2. K-Subarrays

A *k-subarray* of an array is:

- a subarray (contiguous elements)
- the sum of the elements is evenly divisible by *k (sum modulo k = 0).*

Given an array of integers, determine the number of *k-subarrays* it contains.

Example
k = 5
nums = [5, 10, 11, 9, 5]

The *10 k-subarrays* are *{5}, {5, 10}, {5, 10, 11, 9}, {5, 10, 11, 9, 5}, {10}, {10, 11, 9}, {10, 11, 9, 5}, {11, 9}, {11, 9, 5}, {5}.*

Function Description
Complete the function *kSub* in the editor below. The function must return a long integer that represents the number of *k-subarrays* in the array.

kSub has the following parameter(s):
int k: the divisor of a k-subarray
int nums[n]: an array of integers

Return
long int: the number of k-subarrays in *nums*

Constraints

- $1 \le n \le 3 \times 10^5$
- $1 \le k \le 100$
- $1 \le nums[i] \le 10^4$

▸ Input Format For Custom Testing

▾ Sample Case 0

Sample Input For Custom Testing
Sample Input 0

```
STDIN     Function
-----     --------
3         k = 3
5         nums[] size n = 5
1         nums = [1, 2, 3, 4, 1]
2
3
4
1
```

Sample Output 0

```
4
```

Explanation 0

The subarrays whose sums are evenly divisible by *k = 3* are *{3}, {1, 2}, {1, 2, 3}, {2, 3, 4}.*

3. Visiting Cities

There are a number of cities in a row, and there are two bus lines that go between them. They both visit all cities in order, but one may take longer than the other to go between any two cities. Starting on or moving to the Blue line takes a certain amount of extra time. There is no extra time required to start on or move to the Red line. Determine the minimum cost to move from the first city to each of the cities.

Example
red = [2, 3, 4]
blue = [3, 1, 1]
blueCost = 2

There are *4* cities numbered *0* through *3*. Times from city 0 to cities 1, 2, and 3 are at indices 0, 1, and 2 respectively in the *red* and *blue* arrays.
Through the explanation, an answer array, *ans*, will be created.

- The minimum cost to go from city 0 to itself is 0. Now *ans =[0]*
- The time from city 0 to city 1 is
  - *2* on the Red line
  - *3 + blueCost = 5* on the Blue line.
    - The *blueCost* applies when you start on the Blue line.
  - The minimum time to city 1 is *2* on the Red line, so *ans = [0, 2]*.
- Continuing to city 2:
  - stay on the Red line, arriving at *2 + 3 = 5*
  - switch to the Blue line and arrive at *2 + 1 + 2 = 5.*
    - The *blueCost* applies when you switch to the Blue line.
  - In this case, you arrive at time *5* regardless of the carrier on the second leg. Now *ans = [0, 2, 5]*.
- To get to city 3:
  - take the Red line, arriving at *5 + 4 =9*
  - stay on the Blue line arriving at *5 + 1 = 6.*
    - The *blueCost* does not apply if you stay on the Blue line or move to the Red line.
- The final *ans* array is *[0, 2, 5, 6]*.

Function Description
Complete the function *minimumCost* in the editor below.
minimumCost has the following parameters:
int red[n]: the times to travel on the Red line
int blue[n]: the times to travel on the Blue line
int blueCost: the time penalty to start on or switch to the Blue line

Returns
int[n]: the minimum cost of visiting each of the other cities from city 0

Constraints

- $2 \le n \le 2 \times 10^5$
- $1 \le red[i], blue[i], blueCost \le 10^9$

▾ Input Format For Custom Testing

The first line contains an integer, *n,* the size of *red[]*.
Each of the following *n* lines contains an integer, *red[i].*
The next line contains an integer, *n,* the size of *blue[]*.
Each of the following *n* lines contains an integer, *blue[i].*
The last line contains an integer, *blueCost.*

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
1        red[] size n = 1
5        red = [5]
1        blue[] size n = 1
3        blue = [3]
1        blueCost = 1
```

Sample Output

```
0
4
```

Explanation
You are already at city-0 so the time taken to reach city-0 is 0.
The time to reach city-1 from city-0:
take the Red line: 5.
take the Blue line: 3 + 1 = 4.
The minimum time to reach city-1 is 4.

▾ Sample Case 1

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
2        red[] size n = 2
40       red = [40, 20]
20
2        blue[] size n = 2
30       blue = [30, 25]
25
5        blueCost = 5
```

Sample Output

```
0
35
55
```

Explanation
The time to reach city 0 is always 0.
The time to reach city 1 is:
Red line: 40
Blue line: 30 + 5 = 35
The minimum is 35 on the Blue line.
The time to reach city 2 is:
Red line: 35 + 20 = 55
Blue line: 35 + 25 = 60 (there is no penalty since you were already on the Blue line)
The minimum is 55.