[i] <= 'Z') t2[s[i] - 'A']++;

         for(int i=0; i<26; i++){

            if(t1[i] == 0 && t2[i] == 0) continue;

            else if(t1[i] == 0 && t2[i] > 0) return false;

            else if(t1[i] > 0 && t2[i] == 0) return false;

        }

        return true;

    }

    string longestNiceSubstring(string s) {

        if(s == "") return "";

        if(check(s)) return s;

        unordered\_map<char,int> m;

        vector<int> t1(26, 0), t2(26, 0);

        for(int i=0; i<s.length(); i++) if(s[i] >= 'a' && s[i] <= 'z') t1[s[i] - 'a']++;

        for(int i=0; i<s.length(); i++) if(s[i] >= 'A' && s[i] <= 'Z') t2[s[i] - 'A']++;

        for(int i=0; i<26; i++){

            if(t1[i] == 0 && t2[i] == 0) continue;

            else if(t1[i] == 0 && t2[i] > 0) m[i + 'A']++;

            else if(t1[i] > 0 && t2[i] == 0) m[i + 'a']++;

        }

        vector<string> ans; string temp = "";

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

            if(m.find(s[i]) == m.end()) temp += s[i];

            else {

                ans.push\_back(temp); temp = "";

            }

        }

        if(temp != "") ans.push\_back(temp);

        string kk = "";

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

            if(ans[i] == "") continue;

            string t = longestNiceSubstring(ans[i]);

            if(t == "") continue;

            if(kk == "" || kk.length() < t.length()) kk = t;

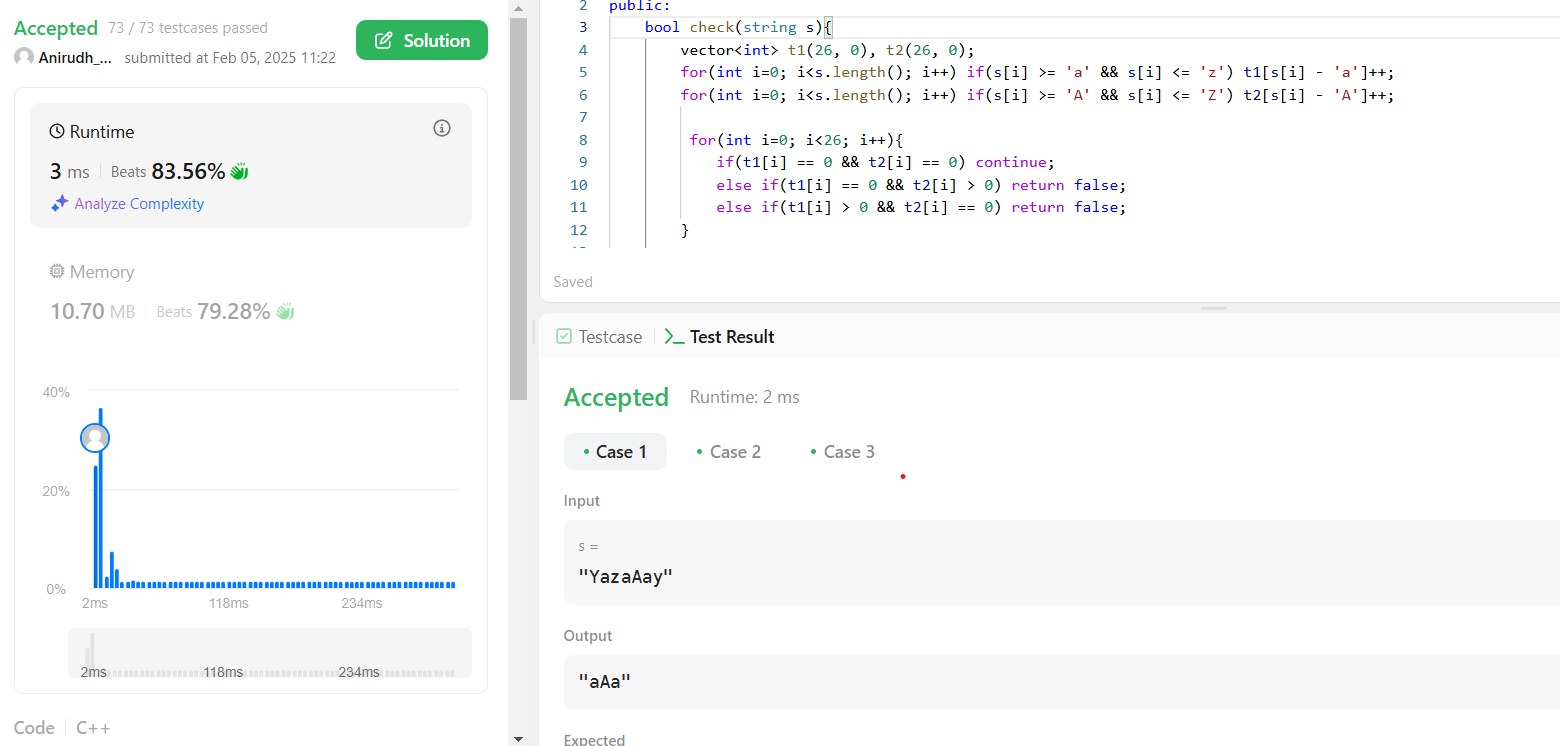
        }

        return kk;

    }

};

**OUTPUT:**

****

1. **REVERSE BITS**

**Problem Statement: R**everse bits of a given 32 bits unsigned integer.

Note:

* Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
* In Java, the compiler represents the signed integers using [2's complement notation](https://en.wikipedia.org/wiki/Two%27s_complement). Therefore, in Example 2 above, the input represents the signed integer -3 and the output represents the signed integer

CODE:

class Solution {

public:

    uint32\_t reverseBits(uint32\_t n) {

        uint32\_t ans = 0;

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

            ans <<= 1;

            ans |= (n & 1);

            n >>= 1;

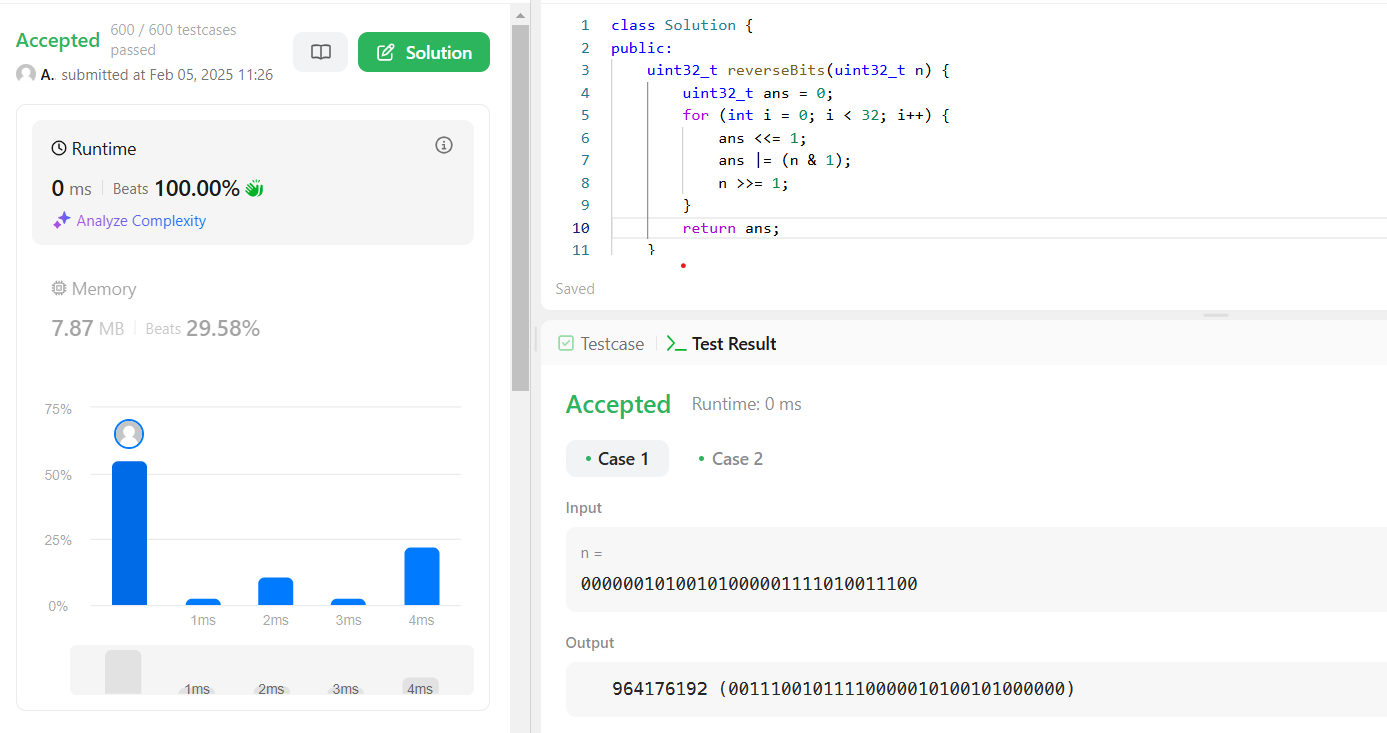
        }

        return ans;

    }

};

OUTPUT:



1. MAXIMUM SUBARRAY:

Problem Statement: Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

CODE:

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int res = nums[0];

int total = 0;

for (int n : nums) {

if (total < 0) {

total = 0;

}

total += n;

res = max(res, total);

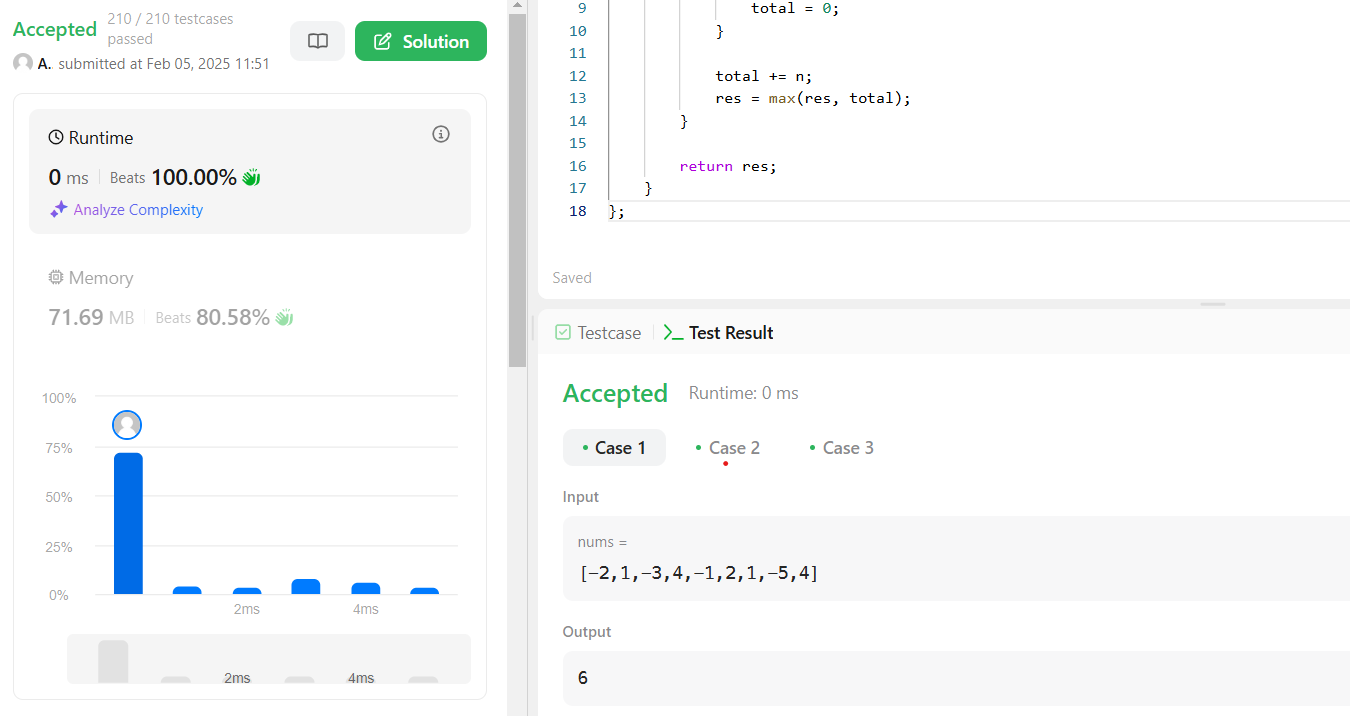
}

return res;

}

};

OUTPUT:



1. SEARCH A 2D MATRIX:

PROBLEM STATEMENT: 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:

class Solution {

public:

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

        int cols = matrix[0].size() - 1;

        int n = matrix.size() - 1;

        int rows = 0;

        while(rows <= n && cols >= 0){

            int toCompare = matrix[rows][cols];

            if(toCompare > target){

                cols--;

            }else if(toCompare < target){

                rows++;

            }else{

                return true;

            }

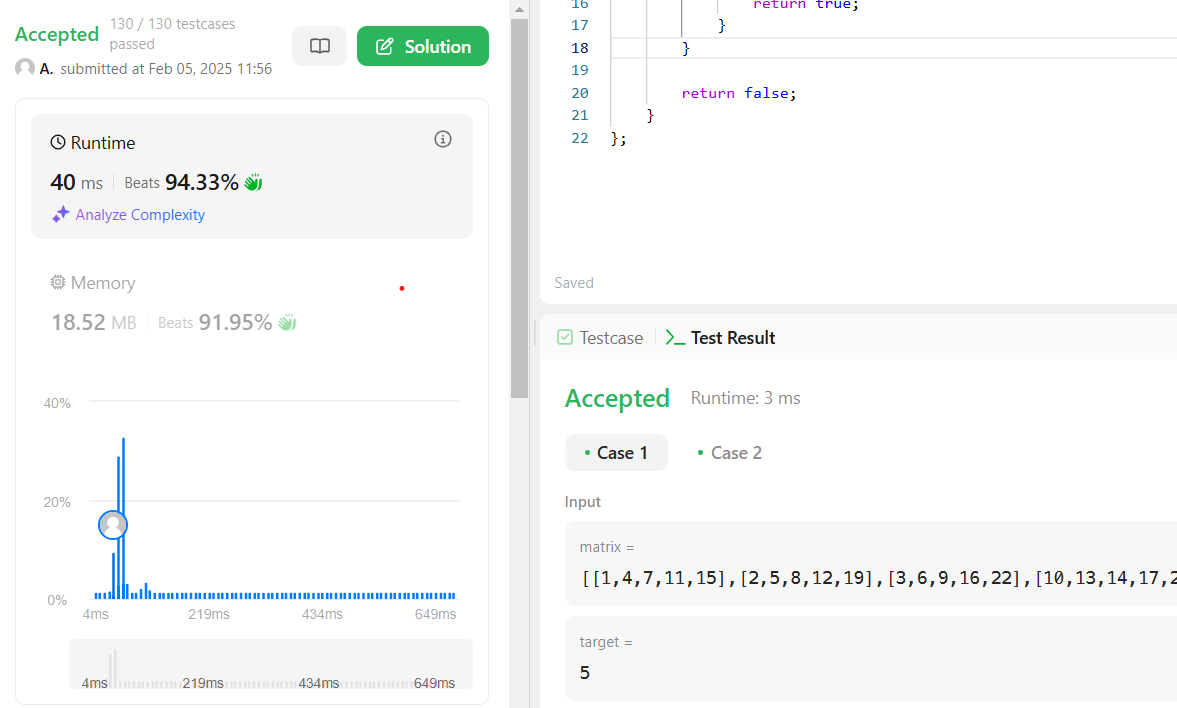
        }

        return false;

    }

};

OUTPUT:



1. SUPER POW:

PROBLEM STATEMENT: 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 {

const int base = 1337;

int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10

{

a %= base;

int result = 1;

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

result = (result \* a) % base;

return result;

}

public:

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

if (b.empty()) return 1;

int last\_digit = b.back();

b.pop\_back();

return powmod(superPow(a, b), 10) \* powmod(a, last\_digit) % base;

}

};

OUTPUT:



1. MEDIAN OF TWO SORTED ARRAYS:

Problem Statement: Given two sorted arrays nums1 and nums2 of size m and n respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

CODE:

class Solution {

public:

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

        int n1 = nums1.size(), n2 = nums2.size();

        if (n1 > n2)

            return findMedianSortedArrays(nums2, nums1);

        int n = n1 + n2;

        int left = (n1 + n2 + 1) / 2;

        int low = 0, high = n1;

        while (low <= high) {

            int mid1 = (low + high) >> 1;

            int mid2 = left - mid1;

            int l1 = INT\_MIN, l2 = INT\_MIN, r1 = INT\_MAX, r2 = INT\_MAX;

            if (mid1 < n1)

                r1 = nums1[mid1];

            if (mid2 < n2)

                r2 = nums2[mid2];

            if (mid1 - 1 >= 0)

                l1 = nums1[mid1 - 1];

            if (mid2 - 1 >= 0)

                l2 = nums2[mid2 - 1];

            if (l1 <= r2 && l2 <= r1) {

                if (n % 2 == 1)

                    return max(l1, l2);

                else

                    return ((double)(max(l1, l2) + min(r1, r2))) / 2.0;

            }

            else if (l1 > r2) {

                high = mid1 - 1;

            }

            else {

                low = mid1 + 1;

            }

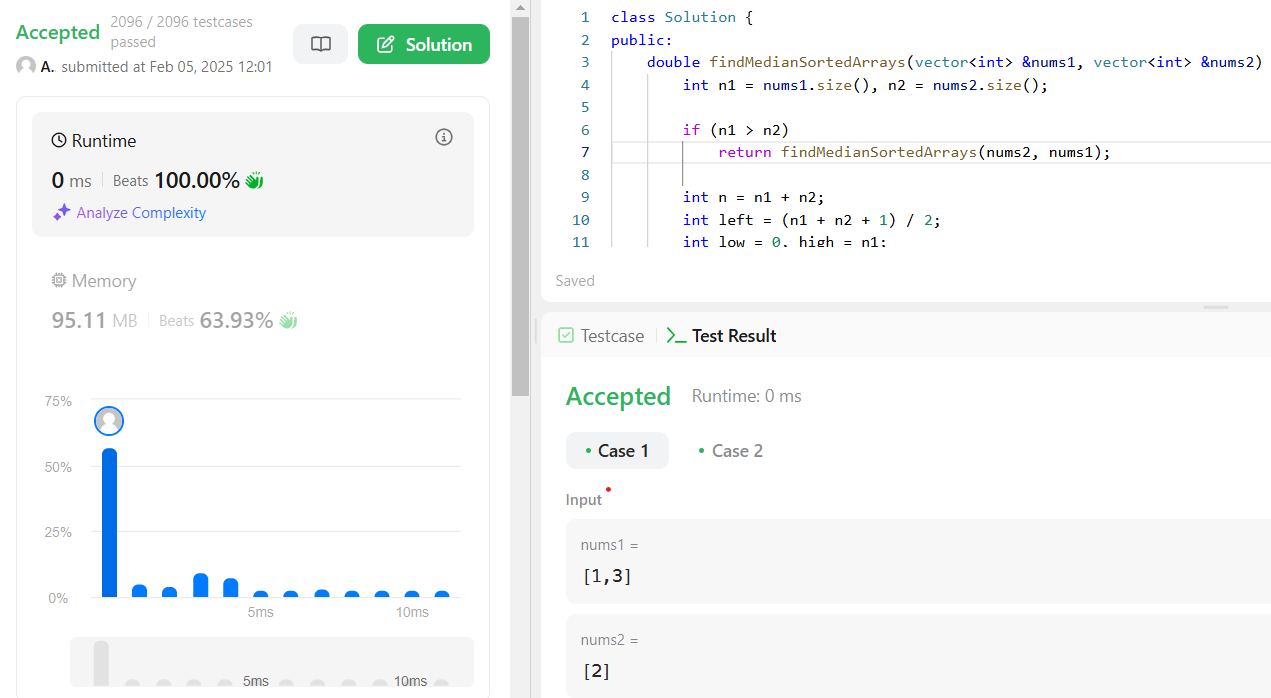
        }

        return 0;

    }

};

OUTPUT:



1. SEARCH IN SORTED ARRAY:  
   Problem Statement: There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums **after** the possible rotation and an integer target, return *the index of*target*if it is in*nums*, or*-1*if it is not in*nums.

You must write an algorithm with O(log n) runtime complexity.

CODE:

class Solution {

public:

    int binarySearch(vector<int>& arr, int s, int e, int target) {

        while (s <= e) {

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

            if (arr[mid] == target) {

                return mid;

            } else if (arr[mid] > target) {

                e = mid - 1;

            } else {

                s = mid + 1;

            }

        }

        return -1;

    }

    int pivot(vector<int>& nums, int size) {

        int s = 0, e = size - 1;

        while (s < e) {

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

            if (nums[mid] >= nums[0]) {

                s = mid + 1;

            } else {

                e = mid;

            }

        }

        return s;

    }

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

        int p = pivot(nums, nums.size());

        int n = nums.size();

        if (target >= nums[p] && target <= nums[n - 1]) {

            return binarySearch(nums, p, n - 1, target);

        } else {

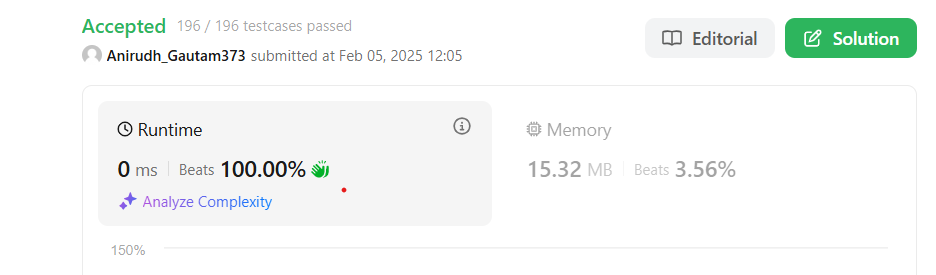
            return binarySearch(nums, 0, p - 1, target);

        }

    }

};

OUTPUT:



1. FIND PEAK ELEMENT:

Problem Statement: 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 left = 0;

        int right = nums.size() - 1;

        while (left < right) {

            int mid = left + (right - left) / 2;

            if (nums[mid] > nums[mid + 1]) {

                right = mid;

            } else {

                left = mid + 1;

            }

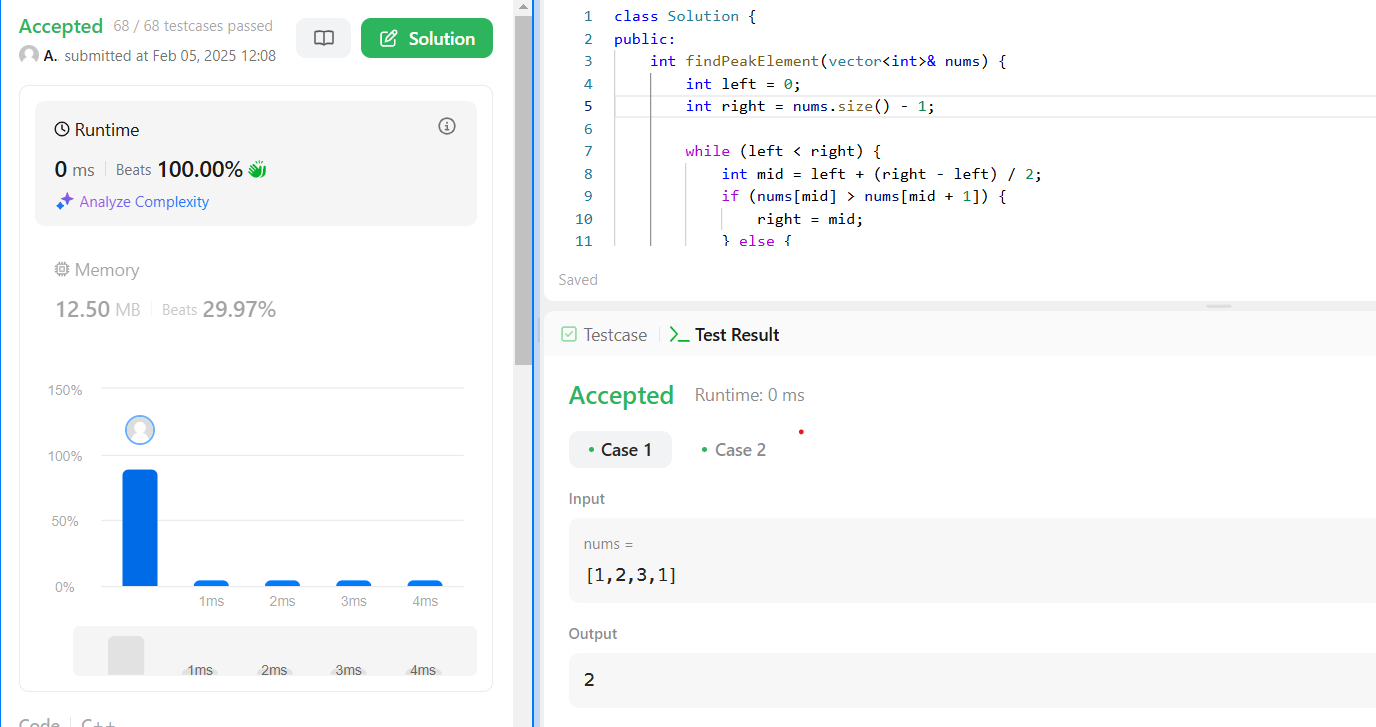
        }

        return left;

    }

};

OUTPUT:



1. Kth LARGEST ELEMENT IN ARRAY:

Problem Statemnt: Given an integer array nums and an integer k, return *the* kth *largest element in the array*.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

CODE:

class Solution {

public:

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

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

        for (int num : nums) {

            minHeap.push(num);

            if (minHeap.size() > k) minHeap.pop();

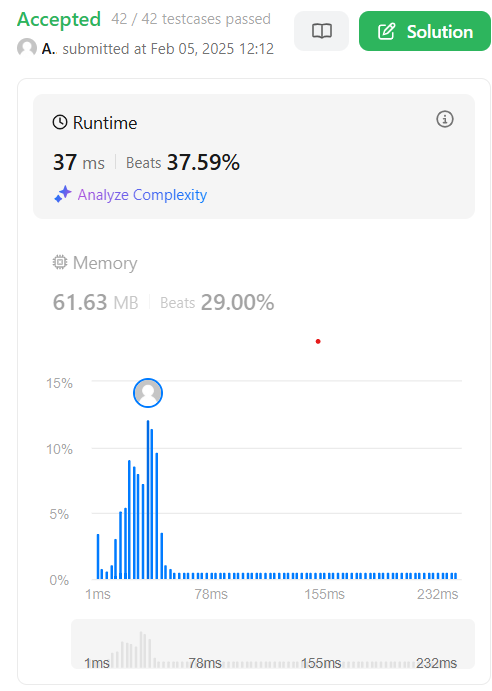
        }

        return minHeap.top();

    }

};

OUTPUT:



1. TOP K FREQUENT ELEMENTS:

Problem Statement: Given an integer array nums and an integer k, return *the* k *most frequent elements*. You may return the answer in **any order**.

CODE:

class Solution {

public:

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

        unordered\_map<int, int> counter;

        for (int n : nums) {

            counter[n]++;

        }

        auto comp = [](pair<int, int>& a, pair<int, int>& b) {

            return a.second < b.second;

        };

        priority\_queue<pair<int, int>, vector<pair<int, int>>, decltype(comp)> heap(comp);

        for (auto& entry : counter) {

            heap.push({entry.first, entry.second});

        }

        vector<int> res;

        while (k-- > 0) {

            res.push\_back(heap.top().first);

            heap.pop();

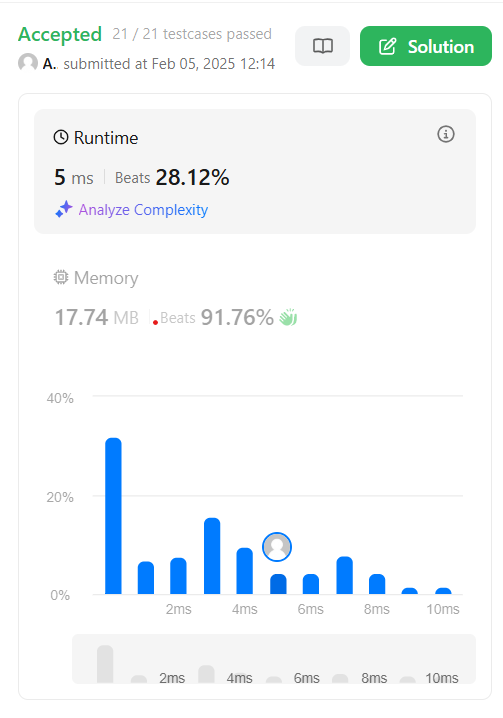
        }

        return res;

    }

};

OUTPUT:



1. MERGE ELEMENTS:

Problem Statement: 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:

class Solution {

public:

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

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

            return a[0] < b[0];

        });

        vector<vector<int>> merged;

        vector<int> prev = intervals[0];

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

            vector<int> interval = intervals[i];

            if (interval[0] <= prev[1]) {

                prev[1] = max(prev[1], interval[1]);

            } else {

                merged.push\_back(prev);

                prev = interval;

            }

        }

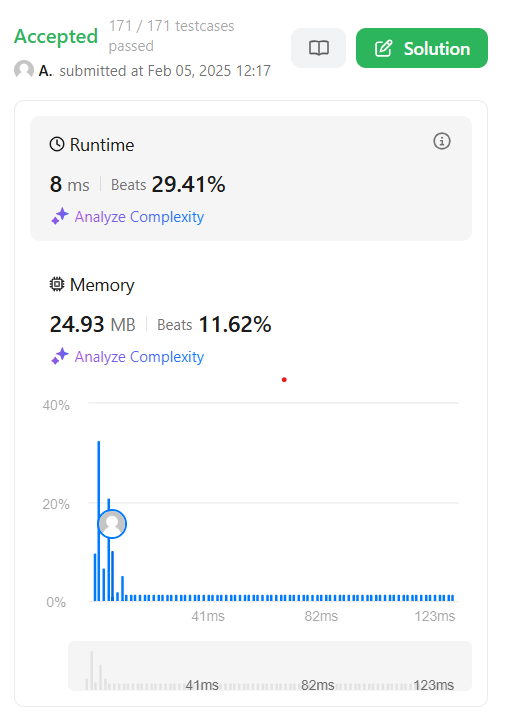
        merged.push\_back(prev);

        return merged;

    }

};

OUTPUT:



1. **SORT COLORS:**

**Problem Statement:** 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:

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int c0 = 0, c1 = 0, c2 = 0;

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

            if(nums[i] == 0)

            c0++;

            else if(nums[i] == 1)

            c1++;

            else

            c2++;

        }

        for(int i = 0; i < c0; i++){

            nums[i] = 0;

        }

        for(int i = c0; i < c0 + c1; i++){

            nums[i] = 1;

        }

        for(int i = c0+c1; i < c0 + c1 + c2; i++){

            nums[i] = 2;

        }

    }

};

OUTPUT:

