AlphaGrep *2022 Oct04*

All 4 questions are about array. Most of them are not about Dynamic programming.

1. Global Maximum

Consider an array of distinct positive integers where the elements are sorted in ascending order. We want to find all the subsequences of the array consisting of exactly *m* elements. For example, given array *a = [1, 2, 3, 4]*, the subsequences consisting of *m = 3* elements are *[1, 2, 3]*, *[1, 2, 4]*, *[1, 3, 4]*, and *[2, 3, 4]*. Once we have all of the *m*-element subsequences, we find the value of *globalMaximum* using the following pseudocode:

```
globalMaximum = 0

for each subsequence, s, consisting of m elements {
   currentMinimum = 10^18

   for each (x, y) pair of elements in subsequence s {
      absoluteDifference = abs(x - y)

      if absoluteDifference < currentMinimum {
         currentMinimum = absoluteDifference
      }
   }

   if currentMinimum > globalMaximum {
      globalMaximum = currentMinimum
   }
}
```

For example, given the array *a = [2,3,5,9]* and a length *m = 3*, first find all subsequences of length *m*.

```
[2,3,5]
[2,3,9]
[2,5,9]
[3,5,9]
```

Debugging output from the pseudocode is:

```
globalMaximum: 0

Subsequence: [2, 3, 5]
currentMinimum: 1000000000000000000
x: 3 y: 2 abs(x-y): 1 currentMinimum: 1
x: 5 y: 2 abs(x-y): 3 currentMinimum: 1
x: 5 y: 3 abs(x-y): 2 currentMinimum: 1
globalMaximum: max(globalMaximum, currentMinimum) = max(1, 0) = 1

Subsequence: [2, 3, 9]
currentMinimum: 1000000000000000000
x: 3 y: 2 abs(x-y): 1 currentMinimum: 1
x: 9 y: 2 abs(x-y): 7 currentMinimum: 1
x: 9 y: 3 abs(x-y): 6 currentMinimum: 1
globalMaximum: max(1, 1) = 1

Subsequence: [2, 5, 9]
currentMinimum: 1000000000000000000
x: 5 y: 2 abs(x-y): 3 currentMinimum: 3
x: 9 y: 2 abs(x-y): 7 currentMinimum: 3
x: 9 y: 5 abs(x-y): 4 currentMinimum: 3
globalMaximum: max(1, 3) = 3

Subsequence: [3, 5, 9]
currentMinimum: 1000000000000000000
x: 5 y: 3 abs(x-y): 2 currentMinimum: 2
x: 9 y: 3 abs(x-y): 6 currentMinimum: 2
x: 9 y: 5 abs(x-y): 4 currentMinimum: 2
globalMaximum: max(3, 2) = 3
```

In the example above, the final answer is globalMaximum = 3.

Function Description

Complete the function *findMaximum* in the editor below.

findMaximum has the following parameter(s):

int arr[n]: a sorted array of distinct integers
int m: an integer, the subsequences' length

Return

int: the *globalMaximum* calculated per the pseudocode

Constraints

- $2 \le n \le 10^5$
- $1 \le a[i] \le 10^9$.
- The array consists of distinct positive integers sorted in ascending order.
- $2 \le m \le n$

2. Even Subarray

A subarray is a contiguous portion of an array. Given an array of integers, determine the number of distinct subarrays that can be formed having at most a given number of odd elements. Two subarrays are distinct if they differ at even one position in their contents.

Example
numbers = [1, 2, 3, 4]
k = 1

The following is a list of the 8 distinct valid subarrays having no more than 1 odd element:

```
[[1], [2], [3], [4], [1,2], [2, 3], [3, 4], [2, 3, 4]]
```

Function Description
Complete the function *evenSubarray* in the editor below.

evenSubarray has the following parameter(s):
int numbers[n]: an array of integers
int k: the maximum number of odd elements that can be in a subarray
Return
int: the number of distinct subarrays that can be formed as described
Constraints

- *1 ≤ n ≤ 1000*
- *1 ≤ k ≤ n*
- *1 ≤ numbers[i]≤ 250*

Sample Input 0

```
STDIN    Function
-----    --------
4    →   numbers[] size n = 4
6    →   numbers = [6, 3, 5, 8]
3
5
8
1    →   k = 1
```

Sample Output 0

```
6
```

Explanation 0
The distinct subarrays that can be formed are:

- 0 odd elements: *[6]* and *[8]*.
- 1 odd element: *[6, 3]*, *[3]*, *[5]*, and *[5, 8]*

3. Inversions

A subsequence is created by deleting zero or more elements from a list while maintaining the order. For example, the subsequences of *[1,2,3]* are *[1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]*. An *inversion* is a strictly decreasing subsequence of length *3*. More formally, given an array, $p = p[n]$ an *inversion* in the array is any time some $p[i] > p[j] > p[k]$ and $i < j < k$.

Determine the number of inversions within a given array.

Example
n = 5
arr = [5,3,4,2,1].

The array inversions are:

```
[5,3,2]
[5,3,1]
[5,4,2]
[5,4,1]
[5,2,1]
[3,2,1]
[4,2,1]
```

Example 2
n = 4
prices = [4,2,2,1].

The only inversion is [4, 2, 1] and there are two instances: indices 0, 1, 3 and indices 0, 2, 3. The arrays [4, 2, 2] and [2, 2, 1] are not considered inversions because they are not strictly decreasing.

Function Description
Complete the function *maxInversions* in the editor below.

maxInversions has the following parameter(s):
int prices[n]: an array of integers
Returns
long: a long integer denoting the number of inversions in the array.

Constraints

- $1 \le n \le 5000$
- $1 \le arr[i] \le 10^6$, where $0 \le i < n$

There is a simplified version of this question :

- finding number of duplet-inversion
- finding number of triplet-inversion

4. Counting Binary Substrings

A substring is a group of contiguous characters in a string. For instance, all substrings of *abc* are *[a, b, c, ab, bc, abc]*.

Given a binary representation of a number, determine the total number of substrings present that match the following conditions:

1. The *0*'s and *1*'s are grouped consecutively (e.g., *01, 10, 0011, 1100, 000111*, etc.).
2. The number of *0*'s in the substring is equal to the number of *1*'s in the substring.

As an example, consider the string *001101*. The *4* substrings matching the two conditions include *[0011, 01, 10, 01]*. Note that *01* appears twice, from indices *1-2* and *4-5*. There are other substrings, e.g. *001* and *011* that match the first condition but not the second.

Function Description
Complete the function *counting* in the editor below.

counting has the following parameter(s):
string s: a string representation of a binary integer

Returns
int: the number of substrings of *s* that satisfy the two conditions

Constraints

- $5 \leq |s| \leq 5 \times 10^5$
- *each s[i] is either '0' or '1'*

▶ **Input Format for Custom Testing**

▼ **Sample Case 0**

Sample Input 0

```
STDIN       Function Parameters
-----       -------------------
00110   →   s = "00110"
```

Sample Output 0

```
3
```