Arrays Lecture : 3

3.1 **118. Pascal's Triangle**

Solved

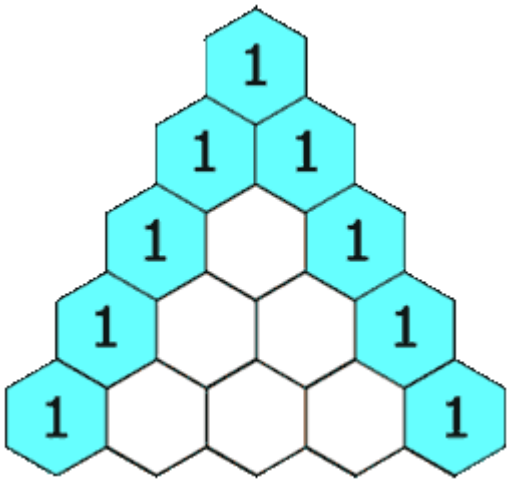
Easy

Topics

Companies

Given an integer numRows, return the first numRows of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



Example 1:

Input: numRows = 5

Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Example 2:

Input: numRows = 1

Output: [[1]]

Constraints:

- 1 <= numRows <= 30

3.2 **229. Majority Element II**

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times.

Example 1:

Input: nums = [3,2,3]

Output: [3]

Example 2:

Input: nums = [1]

Output: [1]

Example 3:

Input: nums = [1,2]

Output: [1,2]

Constraints:

- $1 <= nums.length <= 5 * 10^4$
- $-10^9 <= nums[i] <= 10^9$

3.3 **15. 3Sum**

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]

Output: [[-1,-1,2],[-1,0,1]]

Explanation:

nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.

nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.

nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.

The distinct triplets are [-1,0,1] and [-1,-1,2].

Notice that the order of the output and the order of the triplets does not matter.

Example 2:

Input: nums = [0,1,1]

Output: []

Explanation: The only possible triplet does not sum up to 0.

Example 3:

Input: nums = [0,0,0]

Output: [[0,0,0]]

Explanation: The only possible triplet sums up to 0.

Constraints:

- 3 <= nums.length <= 3000
- -105 <= nums[i] <= 105

3.4 **18. 4Sum**

Given an array nums of n integers, return *an array of all the* ***unique*** *quadruplets* [nums[a], nums[b], nums[c], nums[d]] such that:

- 0 <= a, b, c, d < n
- a, b, c, and d are **distinct**.
- nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in **any order**.

Example 1:

Input: nums = [1,0,-1,0,-2,2], target = 0

Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

Example 2:

Input: nums = [2,2,2,2,2], target = 8

Output: [[2,2,2,2]]

Constraints:

- 1 <= nums.length <= 200
- -109 <= nums[i] <= 109
- -109 <= target <= 109

3.5 **Largest subarray with 0 sum**

Difficulty: **Medium**Accuracy: **41.84%**Submissions: **408K+**Points: **4**Average Time: **20m**

Given an array **arr** containing both positive and negative integers, the task is to compute the length of the largest subarray that has a sum of 0.

Examples:

Input: arr[] = [15, -2, 2, -8, 1, 7, 10, 23]

Output: 5

Explanation: The largest subarray with a sum of 0 is [-2, 2, -8, 1, 7].

Input: arr[] = [2, 10, 4]

Output: 0

Explanation: There is no subarray with a sum of 0.

Input: arr[] = [1, 0, -4, 3, 1, 0]

Output: 5

Explanation: The subarray is [0, -4, 3, 1, 0].

Constraints:
$1 \leq arr.size() \leq 10^6$
$-10^3 \leq arr[i] \leq 10^3$, for each valid i

3.6 **Count Subarrays with given XOR**

Difficulty: **Medium**Accuracy: **58.86%**Submissions: **39K+**Points: **4**

Given an array of integers **arr[]** and a number **k**, count the number of subarrays having **XOR** of their elements as **k**.

Examples:

Input: arr[] = [4, 2, 2, 6, 4], k = 6
Output: 4
Explanation: The subarrays having XOR of their elements as 6 are [4, 2], [4, 2, 2, 6, 4], [2, 2, 6], and [6]. Hence, the answer is 4.

Input: arr[] = [5, 6, 7, 8, 9], k = 5
Output: 2
Explanation: The subarrays having XOR of their elements as 5 are [5] and [5, 6, 7, 8, 9]. Hence, the answer is 2.

Input: arr[] = [1, 1, 1, 1], k = 0
Output: 4
Explanation: The subarrays are [1, 1], [1, 1], [1, 1] and [1, 1, 1, 1].

Constraints:

- $1 \leq arr.size() \leq 10^5$
- $0 \leq arr[i] \leq 10^5$
- $0 \leq k \leq 10$

3.7 **56. Merge Intervals**

Medium

Topics

Companies

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

Example 1:

Input: intervals = [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Example 2:

Input: intervals = [[1,4],[4,5]]

Output: [[1,5]]

Explanation: Intervals [1,4] and [4,5] are considered overlapping.

Constraints:

- $1 <= intervals.length <= 10^4$
- intervals[i].length == 2
- $0 <= start_i <= end_i <= 10^4$

3.8 **88. Merge Sorted Array**

Solved

Easy

Topics

Companies

Hint

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.

Merge nums1 and nums2 into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

Explanation: The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Example 2:

Input: nums1 = [1], m = 1, nums2 = [], n = 0

Output: [1]

Explanation: The arrays we are merging are [1] and [].

The result of the merge is [1].

Example 3:

Input: nums1 = [0], m = 0, nums2 = [1], n = 1

Output: [1]

Explanation: The arrays we are merging are [] and [1].

The result of the merge is [1].

Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

Constraints:

- nums1.length == m + n
- nums2.length == n
- 0 <= m, n <= 200
- 1 <= m + n <= 200
- -10^9 <= nums1[i], nums2[j] <= 10^9

3.9 **2965. Find Missing and Repeated Values**

Easy

Topics

Companies

You are given a **0-indexed** 2D integer matrix grid of size n * n with values in the range [1, n^2]. Each integer appears **exactly once** except a which appears **twice** and b which is **missing**. The task is to find the repeating and missing numbers a and b.

Return *a **0-indexed** integer array* ans *of size* 2 *where* ans[0] *equals to* a *and* ans[1] *equals to* b.

Example 1:

Input: grid = [[1,3],[2,2]]

Output: [2,4]

Explanation: Number 2 is repeated and number 4 is missing so the answer is [2,4].

Example 2:

Input: grid = [[9,1,7],[8,9,2],[3,4,6]]

Output: [9,5]

Explanation: Number 9 is repeated and number 5 is missing so the answer is [9,5].

Constraints:

- 2 <= n == grid.length == grid[i].length <= 50
- 1 <= grid[i][j] <= n * n

- For all x that 1 <= x <= n * n there is exactly one x that is not equal to any of the grid members.
- For all x that 1 <= x <= n * n there is exactly one x that is equal to exactly two of the grid members.
- For all x that 1 <= x <= n * n except two of them there is exactly one pair of i, j that 0 <= i, j <= n - 1 and grid[i][j] ==

3.10 **Count Inversions**

Difficulty: **Medium**Accuracy: **16.93%**Submissions: **664K+**Points: **4**

Given an array of integers **arr[]**. Find the **Inversion Count** in the array.
Two elements arr[i] and arr[j] form an inversion if **arr[i] > arr[j]** and **i < j**.

***Inversion Count**:* For an array, inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted then the inversion count is 0.
If an array is sorted in the reverse order then the inversion count is the maximum.

Examples:

Input: arr[] = [2, 4, 1, 3, 5]
Output: 3

Explanation: The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

Input: arr[] = [2, 3, 4, 5, 6]
Output: 0

Explanation: As the sequence is already sorted so there is no inversion count.

Input: arr[] = [10, 10, 10]
Output: 0

Explanation: As all the elements of array are same, so there is no inversion count.

Constraints:
$1 \leq arr.size() \leq 10^5$
$1 \leq arr[i] \leq 10^4$

3.11 **493. Reverse Pairs**

Hard

Topics

Companies

Hint

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

A **reverse pair** is a pair (i, j) where:

- 0 <= i < j < nums.length and
- nums[i] > 2 * nums[j].

Example 1:

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

Output: 2

Explanation: The reverse pairs are:

(1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 * 1

(3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 * 1

Example 2:

Input: nums = [2,4,3,5,1]

Output: 3

Explanation: The reverse pairs are:

(1, 4) --> nums[1] = 4, nums[4] = 1, 4 > 2 * 1

(2, 4) --> nums[2] = 3, nums[4] = 1, 3 > 2 * 1

(3, 4) --> nums[3] = 5, nums[4] = 1, 5 > 2 * 1

Constraints:

- 1 <= nums.length <= $5 * 10^4$
- -2^{31} <= nums[i] <= $2^{31} - 1$

3.12 **152. Maximum Product Subarray**

Medium

Topics

Companies

Given an integer array nums, find a subarray that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

Example 1:

Input: nums = [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.

Example 2:

Input: nums = [-2,0,-1]

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

Constraints:

- 1 <= nums.length <= $2 * 10^4$
- -10 <= nums[i] <= 10
- The product of any subarray of nums is **guaranteed** to fit in a **32-bit** integer.