# **1. Easy**

## 1.1. How Many Numbers Are Smaller Than the Current Number

**Link:** <https://leetcode.com/problems/how-many-numbers-are-smaller-than-the-current-number/>

Given the array nums, for each nums[i] find out how many numbers in the array are smaller than it. That is, for each nums[i] you have to count the number of valid j's such that j != i **and** nums[j] < nums[i].

Return the answer in an array.

**Example 1:**

**Input:** nums = [8,1,2,2,3]

**Output:** [4,0,1,1,3]

**Explanation:**

For nums[0]=8 there exist four smaller numbers than it (1, 2, 2 and 3).

For nums[1]=1 does not exist any smaller number than it.

For nums[2]=2 there exist one smaller number than it (1).

For nums[3]=2 there exist one smaller number than it (1).

For nums[4]=3 there exist three smaller numbers than it (1, 2 and 2).

**Example 2:**

**Input:** nums = [6,5,4,8]

**Output:** [2,1,0,3]

**Example 3:**

**Input:** nums = [7,7,7,7]

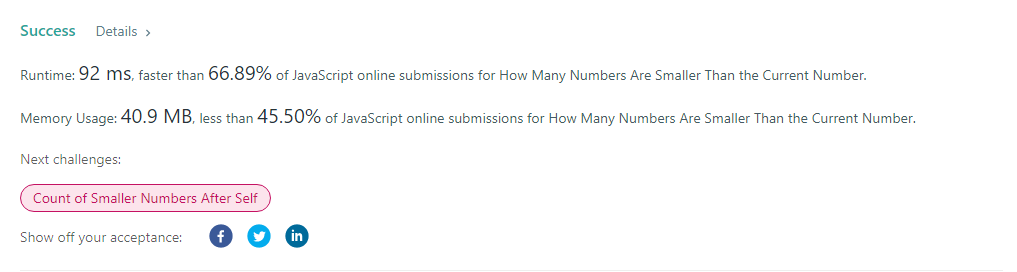
**Output:** [0,0,0,0]

**Constraints:**

2 <= nums.length <= 500

0 <= nums[i] <= 100

**Result:**



# **2. Medium**

## 2.1. Rotated Digits

**Link:** <https://leetcode.com/problems/rotated-digits/>

An integer x is a **good** if after rotating each digit individually by 180 degrees, we get a valid number that is different from x. Each digit must be rotated - we cannot choose to leave it alone.

A number is valid if each digit remains a digit after rotation. For example:

0, 1, and 8 rotate to themselves,

2 and 5 rotate to each other (in this case they are rotated in a different direction, in other words, 2 or 5 gets mirrored),

6 and 9 rotate to each other, and

the rest of the numbers do not rotate to any other number and become invalid.

Given an integer n, return *the number of****good****integers in the range*[1, n].

**Example 1:**

**Input:** n = 10

**Output:** 4

**Explanation:** There are four good numbers in the range [1, 10] : 2, 5, 6, 9.

Note that 1 and 10 are not good numbers, since they remain unchanged after rotating.

**Example 2:**

**Input:** n = 1

**Output:** 0

**Example 3:**

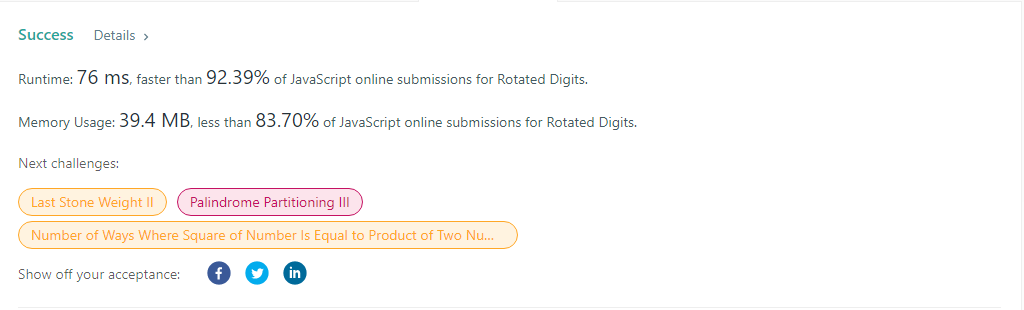
**Input:** n = 2

**Output:** 1

**Constraints:**

1 <= n <= 104

**Result:**

****

## 2.2. Average Waiting Time

**Link:** <https://leetcode.com/problems/average-waiting-time/>

There is a restaurant with a single chef. You are given an array customers, where customers[i] = [arrivali, timei]:

arrivali is the arrival time of the ith customer. The arrival times are sorted in **non-decreasing** order.

timei is the time needed to prepare the order of the ith customer.

When a customer arrives, he gives the chef his order, and the chef starts preparing it once he is idle. The customer waits till the chef finishes preparing his order. The chef does not prepare food for more than one customer at a time. The chef prepares food for customers **in the order they were given in the input**.

Return *the****average****waiting time of all customers*. Solutions within 10-5 from the actual answer are considered accepted.

**Example 1:**

**Input:** customers = [[1,2],[2,5],[4,3]]

**Output:** 5.00000

**Explanation:**

1) The first customer arrives at time 1, the chef takes his order and starts preparing it immediately at time 1, and finishes at time 3, so the waiting time of the first customer is 3 - 1 = 2.

2) The second customer arrives at time 2, the chef takes his order and starts preparing it at time 3, and finishes at time 8, so the waiting time of the second customer is 8 - 2 = 6.

3) The third customer arrives at time 4, the chef takes his order and starts preparing it at time 8, and finishes at time 11, so the waiting time of the third customer is 11 - 4 = 7.

So the average waiting time = (2 + 6 + 7) / 3 = 5.

**Example 2:**

**Input:** customers = [[5,2],[5,4],[10,3],[20,1]]

**Output:** 3.25000

**Explanation:**

1) The first customer arrives at time 5, the chef takes his order and starts preparing it immediately at time 5, and finishes at time 7, so the waiting time of the first customer is 7 - 5 = 2.

2) The second customer arrives at time 5, the chef takes his order and starts preparing it at time 7, and finishes at time 11, so the waiting time of the second customer is 11 - 5 = 6.

3) The third customer arrives at time 10, the chef takes his order and starts preparing it at time 11, and finishes at time 14, so the waiting time of the third customer is 14 - 10 = 4.

4) The fourth customer arrives at time 20, the chef takes his order and starts preparing it immediately at time 20, and finishes at time 21, so the waiting time of the fourth customer is 21 - 20 = 1.

So the average waiting time = (2 + 6 + 4 + 1) / 4 = 3.25.

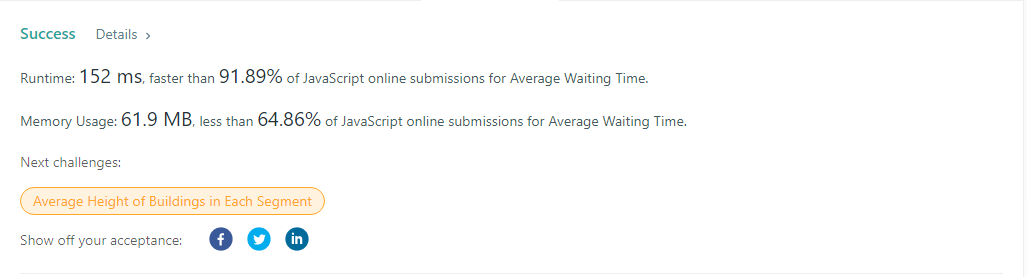
**Constraints:**

1 <= customers.length <= 105

1 <= arrivali, timei <= 104

arrivali<= arrivali+1

**Result:**

****

# 3.Hard

## 3.1. Median Of Two Sorted Arrays

**Link:** <https://leetcode.com/problems/median-of-two-sorted-arrays/>

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)).

**Example 1:**

**Input:** nums1 = [1,3], nums2 = [2]

**Output:** 2.00000

**Explanation:** merged array = [1,2,3] and median is 2.

**Example 2:**

**Input:** nums1 = [1,2], nums2 = [3,4]

**Output:** 2.50000

**Explanation:** merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

**Constraints:**

nums1.length == m

nums2.length == n

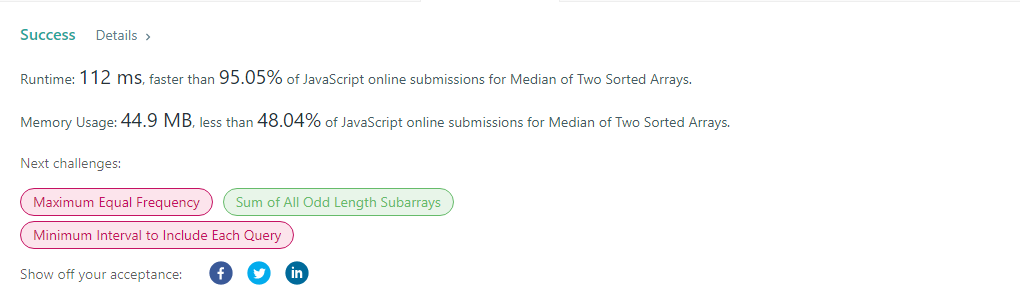
0 <= m <= 1000

0 <= n <= 1000

1 <= m + n <= 2000

-106 <= nums1[i], nums2[i] <= 106

**Result:**



## 3.2. Number of Visible People in a Queue

**Link:** <https://leetcode.com/problems/number-of-visible-people-in-a-queue/>

There are n people standing in a queue, and they numbered from 0 to n - 1 in **left to right** order. You are given an array heights of **distinct** integers where heights[i] represents the height of the ith person.

A person can **see** another person to their right in the queue if everybody in between is **shorter** than both of them. More formally, the ith person can see the jth person if i < j and min(heights[i], heights[j]) > max(heights[i+1], heights[i+2], ..., heights[j-1]).

Return *an array*answer*of length*n*where*answer[i]*is the****number of people****the*ith*person can****see****to their right in the queue*.

**Example 1:**



**Input:** heights = [10,6,8,5,11,9]

**Output:** [3,1,2,1,1,0]

**Explanation:**

Person 0 can see person 1, 2, and 4.

Person 1 can see person 2.

Person 2 can see person 3 and 4.

Person 3 can see person 4.

Person 4 can see person 5.

Person 5 can see no one since nobody is to the right of them.

**Example 2:**

**Input:** heights = [5,1,2,3,10]

**Output:** [4,1,1,1,0]

**Constraints:**

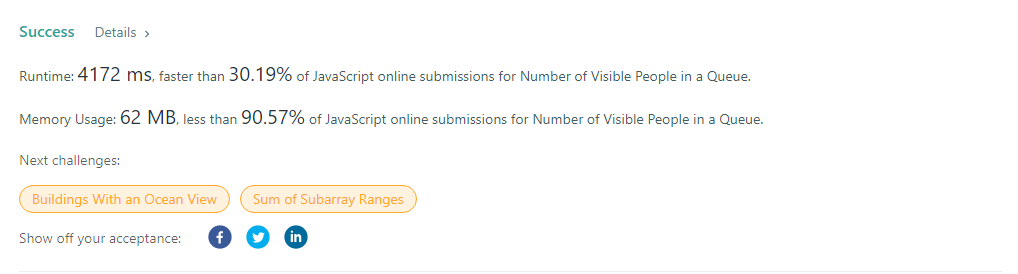
n == heights.length

1 <= n <= 105

1 <= heights[i] <= 105

All the values of heights are **unique**.

**Result:**



## 3.3. Minimum Number of K Consecutive Bit Flips

**Link:** <https://leetcode.com/problems/minimum-number-of-k-consecutive-bit-flips/>

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

A **k-bit flip** is choosing a **subarray** of length k from nums and simultaneously changing every 0 in the subarray to 1, and every 1 in the subarray to 0.

Return *the minimum number of****k-bit flips****required so that there is no*0*in the array*. If it is not possible, return -1.

A **subarray** is a **contiguous** part of an array.

**Example 1:**

**Input:** nums = [0,1,0], k = 1

**Output:** 2

**Explanation:** Flip nums[0], then flip nums[2].

**Example 2:**

**Input:** nums = [1,1,0], k = 2

**Output:** -1

**Explanation:** No matter how we flip subarrays of size 2, we cannot make the array become [1,1,1].

**Example 3:**

**Input:** nums = [0,0,0,1,0,1,1,0], k = 3

**Output:** 3

**Explanation:**

Flip nums[0],nums[1],nums[2]: nums becomes [1,1,1,1,0,1,1,0]

Flip nums[4],nums[5],nums[6]: nums becomes [1,1,1,1,1,0,0,0]

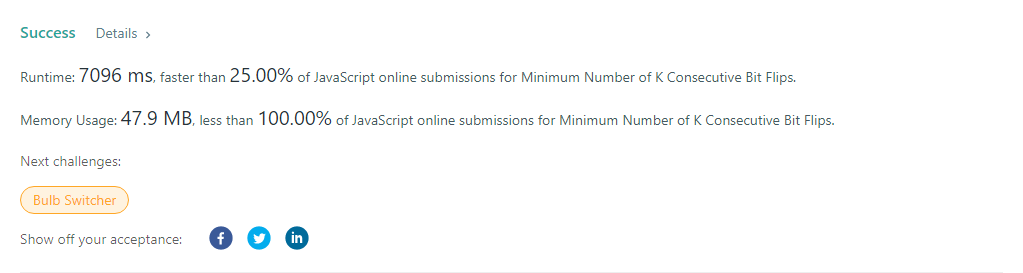
Flip nums[5],nums[6],nums[7]: nums becomes [1,1,1,1,1,1,1,1]

**Constraints:**

1 <= nums.length <= 105

1 <= k <= nums.length

**Result:**



## 3.4. Reducing Dishes

**Link:** <https://leetcode.com/problems/reducing-dishes/>

A chef has collected data on the satisfaction level of his n dishes. Chef can cook any dish in 1 unit of time.

**Like-time coefficient** of a dish is defined as the time taken to cook that dish including previous dishes multiplied by its satisfaction level i.e. time[i] \* satisfaction[i].

Return *the maximum sum of****like-time coefficient****that the chef can obtain after dishes preparation*.

Dishes can be prepared in **any**order and the chef can discard some dishes to get this maximum value.

**Example 1:**

**Input:** satisfaction = [-1,-8,0,5,-9]

**Output:** 14

**Explanation:** After Removing the second and last dish, the maximum total **like-time coefficient** will be equal to (-1\*1 + 0\*2 + 5\*3 = 14).

Each dish is prepared in one unit of time.

**Example 2:**

**Input:** satisfaction = [4,3,2]

**Output:** 20

**Explanation:** Dishes can be prepared in any order, (2\*1 + 3\*2 + 4\*3 = 20)

**Example 3:**

**Input:** satisfaction = [-1,-4,-5]

**Output:** 0

**Explanation:** People do not like the dishes. No dish is prepared.

**Constraints:**

* n == satisfaction.length
* 1 <= n <= 500
* -1000 <= satisfaction[i] <= 1000

**Result:**

