The brute force approach will solve the first test case.

We will solve for the large subtask

I will explain this using an Example

Let's take an array A=[4,5,1,2]

We will calculate the contribution of each position in the total result.

let's consider the third position, i.e element "1"

All the sub-array with "1" as the ending elements are

[1], [5,1], [4,5,1]

contribution of 1 in each was

for i in 1 to K

[1]=1\*1^i

[5,1]=1\*2^i

[4,5,1]=1\*3^i

I.e. contribution of “1” in subarrays ending with “1” is

1\*(1^i+2^i….P^i) (P is the position of “1”)

in other sub-arrays with "1", it has the same contribution as the above

i.e.

[1,2]=1\*1^i

[5,1,2]=1\*2^i

[4,5,1,2]=1\*3^i

We could see that as we get more number of elements on the right side of "1" (plus 1) in the original array, its contribution increases by that many time

Therefore, we can say that contribution of "1" at position "3" is

2\*1\*(1^i+2^i+3^i)

I.e.

(N-P+1)\*1\*1^i+2^i..P^i)

N= length of the original array

P= position of the element(“1”)

i.e. for an element Ax at position "P" and array of size "N", its Total contribution will be

(N-P+1)\*Ax\*Σm=1k Σi=1P im

Now lets consider the Two summations

For i=1

Σm=1k 1m

For i=2

Σm=1k (1m+2m)

For i=3

Σm=1k (1m+2m+3m)

If Function F() returns the value of the above summation, then

F[ i ] = F[ i - 1 ] + Σm=1kPm (P i position of element )

Where Σm=1kPm =P\*(P^K -1)/(P-1) {GP summation}

The sudo code becomes

For i in 2 to N:

F+=P\*(P^K -1)/(P-1)

Ans+=(N-P+1)\*Ax\*F

\*Note- For P=1 F is K\* i.e. we solve that separately

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Now we have to solve the integer overflow problem and modulus problem

Basic Modulus operation

1. (A\*B)%C = (A%C\*B%C)%C
2. (A^B)%C=(A%C^B%C)%C

We make our own Fast Exponential function which calculates power with inbuild modulus operations

I.e.

For pow(7,14)

We solve it by breaking it in (7^8%MOD \* 7^4%MOD \* 7^2%MOD)

{MOD is 1000000007, given to us)

Where we find the value of rightmost and then substitute in its left neighbor.

The elements can be written in terms of their right-side elements

For more information check <https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/fast-modular-exponentiation>

As Division is not possible in Modulus , We write (1/(P-1))%MOD as it’s inverse

Which is pow(P-1,MOD-2)

Therefore the equation becomes (P\*(pow(P,K)-1)\*pow(P-1,MOD-2))%MOD

After taking MOD everywhere

We finally reach the equation in Code

Note\* F is Sum in Code

Sum for P=1 is K as the equation does not work for P=1