Codeforces Round #687 (Div. 1, based on Technocup 2021 Elimination Round 2)

A. Bouncing Ball

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You're creating a game level for some mobile game. The level should contain some number of cells aligned in a row from left to right and numbered with consecutive integers starting from 1, and in each cell you can either put a platform or leave it empty.

In order to pass a level, a player must throw a ball from the left so that it first lands on a platform in the cell p, then bounces off it, then bounces off a platform in the cell $(p + k)$, then a platform in the cell $(p + 2k)$, and so on every k-th platform until it goes farther than the last cell. If any of these cells has no platform, you can't pass the level with these p and k.

You already have some level pattern a_1, a_2, a_3, ..., a_n, where $a_i = 0$ means there is no platform in the cell i, and $a_i = 1$ means there is one. You want to modify it so that the level can be passed with given p and k. In x seconds you can add a platform in some empty cell. In y seconds you can remove the first cell completely, reducing the number of cells by one, and renumerating the other cells keeping their order. You can't do any other operation. You **can not** reduce the number of cells to less than p.

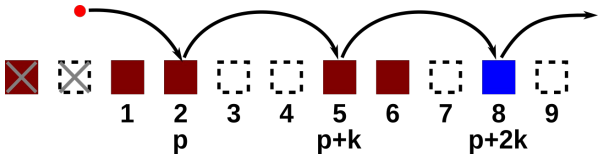

Illustration for the third example test case. Crosses mark deleted cells. Blue platform is the newly added.

What is the minimum number of seconds you need to make this level passable with given p and k?

Input

The first line contains the number of test cases t ($1 \le t \le 100$). Description of test cases follows.

The first line of each test case contains three integers n, p, and k ($1 \le p \le n \le 10^5$, $1 \le k \le n$) — the number of cells you have, the first cell that should contain a platform, and the period of ball bouncing required.

The second line of each test case contains a string $a_1a_2a_3 \dots a_n$ ($a_i = 0$ or $a_i = 1$) — the initial pattern written **without spaces**.

The last line of each test case contains two integers x and y ($1 \le x, y \le 10^4$) — the time required to add a platform and to remove the first cell correspondingly.

The sum of n over test cases does not exceed 10^5.

Output

For each test case output a single integer — the minimum number of seconds you need to modify the level accordingly.

It can be shown that it is always possible to make the level passable.

Example

input
3 10 3 2 0101010101 2 2 5 4 1 00000 2 10 11 2 3 10110011000 4 3
output
2 4 10

Note

In the first test case it's best to just remove the first cell, after that all required platforms are in their places: ~~0~~**1**0**1**0**1**0**1**0**1**. The stroked out digit is removed, the bold ones are where platforms should be located. The time required is $y = 2$.

In the second test case it's best to add a platform to both cells 4 and 5: `00`**`00`** $\rightarrow$ `00`**`11`**. The time required is $x \cdot 2 = 4$.

In the third test case it's best to to remove the first cell twice and then add a platform to the cell which was initially 10-th: ~~**1**0~~**1**1**0**0**1**1**0**~~**0**~~ $\rightarrow$ ~~**1**0~~**1**1**0**0**1**1**0**~~**1**~~**0**. The time required is $y \cdot 2 + x = 10$.

B. XOR-gun

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Arkady owns a **non-decreasing** array $a_1, a_2, \dots, a_n$. You are jealous of its beauty and want to destroy this property. You have a so-called *XOR-gun* that you can use one or more times.

In one step you can select two **consecutive** elements of the array, let's say x and y, remove them from the array and insert the

integer $x \oplus y$ on their place, where $\oplus$ denotes the bitwise XOR operation. Note that the length of the array decreases by one after the operation. You can't perform this operation when the length of the array reaches one.

For example, if the array is $[2, 5, 6, 8]$, you can select 5 and 6 and replace them with $5 \oplus 6 = 3$. The array becomes $[2, 3, 8]$.

You want the array no longer be non-decreasing. What is the minimum number of steps needed? If the array stays non-decreasing no matter what you do, print -1.

Input
The first line contains a single integer n ($2 \le n \le 10^5$) — the initial length of the array.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the elements of the array. It is guaranteed that $a_i \le a_{i+1}$ for all $1 \le i < n$.

Output
Print a single integer — the minimum number of steps needed. If there is no solution, print -1.

Examples

input
4 2 5 6 8
output
1

input
3 1 2 3
output
-1

input
5 1 2 4 6 20
output
2

Note
In the first example you can select 2 and 5 and the array becomes $[7, 6, 8]$.

In the second example you can only obtain arrays $[1, 1]$, $[3, 3]$ and $[0]$ which are all non-decreasing.

In the third example you can select 1 and 2 and the array becomes $[3, 4, 6, 20]$. Then you can, for example, select 3 and 4 and the array becomes $[7, 6, 20]$, which is no longer non-decreasing.

C. New Game Plus!

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Wabbit is playing a game with n bosses numbered from 1 to n. The bosses can be fought in any order. Each boss needs to be defeated **exactly once**. There is a parameter called **boss bonus** which is initially 0.

When the i-th boss is defeated, the current **boss bonus** is added to Wabbit's score, and then the value of the **boss bonus** increases by the point increment c_i. Note that c_i can be negative, which means that other bosses now give fewer points.

However, Wabbit has found a glitch in the game. At any point in time, he can reset the playthrough and start a New Game Plus playthrough. This will set the current **boss bonus** to 0, while all defeated bosses remain defeated. The current score is also saved and does **not** reset to zero after this operation. This glitch can be used **at most** k times. He can reset after defeating any number of bosses (including before or after defeating all of them), and he also can reset the game several times in a row without defeating any boss.

Help Wabbit determine the maximum score he can obtain if he has to defeat **all** n bosses.

Input
The first line of input contains two spaced integers n and k ($1 \le n \le 5 \cdot 10^5$, $0 \le k \le 5 \cdot 10^5$), representing the number of bosses and the number of resets allowed.

The next line of input contains n spaced integers $c_1, c_2, \ldots, c_n$ ($-10^6 \le c_i \le 10^6$), the point increments of the n bosses.

Output
Output a single integer, the maximum score Wabbit can obtain by defeating all n bosses (this value may be negative).

Examples

input
3 0 1 1 1
output
3

input
5 1 -1 -2 -3 -4 5

output
11

input
13 2 3 1 4 1 5 -9 -2 -6 -5 -3 -5 -8 -9
output
71

Note
In the first test case, no resets are allowed. An optimal sequence of fights would be

- Fight the first boss $(+0)$. Boss bonus becomes equal to 1.
- Fight the second boss $(+1)$. Boss bonus becomes equal to 2.
- Fight the third boss $(+2)$. Boss bonus becomes equal to 3.

Thus the answer for the first test case is $0 + 1 + 2 = 3$.

In the second test case, it can be shown that one possible optimal sequence of fights is

- Fight the fifth boss $(+0)$. Boss bonus becomes equal to 5.
- Fight the first boss $(+5)$. Boss bonus becomes equal to 4.
- Fight the second boss $(+4)$. Boss bonus becomes equal to 2.
- Fight the third boss $(+2)$. Boss bonus becomes equal to -1.
- Reset. Boss bonus becomes equal to 0.
- Fight the fourth boss $(+0)$. Boss bonus becomes equal to -4.

Hence the answer for the second test case is $0 + 5 + 4 + 2 + 0 = 11$.

D. Cakes for Clones

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You live on a number line. You are initially (at time moment $t = 0$) located at point $x = 0$. There are n events of the following type: at time t_i a small cake appears at coordinate x_i. To collect this cake, you have to be at this coordinate at this point, otherwise the cake spoils immediately. No two cakes appear at the same time and no two cakes appear at the same coordinate.

You can move with the speed of 1 length unit per one time unit. Also, at any moment you can create a clone of yourself at the same point where you are located. The clone can't move, but it will collect the cakes appearing at this position for you. The clone **disappears** when you create another clone. If the new clone is created at time moment t, the old clone can collect the cakes that appear before or at the time moment t, and the new clone can collect the cakes that appear at or after time moment t.

Can you collect all the cakes (by yourself or with the help of clones)?

Input
The first line contains a single integer n ($1 \le n \le 5000$) — the number of cakes.

Each of the next n lines contains two integers t_i and x_i ($1 \le t_i \le 10^9$, $-10^9 \le x_i \le 10^9$) — the time and coordinate of a cake's appearance.

All times are distinct and are given in increasing order, all the coordinates are **distinct**.

Output
Print "YES" if you can collect all cakes, otherwise print "NO".

Examples

input
3 2 2 5 5 6 1
output
YES

input
3 1 0 5 5 6 2
output
YES

input
3 2 1 5 5 6 0
output
NO

Note
In the first example you should start moving towards 5 right away, leaving a clone at position 1 at time moment 1, and collecting the cake at position 2 at time moment 2. At time moment 5 you are at the position 5 and collect a cake there, your clone collects

the last cake at time moment 6.

In the second example you have to leave a clone at position 0 and start moving towards position 5. At time moment 1 the clone collects a cake. At time moment 2 you should create a new clone at the current position 2, it will collect the last cake in future. You will collect the second cake at position 5.

In the third example there is no way to collect all cakes.

E. XOR-ranges

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Given integers $c_0, c_1, \ldots, c_{k-1}$ we can define the cost of a number $0 \le x < 2^k$ as $p(x) = \sum_{i=0}^{k-1} \left(\left\lfloor \frac{x}{2^i} \right\rfloor \bmod 2 \right) \cdot c_i$. In other words, the cost of number x is the sum of c_i over the bits of x which are equal to one.

Let's define the cost of array a of length $n \ge 2$ with elements from $[0, 2^k)$ as follows: $cost(a) = \sum_{i=1}^{n-1} p(a_i \oplus a_{i+1})$, where $\oplus$ denotes bitwise exclusive OR operation.

You have to construct an array of length n with minimal cost, given that each element should belong to the given segment: $l_i \le a_i \le r_i$.

Input

The first line contains two integers n and k ($2 \le n \le 50$, $1 \le k \le 50$) — the size of an array and bit length of the numbers in question.

Next n lines contain the restrictions for elements of the array: the i-th line contains two integers l_i and r_i ($0 \le l_i \le r_i < 2^k$).

The last line contains integers $c_0, c_1, \ldots, c_{k-1}$ ($0 \le c_i \le 10^{12}$).

Output

Output one integer — the minimal cost of an array satisfying all the restrictions.

Examples

input
4 3 3 3 5 5 6 6 1 1 5 2 7
output
30

input
3 3 2 2 3 4 4 6 1 10 100
output
102

Note

In the first example there is only one array satisfying all the restrictions — $[3, 5, 6, 1]$ — and its cost is equal to $cost([3, 5, 6, 1]) = p(3 \oplus 5) + p(5 \oplus 6) + p(6 \oplus 1) = p(6) + p(3) + p(7) = (c_1 + c_2) + (c_0 + c_1) + (c_0 + c_1 + c_2) = (2 + 7) + (5 + 2) + (5 +$.

In the second example the only optimal array is $[2, 3, 6]$.