# Count Sequence

In some 1D array, sometimes we want to count the sequence with some specific condition. For such problem, we normally think as if we are doing a dynamic programming. We scan the array from left to right and stand at every position and think how many such sequences which shall satisfy the condition with the end element as current one.

## 446. Arithmetic Slices II - Subsequence

Hard

A sequence of numbers is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequences:

1, 3, 5, 7, 9

7, 7, 7, 7

3, -1, -5, -9

The following sequence is not arithmetic.

1, 1, 2, 5, 7

A zero-indexed array A consisting of N numbers is given. A **subsequence** slice of that array is any sequence of integers (P0, P1, ..., Pk) such that 0 ≤ P0 < P1 < ... < Pk < N.

A **subsequence** slice (P0, P1, ..., Pk) of array A is called arithmetic if the sequence A[P0], A[P1], ..., A[Pk-1], A[Pk] is arithmetic. In particular, this means that k ≥ 2.

The function should return the number of arithmetic subsequence slices in the array A.

The input contains N integers. Every integer is in the range of -231 and 231-1 and 0 ≤ N ≤ 1000. The output is guaranteed to be less than 231-1.

**Example:**

**Input:** [2, 4, 6, 8, 10]

**Output:** 7

**Explanation:**

All arithmetic subsequence slices are:

[2,4,6]

[4,6,8]

[6,8,10]

[2,4,6,8]

[4,6,8,10]

[2,4,6,8,10]

[2,6,10]

### Analysis:

For every number, you pick every previous number to calculate the difference, and add the number of arithmetic numbers to it. The arithmetic numbers already count in the previous should be the number of arithmetic slices added to this number.

/// <summary>

/// Leet code #446. Arithmetic Slices II - Subsequence

///

/// A sequence of numbers is called arithmetic if it consists of at

/// least three elements and if the difference between any two

/// consecutive elements is the same.

///

/// For example, these are arithmetic sequences:

/// 1, 3, 5, 7, 9

/// 7, 7, 7, 7

/// 3, -1, -5, -9

///

/// The following sequence is not arithmetic.

/// 1, 1, 2, 5, 7

/// A zero-indexed array A consisting of N numbers is given. A

/// subsequence slice of that array is any sequence of integers

/// (P0, P1, ..., Pk) such that 0 ≤ P0 < P1 < ... < Pk < N.

///

/// A subsequence slice (P0, P1, ..., Pk) of array A is called

/// arithmetic if the sequence A[P0], A[P1], ..., A[Pk-1], A[Pk]

/// is arithmetic. In particular, this means that k ≥ 2.

///

/// The function should return the number of arithmetic subsequence

/// slices in the array A.

///

/// The input contains N integers. Every integer is in the range of -2^31

/// and 2^31-1 and 0 ≤ N ≤ 1000. The output is guaranteed to be less than

/// 2^31-1.

///

/// Example:

/// Input: [2, 4, 6, 8, 10]

/// Output: 7

///

/// Explanation:

/// All arithmetic subsequence slices are:

/// [2,4,6]

/// [4,6,8]

/// [6,8,10]

/// [2,4,6,8]

/// [4,6,8,10]

/// [2,4,6,8,10]

/// [2,6,10]

/// </summary>

int LeetCodeArray::numberOfArithmeticSlicesII(vector<int>& A)

{

int result = 0;

vector<unordered\_map<long long, int>> dp(A.size());

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

{

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

{

long long diff = (long long)A[i] - (long long)A[j];

if (dp[j].count(diff) > 0)

{

dp[i][diff] += dp[j][diff];

result += dp[j][diff];

}

dp[i][diff]++;

}

}

return result;

}

## 368. Largest Divisible Subset

Medium

Given a set of **distinct** positive integers, find the largest subset such that every pair (Si, Sj) of elements in this subset satisfies:

Si % Sj = 0 or Sj % Si = 0.

If there are multiple solutions, return any subset is fine.

**Example 1:**

**Input:** [1,2,3]

**Output:** [1,2] (of course, [1,3] will also be ok)

**Example 2:**

**Input:** [1,2,4,8]

**Output:** [1,2,4,8]

### Analysis:

Sort the array and build longest divisible chain.

/// <summary>

/// Leet code #368. Largest Divisible Subset

///

/// Given a set of distinct positive integers, find the largest

/// subset such that every pair (Si, Sj) of elements in this subset

/// satisfies: Si % Sj = 0 or Sj % Si = 0.

///

/// If there are multiple solutions, return any subset is fine.

///

/// Example 1:

/// nums: [1,2,3]

/// Result: [1,2] (of course, [1,3] will also be ok)

///

/// Example 2:

/// nums: [1,2,4,8]

/// Result: [1,2,4,8]

/// </summary>

vector<int> LeetCodeArray::largestDivisibleSubset(vector<int>& nums)

{

vector<int> result;

vector<vector<int>> set\_map(nums.size());

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

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

{

for (size\_t j = 0; j < i; j++)

{

if ((nums[i] % nums[j]) == 0)

{

if (set\_map[i].size() < set\_map[j].size())

{

set\_map[i] = set\_map[j];

}

}

}

set\_map[i].push\_back(nums[i]);

if (set\_map[i].size() > result.size()) result = set\_map[i];

}

return result;

}

## 891. Sum of Subsequence Widths

Hard

Given an array of integers A, consider all non-empty subsequences of A.

For any sequence S, let the *width* of S be the difference between the maximum and minimum element of S.

Return the sum of the widths of all subsequences of A.

As the answer may be very large, **return the answer modulo 10^9 + 7**.

**Example 1:**

**Input:** [2,1,3]

**Output:** 6

**Explanation:**

Subsequences are [1], [2], [3], [2,1], [2,3], [1,3], [2,1,3].

The corresponding widths are 0, 0, 0, 1, 1, 2, 2.

The sum of these widths is 6.

**Note:**

* 1 <= A.length <= 20000
* 1 <= A[i] <= 20000

### Analysis:

For every pair of maximum and minimum value, regardless their location, the sub sequences will involve all the values which is no less than them. So if we sort all the values, and calculate the every pair of the value and calculate the number of array, size from 2 to the distance between them, as the factor. The total sum is the answer.

/// <summary>

/// Leet code #891. Sum of Subsequence Widths

///

/// Given an array of integers A, consider all non-empty subsequences of A.

///

/// For any sequence S, let the width of S be the difference between the

/// maximum and minimum element of S.

///

/// Return the sum of the widths of all subsequences of A.

///

/// As the answer may be very large, return the answer modulo 10^9 + 7.

///

/// Example 1:

///

/// Input: [2,1,3]

/// Output: 6

/// Explanation:

/// Subsequences are [1], [2], [3], [2,1], [2,3], [1,3], [2,1,3].

/// The corresponding widths are 0, 0, 0, 1, 1, 2, 2.

/// The sum of these widths is 6.

///

/// Note:

///

/// 1 <= A.length <= 20000

/// 1 <= A[i] <= 20000

/// </summary>

int LeetCodeArray::sumSubseqWidths(vector<int>& A)

{

int M = 1000000007;

vector<int> nums = A;

int result = 0;

vector<int> factor(A.size());

for (size\_t i = 0; i < A.size(); i++)

{

if (i == 0) factor[i] = 1;

else factor[i] = (factor[i - 1] << 1) % M;

}

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

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

{

result = (result + ((long long)factor[i] - (long long)1) \* nums[i]) % M;

result = (result - ((long long)factor[A.size() - (long long)1 - (long long)i] - 1) \* nums[i]) % M;

result = (result + M) % M;

}

return result;

}

## 1063. Number of Valid Subarrays

Hard

Given an array A of integers, return the number of **non-empty continuous subarrays** that satisfy the following condition:

The leftmost element of the subarray is not larger than other elements in the subarray.

**Example 1:**

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

**Output:** 11

**Explanation:** There are 11 valid subarrays: [1],[4],[2],[5],[3],[1,4],[2,5],[1,4,2],[2,5,3],[1,4,2,5],[1,4,2,5,3].

**Example 2:**

**Input:** [3,2,1]

**Output:** 3

**Explanation:** The 3 valid subarrays are: [3],[2],[1].

**Example 3:**

**Input:** [2,2,2]

**Output:** 6

**Explanation:** There are 6 valid subarrays: [2],[2],[2],[2,2],[2,2],[2,2,2].

**Note:**

1. 1 <= A.length <= 50000
2. 0 <= A[i] <= 100000

### Analysis:

For every subarray starting from index [i], the element must be equal or less than all the elements in the right, it can expand as long as possible until you see a smaller number. We can use a monotone stack keep track on all valid heads. Every time we count the stack size as the valid subarray count ending current index.

/// <summary>

/// Leet code #1063. Number of Valid Subarrays

///

/// Given an array A of integers, return the number of non-empty continuous

/// subarrays that satisfy the following condition:

///

/// The leftmost element of the subarray is not larger than other elements

/// in the subarray.

///

/// Example 1:

///

/// Input: [1,4,2,5,3]

/// Output: 11

/// Explanation: There are 11 valid subarrays: [1],[4],[2],[5],[3],[1,4],

/// [2,5],[1,4,2],[2,5,3],[1,4,2,5],[1,4,2,5,3].

///

/// Example 2:

///

/// Input: [3,2,1]

/// Output: 3

/// Explanation: The 3 valid subarrays are: [3],[2],[1].

///

/// Example 3:

///

/// Input: [2,2,2]

/// Output: 6

/// Explanation: There are 6 valid subarrays: [2],[2],[2],[2,2],[2,2],[2,2,2].

///

///

/// Note:

///

/// 1. 1 <= A.length <= 50000

/// 2. 0 <= A[i] <= 100000

/// </summary>

int LeetCodeArray::validSubarrays(vector<int>& nums)

{

int result = 0;

stack<int> dp;

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

{

if (dp.empty())

{

dp.push(nums[i]);

}

else

{

while (!dp.empty())

{

if (dp.top() > nums[i])

{

dp.pop();

}

else

{

break;

}

}

dp.push(nums[i]);

}

result += dp.size();

}

return result;

}

## [419. Battleships in a Board](https://leetcode.com/problems/battleships-in-a-board/)

Medium

Given an m x n matrix board where each cell is a battleship 'X' or empty '.', return the number of the ***battleships*** on board.

**Battleships** can only be placed horizontally or vertically on board. In other words, they can only be made of the shape 1 x k (1 row, k columns) or k x 1 (k rows, 1 column), where k can be of any size. At least one horizontal or vertical cell separates between two battleships (i.e., there are no adjacent battleships).

**Example 1:**

A grid of white squares with black x

Description automatically generated

**Input:** board = [["X",".",".","X"],[".",".",".","X"],[".",".",".","X"]]

**Output:** 2

**Example 2:**

**Input:** board = [["."]]

**Output:** 0

**Constraints:**

* m == board.length
* n == board[i].length
* 1 <= m, n <= 200
* board[i][j] is either '.' or 'X'.

**Follow up:** Could you do it in one-pass, using only O(1) extra memory and without modifying the values board?

### Analysis:

Count up and left.

/// <summary>

/// Leet Code 419. Battleships in a Board

///

/// Medium

///

/// Given an m x n matrix board where each cell is a battleship 'X' or empty

/// '.', return the number of the battleships on board.

///

/// Battleships can only be placed horizontally or vertically on board. In

/// other words, they can only be made of the shape 1 x k (1 row, k columns)

/// or k x 1 (k rows, 1 column), where k can be of any size. At least one

/// horizontal or vertical cell separates between two battleships (i.e.,

/// there are no adjacent battleships).

///

/// Example 1:

/// Input: board = [["X",".",".","X"],[".",".",".","X"],[".",".",".","X"]]

/// Output: 2

///

/// Example 2:

/// Input: board = [["."]]

/// Output: 0

///

/// Constraints:

/// 1. m == board.length

/// 2. n == board[i].length

/// 3. 1 <= m, n <= 200

/// 4. board[i][j] is either '.' or 'X'.

/// Follow up: Could you do it in one-pass, using only O(1) extra memory and

/// without modifying the values board?

/// </summary>

int LeetCodeArray::countBattleships(vector<vector<char>>& board)

{

int count = 0;

for (size\_t i = 0; i < board.size(); i++)

{

for (size\_t j = 0; j < board[0].size(); j++)

{

if ((board[i][j] == 'X') && ((i == 0) || (board[i - 1][j] == '.')) &&

((j == 0) || (board[i][j - 1] == '.')))

{

count++;

}

}

}

return count;

}

## [1248. Count Number of Nice Subarrays](https://leetcode.com/problems/count-number-of-nice-subarrays/)

Medium

Given an array of integers nums and an integer k. A continuous subarray is called **nice** if there are k odd numbers on it.

Return the number of ***nice*** sub-arrays.

**Example 1:**

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

**Output:** 2

**Explanation:** The only sub-arrays with 3 odd numbers are [1,1,2,1] and [1,2,1,1].

**Example 2:**

**Input:** nums = [2,4,6], k = 1

**Output:** 0

**Explanation:** There is no odd numbers in the array.

**Example 3:**

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

**Output:** 16

**Constraints:**

* 1 <= nums.length <= 50000
* 1 <= nums[i] <= 10^5
* 1 <= k <= nums.length

### Analysis:

Count up even number before every odd.

/// <summary>

/// Leet code #1248. Count Number of Nice Subarrays

///

/// Given an array of integers nums and an integer k. A subarray is called

/// nice if there are k odd numbers on it.

///

/// Return the number of nice sub-arrays.

///

/// Example 1:

///

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

/// Output: 2

/// Explanation: The only sub-arrays with 3 odd numbers are [1,1,2,1]

/// and [1,2,1,1].

///

/// Example 2:

///

/// Input: nums = [2,4,6], k = 1

/// Output: 0

/// Explanation: There is no odd numbers in the array.

///

/// Example 3:

///

/// Input: nums = [2,2,2,1,2,2,1,2,2,2], k = 2

/// Output: 16

///

/// Constraints:

/// 1. 1 <= nums.length <= 50000

/// 2. 1 <= nums[i] <= 10^5

/// 3. 1 <= k <= nums.length

/// </summary>

int LeetCodeArray::numberOfSubarrays(vector<int>& nums, int k)

{

deque<int> window;

int count = 1;

int result = 0;

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

{

if (nums[i] % 2 == 1)

{

window.push\_back(count);

count = 1;

}

else

{

count++;

}

if (window.size() == k)

{

result += window.front();

}

else if (window.size() == k + 1)

{

window.pop\_front();

result += window.front();

}

}

return result;

}

## [2962. Count Subarrays Where Max Element Appears at Least K Times](https://leetcode.com/problems/count-subarrays-where-max-element-appears-at-least-k-times/)

Solved

Medium

Topics

Companies

You are given an integer array nums and a **positive** integer k.

Return the number of subarrays where the ***maximum*** element of nums appears ***at least*** k times in that subarray.

A **subarray** is a contiguous sequence of elements within an array.

**Example 1:**

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

**Output:** 6

**Explanation:** The subarrays that contain the element 3 at least 2 times are: [1,3,2,3], [1,3,2,3,3], [3,2,3], [3,2,3,3], [2,3,3] and [3,3].

**Example 2:**

**Input:** nums = [1,4,2,1], k = 3

**Output:** 0

**Explanation:** No subarray contains the element 4 at least 3 times.

**Constraints:**

* 1 <= nums.length <= 105
* 1 <= nums[i] <= 106
* 1 <= k <= 105

### Analysis:

The maximum number in the array is determined, count such numbers when reach count of K of such number, count how many elements before the first element (inclusive) on this k numbers.

/// <summary>

/// Leet Code 2962. Count Subarrays Where Max Element Appears at Least

/// K Times

///

/// Medium

///

/// You are given an integer array nums and a positive integer k.

///

/// Return the number of subarrays where the maximum element of nums appears

/// at least k times in that subarray.

///

/// A subarray is a contiguous sequence of elements within an array.

///

/// Example 1:

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

/// Output: 6

/// Explanation: The subarrays that contain the element 3 at least 2 times

/// are: [1,3,2,3], [1,3,2,3,3], [3,2,3], [3,2,3,3], [2,3,3] and [3,3].

///

/// Example 2:

/// Input: nums = [1,4,2,1], k = 3

/// Output: 0

/// Explanation: No subarray contains the element 4 at least 3 times.

///

/// Constraints:

/// 1. 1 <= nums.length <= 10^5

/// 2. 1 <= nums[i] <= 10^6

/// 3. 1 <= k <= 10^5

/// </summary>

long long LeetCodeArray::countSubarraysMaxValue(vector<int>& nums, int k)

{

int max\_val = 0;

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

{

max\_val = max(max\_val, nums[i]);

}

long long result = 0;

vector<int> dp;

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

{

if (nums[i] == max\_val) dp.push\_back(i);

if ((int)dp.size() >= k)

{

result += (long long)dp[dp.size() - k] + 1;

}

}

return result;

}

## [1442. Count Triplets That Can Form Two Arrays of Equal XOR](https://leetcode.com/problems/count-triplets-that-can-form-two-arrays-of-equal-xor/)

Medium

Given an array of integers arr.

We want to select three indices i, j and k where (0 <= i < j <= k < arr.length).

Let's define a and b as follows:

* a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]
* b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]

Note that **^** denotes the **bitwise-xor** operation.

Return the number of triplets (i, j and k) Where a == b.

**Example 1:**

**Input:** arr = [2,3,1,6,7]

**Output:** 4

**Explanation:** The triplets are (0,1,2), (0,2,2), (2,3,4) and (2,4,4)

**Example 2:**

**Input:** arr = [1,1,1,1,1]

**Output:** 10

**Constraints:**

* 1 <= arr.length <= 10^5
* 1 <= arr[i] <= 10^8

### Analysis:

When prefix XOR see duplicate at p before, we can split any p + 1 to current index k - 1, which means we will calculate from all previous position + 1 with duplicate XOR value to current index - 1, which means we will add index k \* duplicate count – sum (all previous position + 1).

/// <summary>

/// Leet code #1442. Count Triplets That Can Form Two Arrays of Equal XOR

///

/// Medium

///

/// Given an array of integers arr.

///

/// We want to select three indices i, j and k where

/// (0 <= i < j <= k < arr.length).

///

/// Let's define a and b as follows:

///

/// a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]

/// b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]

/// Note that ^ denotes the bitwise-xor operation.

///

/// Return the number of triplets (i, j and k) Where a == b.

///

/// Example 1:

/// Input: arr = [2,3,1,6,7]

/// Output: 4

/// Explanation: The triplets are (0,1,2), (0,2,2), (2,3,4) and (2,4,4)

///

/// Example 2:

/// Input: arr = [1,1,1,1,1]

/// Output: 10

///

/// Example 3:

/// Input: arr = [2,3]

/// Output: 0

///

/// Example 4:

/// Input: arr = [1,3,5,7,9]

/// Output: 3

///

/// Example 5:

/// Input: arr = [7,11,12,9,5,2,7,17,22]

/// Output: 8

///

/// Constraints:

/// 1. 1 <= arr.length <= 300

/// 2. 1 <= arr[i] <= 10^8

/// </summary>

int LeetCodeArray::countTriplets(vector<int>& arr)

{

int result = 0;

map<int, pair<int, int>> xor\_map;

int sum\_xor = 0;

xor\_map[0] = { 0, 1 };

for (size\_t i = 0; i < arr.size(); i++)

{

sum\_xor = sum\_xor ^ arr[i];

if (xor\_map.count(sum\_xor) > 0)

{

result += (i \* xor\_map[sum\_xor].second - xor\_map[sum\_xor].first);

}

xor\_map[sum\_xor].first += i + 1;

xor\_map[sum\_xor].second ++;

}

return result;

}

## [2963. Count the Number of Good Partitions](https://leetcode.com/problems/count-the-number-of-good-partitions/)

Hard

You are given a **0-indexed** array nums consisting of **positive** integers.

A partition of an array into one or more **contiguous** subarrays is called **good** if no two subarrays contain the same number.

Return the ***total number*** of good partitions of nums.

Since the answer may be large, return it **modulo** 109 + 7.

**Example 1:**

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

**Output:** 8

**Explanation:** The 8 possible good partitions are: ([1], [2], [3], [4]), ([1], [2], [3,4]), ([1], [2,3], [4]), ([1], [2,3,4]), ([1,2], [3], [4]), ([1,2], [3,4]), ([1,2,3], [4]), and ([1,2,3,4]).

**Example 2:**

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

**Output:** 1

**Explanation:** The only possible good partition is: ([1,1,1,1]).

**Example 3:**

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

**Output:** 2

**Explanation:** The 2 possible good partitions are: ([1,2,1], [3]) and ([1,2,1,3]).

**Constraints:**

* 1 <= nums.length <= 105
* 1 <= nums[i] <= 109

### Analysis:

Use greedy to expand the partition until every number in one partition, then use side and bar theory to calculate number of partition combinations.

/// <summary>

/// Leet Code 2963. Count the Number of Good Partitions

///

/// Hard

///

/// You are given a 0-indexed array nums consisting of positive integers.

///

/// A partition of an array into one or more contiguous subarrays is called

/// good if no two subarrays contain the same number.

///

/// Return the total number of good partitions of nums.

///

/// Since the answer may be large, return it modulo 10^9 + 7.

///

/// Example 1:

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

/// Output: 8

/// Explanation: The 8 possible good partitions are: ([1], [2], [3], [4]),

/// ([1], [2], [3,4]), ([1], [2,3], [4]), ([1], [2,3,4]), ([1,2], [3], [4]),

/// ([1,2], [3,4]), ([1,2,3], [4]), and ([1,2,3,4]).

///

/// Example 2:

/// Input: nums = [1,1,1,1]

/// Output: 1

/// Explanation: The only possible good partition is: ([1,1,1,1]).

///

/// Example 3:

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

/// Output: 2

/// Explanation: The 2 possible good partitions are: ([1,2,1], [3])

/// and ([1,2,1,3]).

///

/// Constraints:

/// 1. 1 <= nums.length <= 10^5

/// 2. 1 <= nums[i] <= 10^9

/// </summary>

int LeetCodeArray::numberOfGoodPartitions(vector<int>& nums)

{

unordered\_map<int, int> num\_count;

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

{

num\_count[nums[i]]++;

}

unordered\_set<int> num\_exist;

int partition = 0;

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

{

int n = nums[i];

num\_exist.insert(n);

num\_count[n]--;

if (num\_count[n] == 0)

{

num\_exist.erase(n);

}

if (num\_exist.empty()) partition++;

}

int result = 1;

int M = 1000000007;

for (int i = 0; i < partition - 1; i++)

{

result = result \* 2 % M;

}

return result;

}