# Assignment-1

**Student Name: Tanvi Chhabra UID: 22BCS15679**

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

**Semester: 6th DOP: 05-02-25**

**Subject: Advanced Programming Subject Code: 22CSH-351**

**1.Question:**



**Code:**

class Solution {

    public String longestNiceSubstring(String s) {

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

        char[] arr = s.toCharArray();

        Set<Character> set = new HashSet<>();

        for (char c: arr) set.add(c);

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

            char c = arr[i];

            if (set.contains(Character.toUpperCase(c)) && set.contains(Character.toLowerCase(c))) continue;

            String sub1 = longestNiceSubstring(s.substring(0, i));

            String sub2 = longestNiceSubstring(s.substring(i+1));

            return sub1.length() >= sub2.length() ? sub1 : sub2;

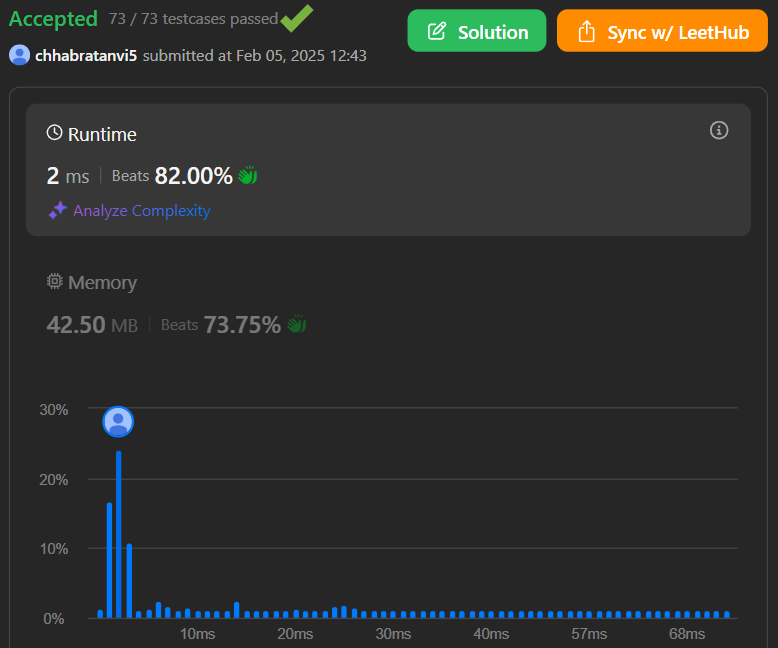
        }

        return s;

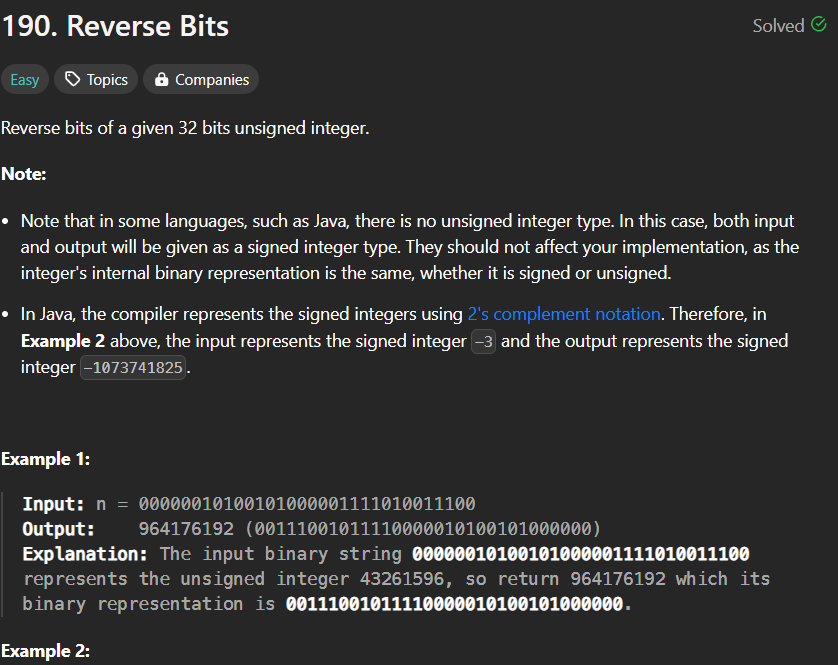
    }

}

**Output:**



**2.Question:**



**Code:**

public class Solution {

// you need treat n as an unsigned value

public int reverseBits(int n) {

int result=0;

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

result<<=1;

result|=(n&1);

n>>>=1;

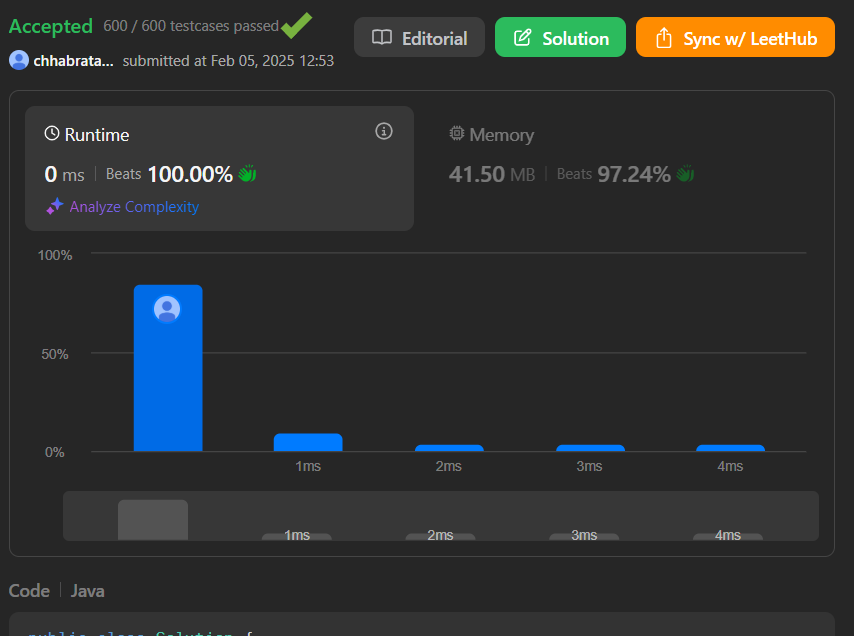
}

return result;

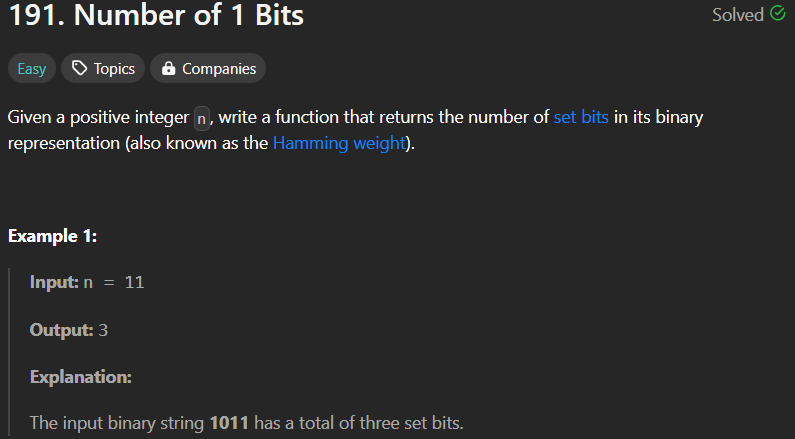
}

}

**Output:**



**3.Question:**

****

**Code:**

public static int hammingWeight(int n) {

int ones = 0;

while(n!=0) {

ones = ones + (n & 1);

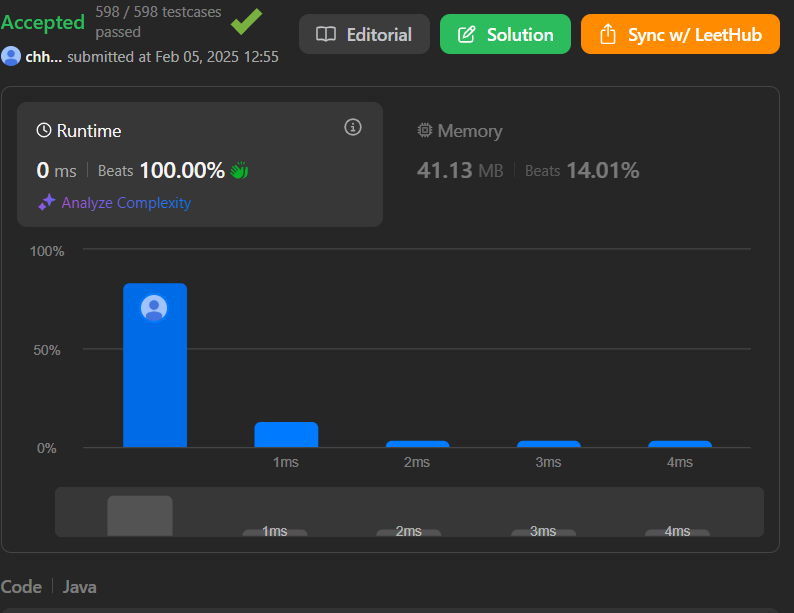
n = n>>>1;

}

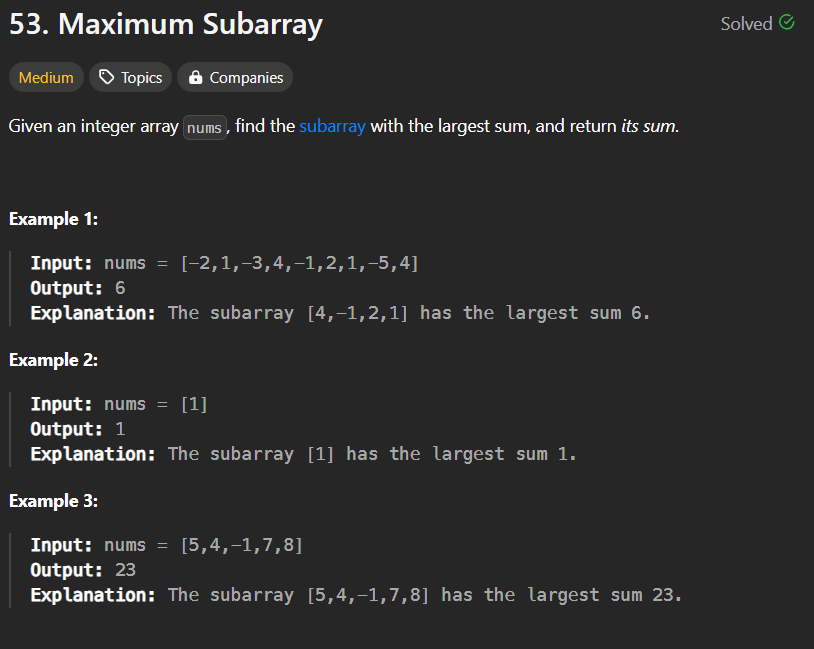
return ones;

}

**Output:**



**4.Question:**



**Code:**

class Solution {

public int maxSubArray(int[] nums) {

int maxSum = Integer.MIN\_VALUE;

int currentSum = 0;

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

currentSum += nums[i];

if (currentSum > maxSum) {

maxSum = currentSum;

}

if (currentSum < 0) {

currentSum = 0;

}

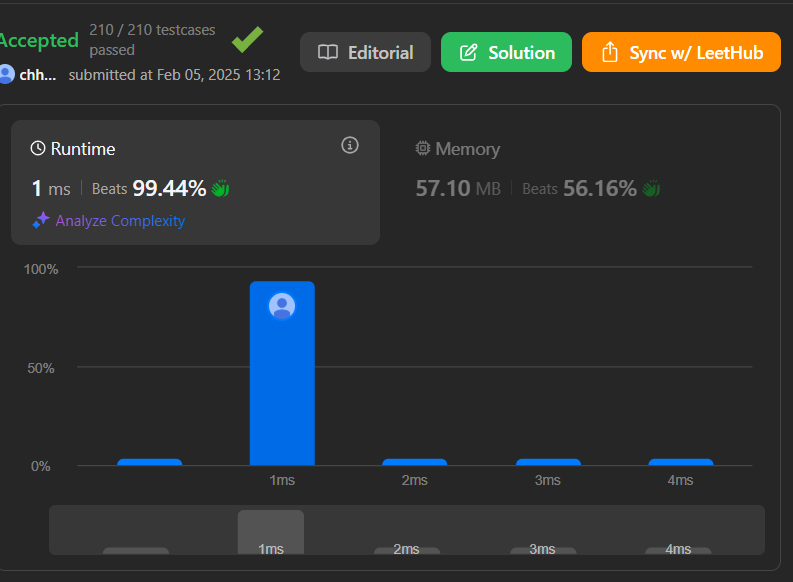
}

return maxSum;

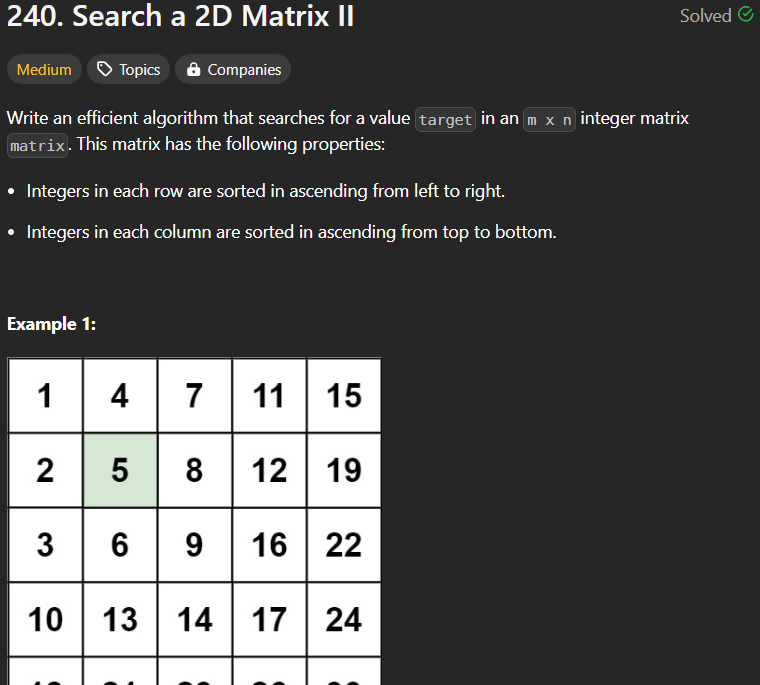
}

}

**Output:**



**5.Question:**



**Code:**

public class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

if(matrix == null || matrix.length < 1 || matrix[0].length <1) {

return false;

}

int col = matrix[0].length-1;

int row = 0;

while(col >= 0 && row <= matrix.length-1) {

if(target == matrix[row][col]) {

return true;

} else if(target < matrix[row][col]) {

col--;

} else if(target > matrix[row][col]) {

row++;

}

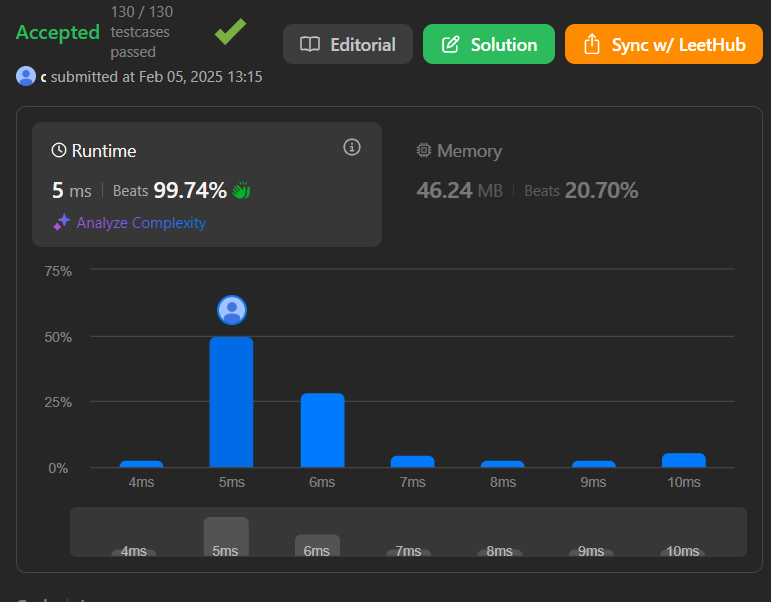
}

return false;

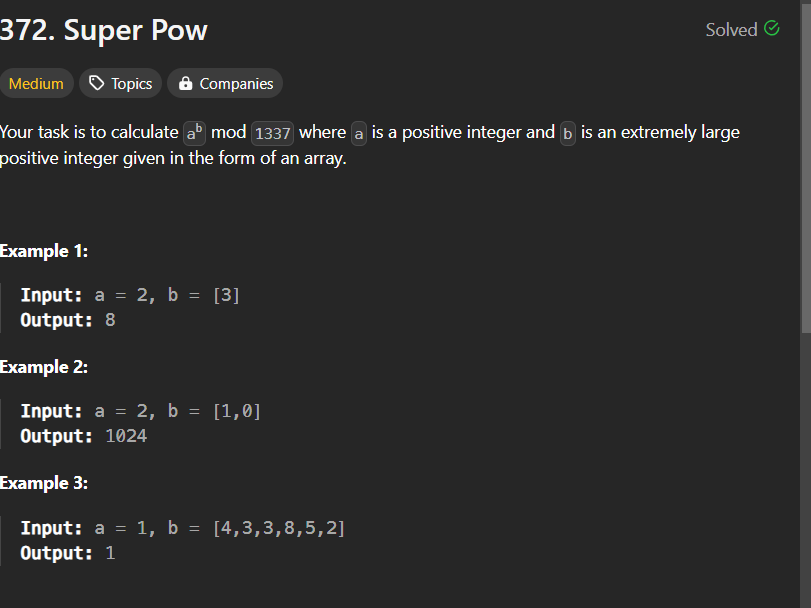
}

}

**Output:**



**6.Question:**



**Code:**

public int superPow(int a, int[] b) {

if (a % 1337 == 0) return 0;

int p = 0;

for (int i : b) p = (p \* 10 + i) % 1140;

if (p == 0) p += 1440;

return power(a, p, 1337);

}

public int power(int a, int n, int mod) {

a %= mod;

int ret = 1;

while (n != 0) {

if ((n & 1) != 0) ret = ret \* a % mod;

a = a \* a % mod;

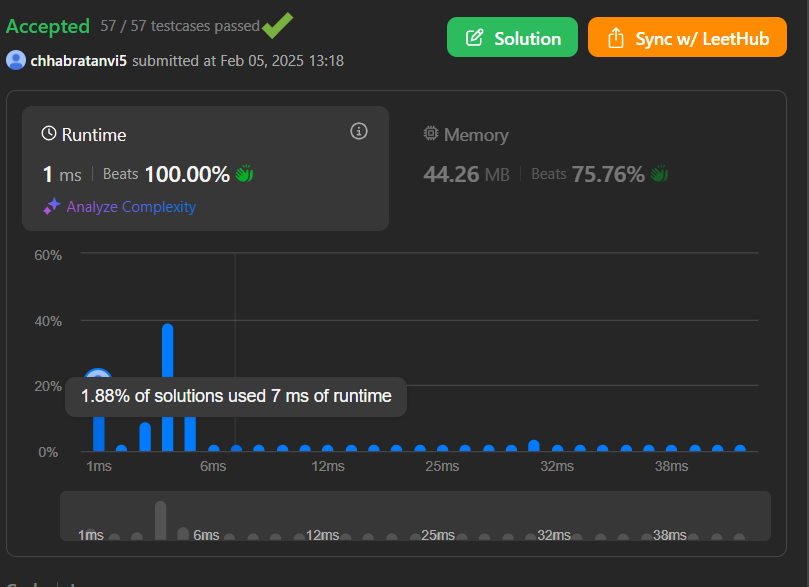
n >>= 1;

}

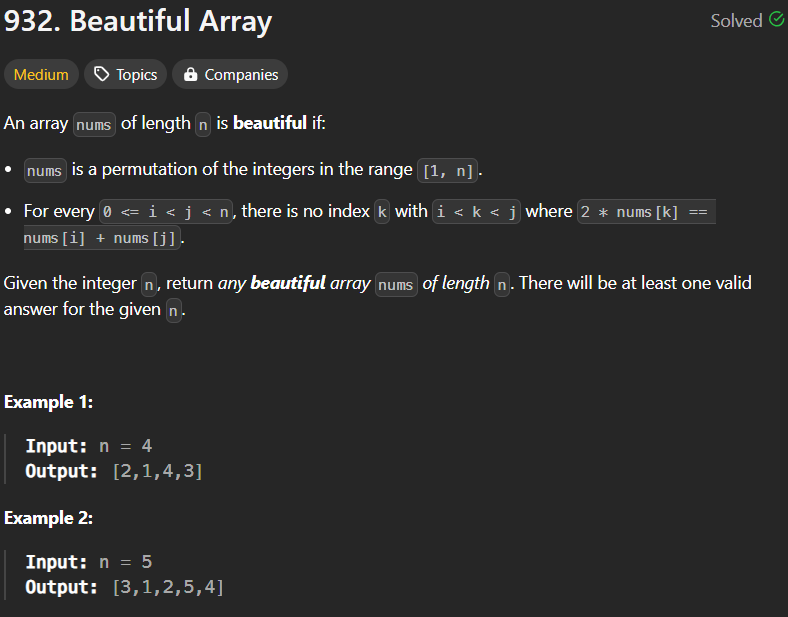
return ret;

}

**Output:**

****

**7. Question:**



**Code:**

class Solution {

public int[] beautifulArray(int n) {

int[] ans = new int[n];

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

ans[i] = i+1;

}

recursion(ans, 0, n-1);

return ans;

}

public void recursion(int[] arr, int left, int right){

if(left >= right)

return;

ArrayList<Integer> l = new ArrayList<>();

ArrayList<Integer> r = new ArrayList<>();

boolean alt = true;// Not worry about whether the factor of the interval is even or odd too much, they can be grouped by

// just picking one and skip one

for(int i = left; i <= right; i++){ // picking the elements and put them into the two groups

if(alt)

l.add(arr[i]);

else

r.add(arr[i]);

alt = !alt;

}

for(int i = left; i <= right; i++){ // merging them into the final array

if(!l.isEmpty())

arr[i] = l.remove(0);

else

arr[i] = r.remove(0);

}

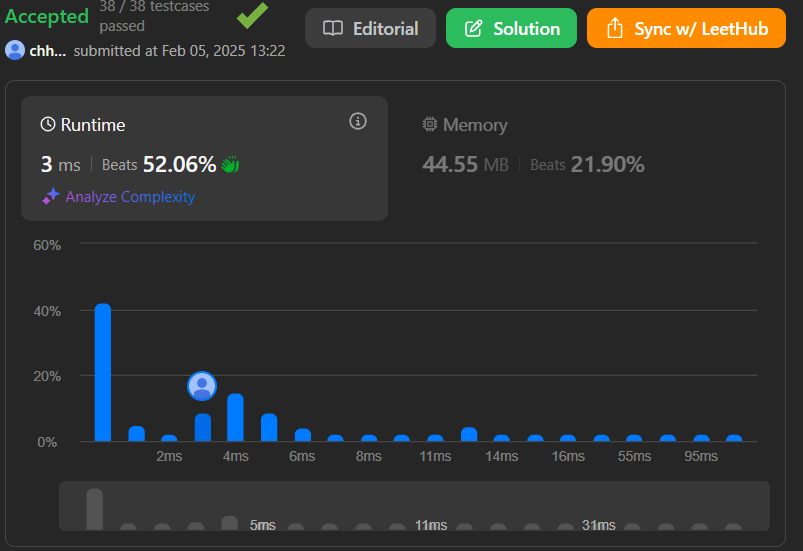
recursion(arr, left, (right+left)/2);

recursion(arr, (left+right)/2+1, right);

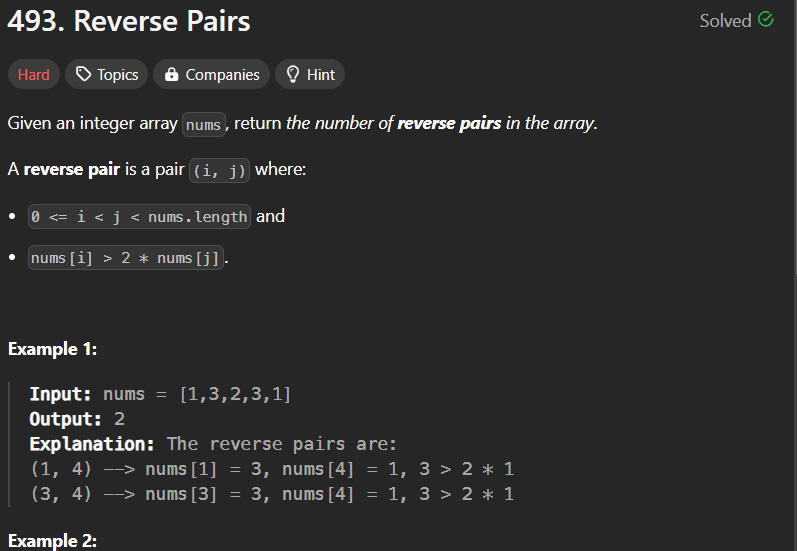
}

}

**Output:**



**8.Question:**



**Code:**

class Solution {

public void merge(int[] arr, int low, int mid, int high) {

ArrayList<Integer> temp = new ArrayList<>();

int left = low;

int right = mid+1;

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

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

temp.add(arr[left++]);

} else {

temp.add(arr[right++]);

}

}

while(left <= mid) temp.add(arr[left++]);

while(right <= high) temp.add(arr[right++]);

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

arr[i] = temp.get(i-low);

}

}

public int countPairs(int[] arr, int low, int mid, int high) {

int right = mid + 1;

int cnt = 0;

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

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

right++;

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

}

return cnt;

}

public int mergeSort(int[] arr, int low, int high) {

int cnt = 0;

if(low >= high) return cnt;

int mid = (low + high) / 2;

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

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

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

merge(arr,low,mid,high);

return cnt;

}

public int reversePairs(int[] nums) {

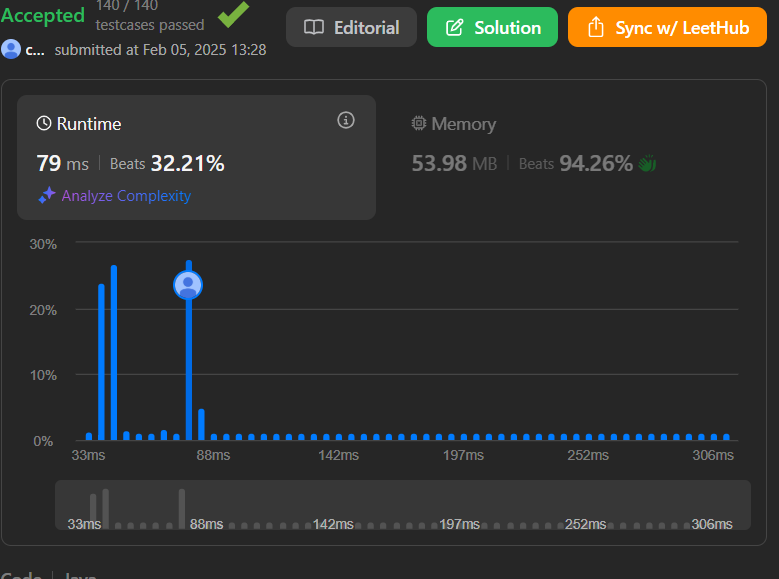
int n = nums.length;

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

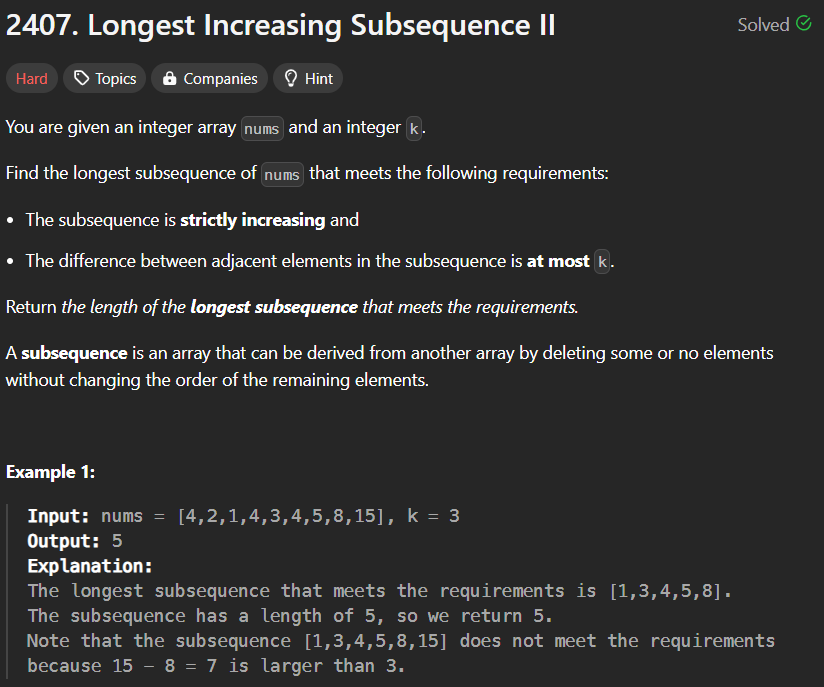
}

}

**Output:**



**9.Question:**



**Code:**

class Solution {

    int N = 100001;

    int[] seg = new int[2\*N];

    void update(int pos, int val){  // update max

        pos += N;

        seg[pos] = val;

        while (pos > 1) {

            pos >>= 1;

            seg[pos] = Math.max(seg[2\*pos], seg[2\*pos+1]);

        }

    }

    int query(int lo, int hi){ // query max [lo, hi)

        lo += N;

        hi += N;

        int res = 0;

        while (lo < hi) {

            if ((lo & 1)==1) {

                res = Math.max(res, seg[lo++]);

            }

            if ((hi & 1)==1) {

                res = Math.max(res, seg[--hi]);

            }

            lo >>= 1;

            hi >>= 1;

        }

        return res;

    }

    public int lengthOfLIS(int[] A, int k) {

        int ans = 0;

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

            int l = Math.max(0, A[i]-k);

            int r = A[i];

            int res = query(l, r) + 1; // best res for the current element

            ans = Math.max(res, ans);

            update(A[i], res); // and update it here

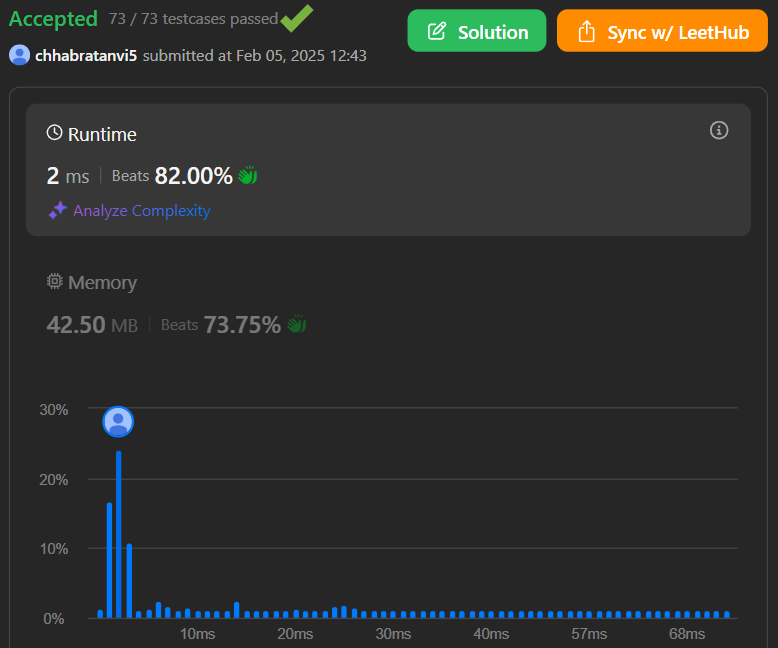
        }

        return ans;

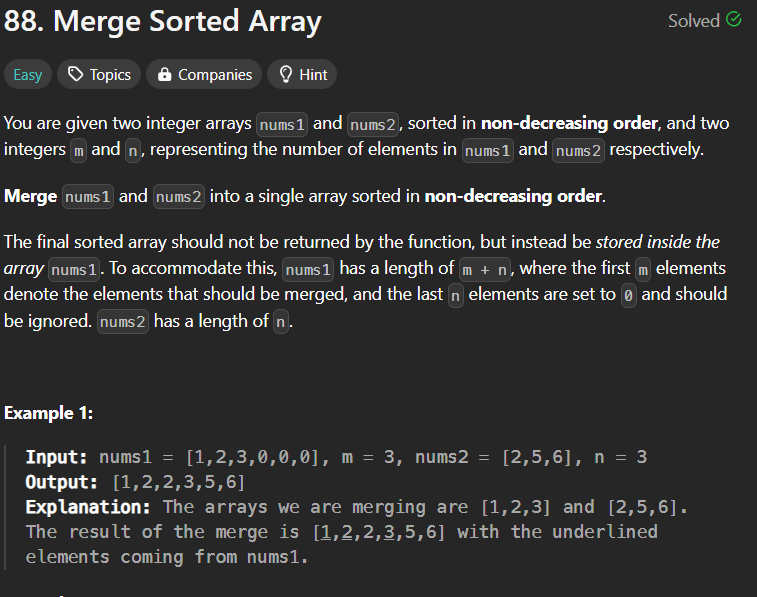
    }

}

**Output:**

****

**10.Question:**



**Code:**

class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

int ptr1 = m - 1, ptr2 = n - 1;

int indexPtr = nums1.length - 1;

while (ptr2 >= 0) {

if (ptr1 >= 0 && nums1[ptr1] > nums2[ptr2]) {

nums1[indexPtr--] = nums1[ptr1--];

} else {

nums1[indexPtr--] = nums2[ptr2--];

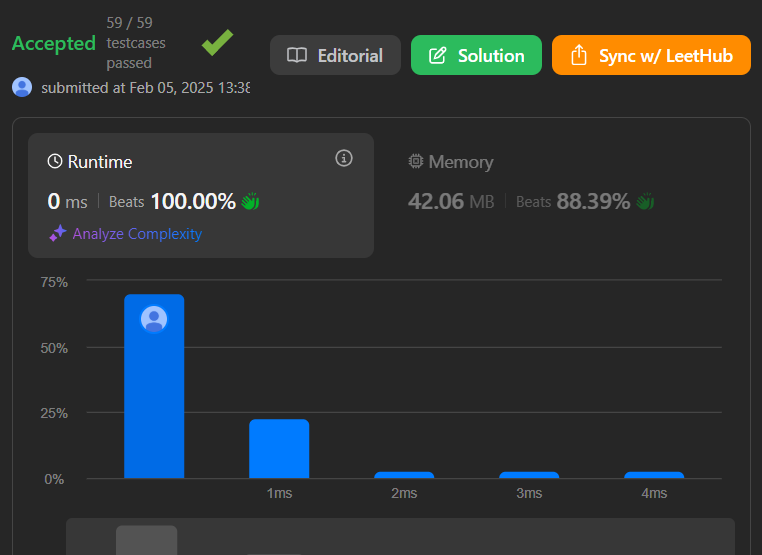
}

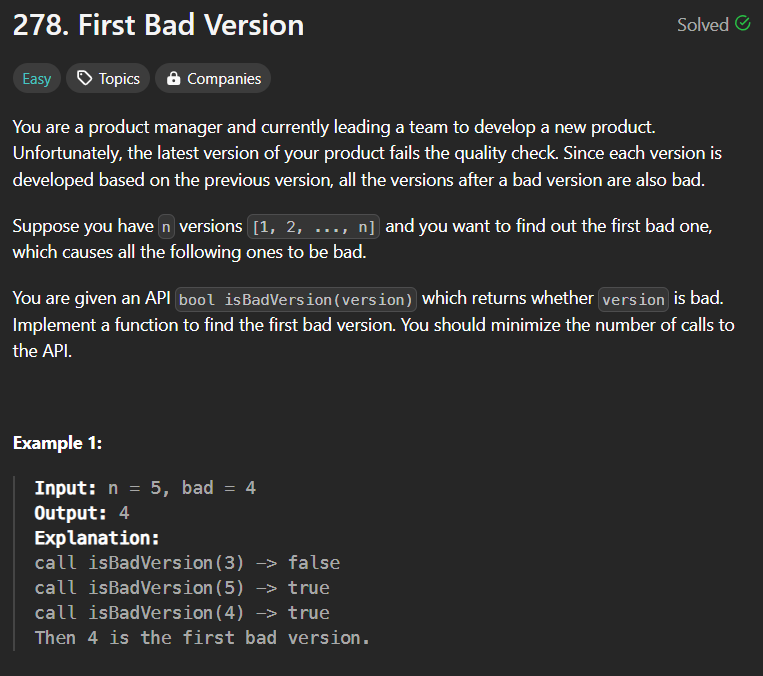
}

}

}

**Output:**



**11.Question:** 

**Code:**

public class Solution extends VersionControl {

public int firstBadVersion(int n) {

int left = 1;

int right = n;

while(left < right) {

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

if(isBadVersion(mid)) {

right = mid;

}

else {

left = mid + 1;

}

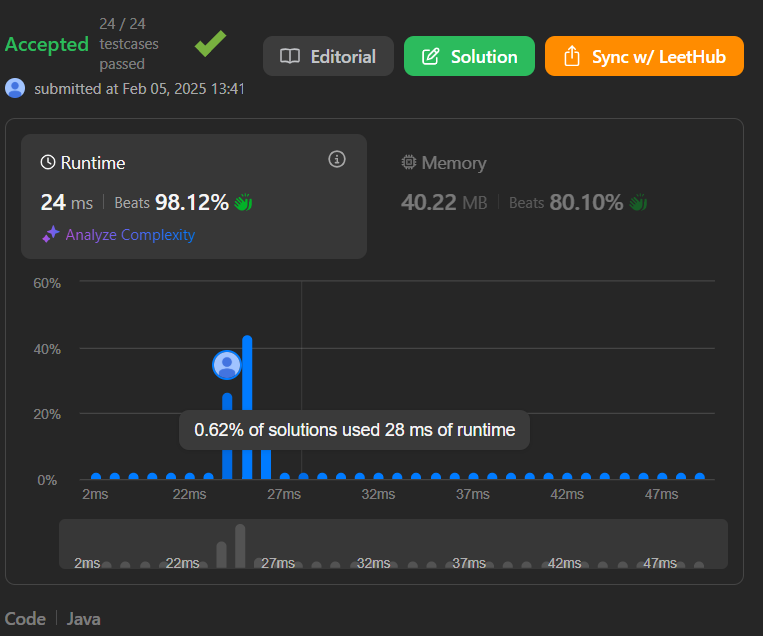
}

return left;

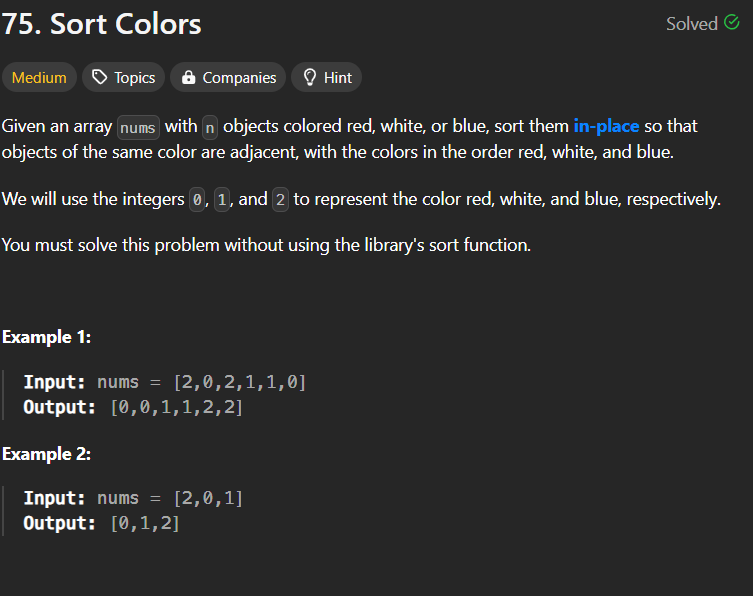
}

}

**Output:**



**12.Question:**



**Code:**

class Solution {

public void sortColors(int[] nums) {

HashMap<Integer, Integer> count = new HashMap<>();

count.put(0, 0);

count.put(1, 0);

count.put(2, 0);

for (int num : nums) {

count.put(num, count.get(num) + 1);

}

int idx = 0;

for (int color = 0; color < 3; color++) {

int freq = count.get(color);

for (int j = 0; j < freq; j++) {

nums[idx] = color;

idx++;

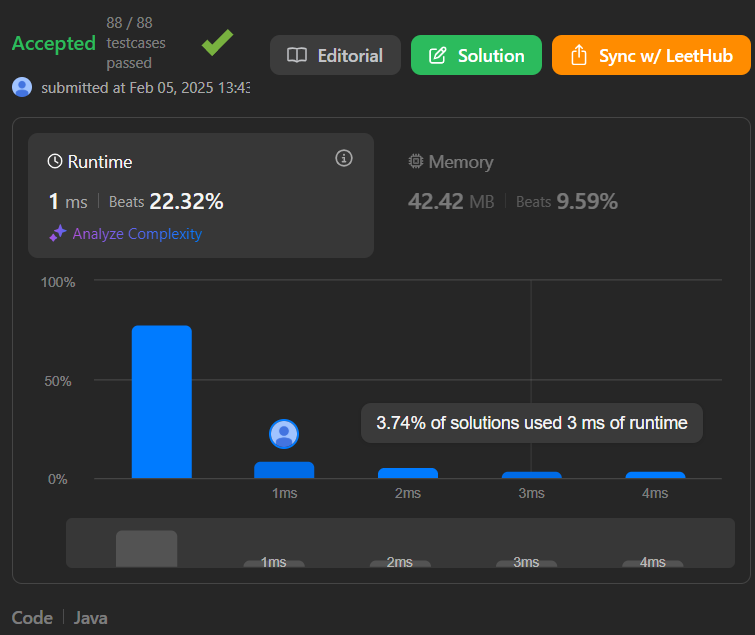
}

}

}

}

**Output:**



**13.Question:**



**Code:**

public int[] topKFrequent(int[] nums, int k) {

// Step 1: Count the frequency of each element

Map<Integer, Integer> h = new HashMap<>(); // num:count

for (int num:nums)

h.put(num,h.getOrDefault(num,0)+1);

// Step 2: Create buckets (arr) where index represents count

//Since it can have multiple nums in 1 bucket count, it will be a list of integer[]

List<Integer>[] arr = new ArrayList[nums.length+1]; //Can't have more size than the length of nums +1 (if all elements are same)

for (int num:h.keySet()) //count-> [0 1 2 3 ....]

{ //num-> [ 3 2 1 ....]

int count = h.get(num);

if (arr[count]==null) //index is the count

arr[count]=new ArrayList<>();

arr[count].add(num); //Add num to bucket at the index corresponding to its count

}

// Step 3: Gather the top k frequent elements

int[] res=new int[k];

int j=0;

for (int i=arr.length-1; i>=0 && j<k; i--) // Traverse from highest count to low

{

if (arr[i]!=null) {

for(int num:arr[i])

{

res[j]=num;

j++;

}

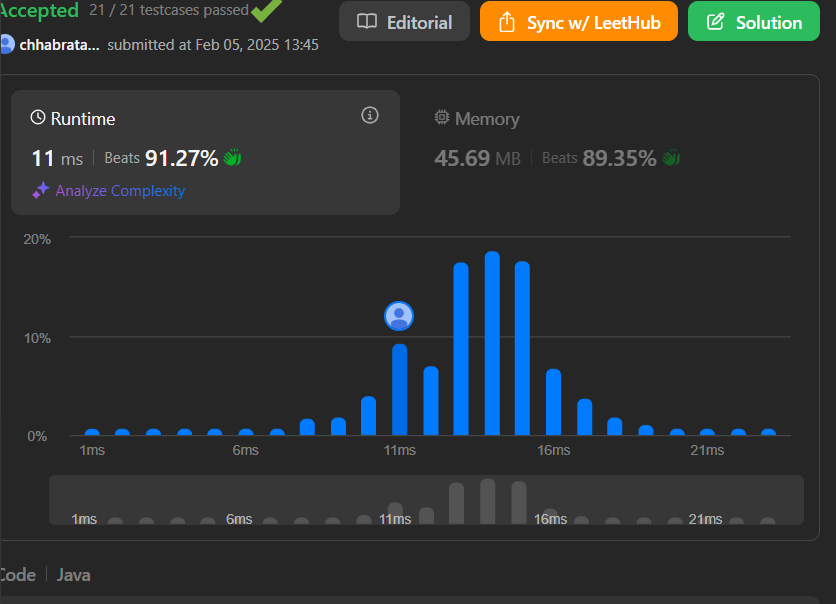
}

}

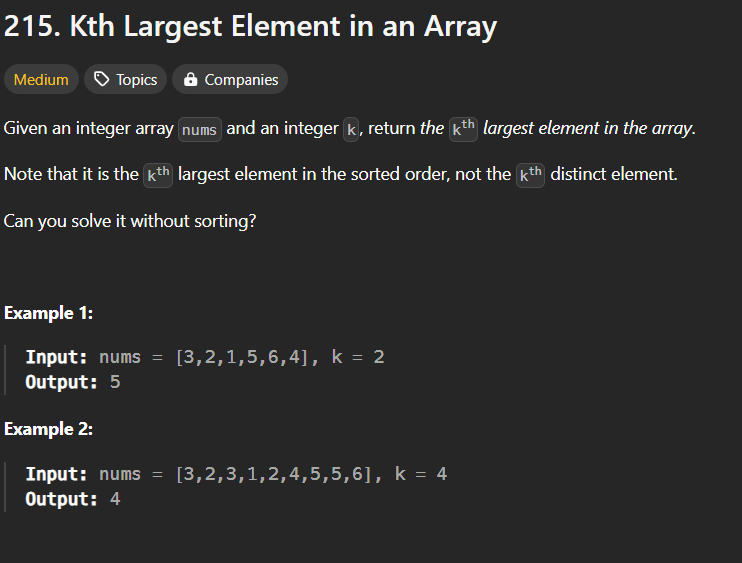
return res;

}

**Output:**



**14.Question:**



**Code:**

class Solution {

public int findKthLargest(int[] nums, int k) {

int[] count = new int[20001];

for (int num : nums)

count[num + 10000]++;

for (int i = count.length - 1; i >= 0; i--)

if (count[i] > 0) {

k -= count[i];

if (k <= 0) return i - 10000;

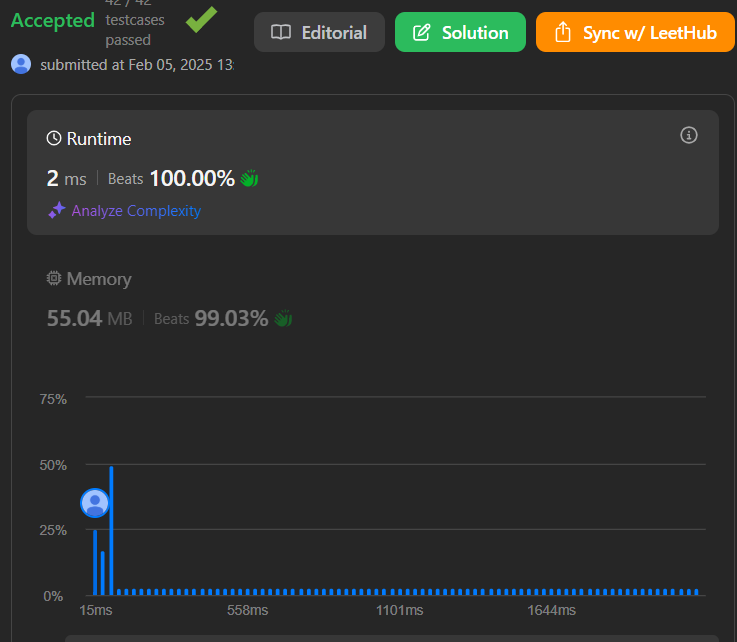
}

return -1;

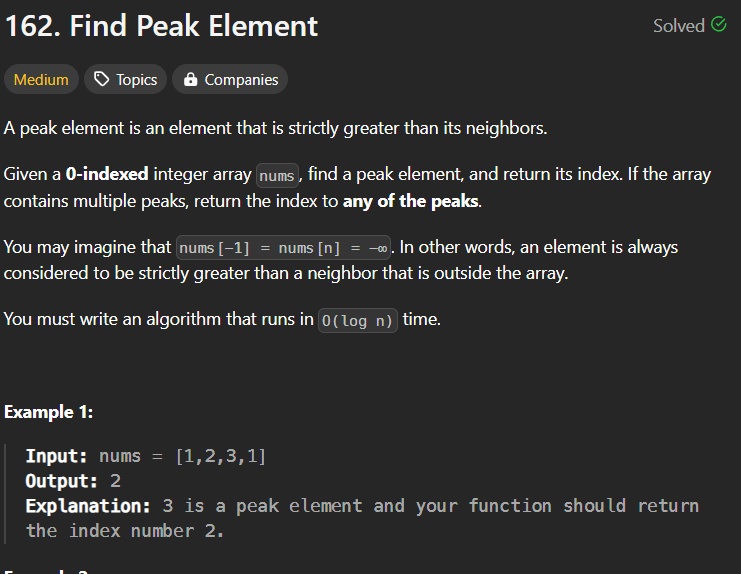
}

}

**Output:**



**15.Question:**



**Code:**

class Solution {

public int findPeakElement(int[] nums) {

int left = 0, right = nums.length-1;

while(left<right){

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

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

right = m;

}else{

left = m+1;

}

}

return left;

}

}

**Output:**

