Codeforces Round #770 (Div. 2)

A. Reverse and Concatenate

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

> Real stupidity beats artificial intelligence every time.
>
> — Terry Pratchett, *Hogfather, Discworld*

You are given a string s of length n and a number k. Let's denote by $rev(s)$ the reversed string s (i.e. $rev(s) = s_n s_{n-1} \ldots s_1$). You can apply one of the two kinds of operations to the string:

- replace the string s with $s + rev(s)$
- replace the string s with $rev(s) + s$

How many different strings can you get as a result of performing **exactly** k operations (possibly of different kinds) on the original string s?

In this statement we denoted the concatenation of strings s and t as $s + t$. In other words, $s + t = s_1 s_2 \ldots s_n t_1 t_2 \ldots t_m$, where n and m are the lengths of strings s and t respectively.

Input

The first line contains one integer t ($1 \le t \le 100$) — number of test cases. Next $2 \cdot t$ lines contain t test cases:

The first line of a test case contains two integers n and k ($1 \le n \le 100, 0 \le k \le 1000$) — the length of the string and the number of operations respectively.

The second string of a test case contains one string s of length n consisting of lowercase Latin letters.

Output

For each test case, print the answer (that is, the number of different strings that you can get after exactly k operations) on a separate line.

It can be shown that the answer does not exceed 10^9 under the given constraints.

Example

input
4 3 2 aab 3 3 aab 7 1 abacaba 2 0 ab

output
2 2 1 1

Note

In the first test case of the example:

After the first operation the string s can become either **aabbaa** or **baaaab**. After the second operation there are 2 possibilities for s: **aabbaaaabbaa** and **baaaabbaaaab**.

B. Fortune Telling

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

> Haha, try to solve this, SelectorUnlimited!
>
> — antontrygubO_o

Your friends Alice and Bob practice fortune telling.

Fortune telling is performed as follows. There is a well-known array a of n non-negative integers indexed from 1 to n. The tellee starts with some non-negative number d and performs one of the two operations for each $i = 1, 2, \ldots, n$, **in the increasing order of i**. The possible operations are:

- replace their current number d with $d + a_i$
- replace their current number d with $d \oplus a_i$ (hereinafter $\oplus$ denotes the [bitwise XOR operation](#))

Notice that the chosen operation may be different for different i and for different tellees.

One time, Alice decided to start with $d = x$ and Bob started with $d = x + 3$. Each of them performed fortune telling and got a particular number in the end. Notice that the friends chose operations independently of each other, that is, they could apply different operations for the same i.

You learnt that either Alice or Bob ended up with number y in the end, but you don't know whose of the two it was. Given the numbers Alice and Bob started with and y, find out who (Alice or Bob) could get the number y after performing the operations. It is guaranteed that on the jury tests, **exactly one** of your friends could have actually gotten that number.

Hacks

You cannot make hacks in this problem.

Input
On the first line of the input, you are given one number t ($1 \le t \le 10^4$) — the number of test cases. The following $2 \cdot t$ lines contain test cases.

The first line of each test case contains three numbers n, x, y ($1 \le n \le 10^5$, $0 \le x \le 10^9$, $0 \le y \le 10^{15}$) — the length of array a, Alice's initial number (Bob's initial number is therefore $x + 3$), and the number that one of the two friends got in the end.

The second line of each test case contains n numbers — the array a ($0 \le a_i \le 10^9$).

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output
For each test case, print the name of the friend who could get the number y: "Alice" or "Bob".

Example

input
4
1 7 9
2
2 0 2
1 3
4 0 1
1 2 3 4
2 1000000000 3000000000
1000000000 1000000000

output
Alice
Alice
Bob
Alice

Note
In the first test case, Alice could get 9 using the following operations: $7 + 2 = 9$.

In the second test case, Alice could get 2 using this operations: $(0 + 1) \oplus 3 = 2$.

In the third test case, Bob started with $x + 3 = 0 + 3 = 3$ and could get 1 this way: $(((3 + 1) + 2) \oplus 3) \oplus 4 = 1$.

C. OKEA

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

> People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world.
>
> — Pedro Domingos

You work for a well-known department store that uses leading technologies and employs mechanistic work — that is, robots!

The department you work in sells $n \cdot k$ items. The first item costs 1 dollar, the second item costs 2 dollars, and so on: i-th item costs i dollars. The items are situated on shelves. The items form a rectangular grid: there are n shelves in total, and each shelf contains exactly k items. We will denote by $a_{i,j}$ the price of j-th item (counting from the left) on the i-th shelf, $1 \le i \le n, 1 \le j \le k$.

Occasionally robots get curious and ponder on the following question: what is the mean price (arithmetic average) of items $a_{i,l}, a_{i,l+1}, \ldots, a_{i,r}$ for some shelf i and indices $l \le r$? Unfortunately, the old robots can only work with whole numbers. If the mean price turns out not to be an integer, they break down.

You care about robots' welfare. You want to arrange the items in such a way that the robots cannot theoretically break. Formally, you want to choose such a two-dimensional array a that:

- Every number from 1 to $n \cdot k$ (inclusively) occurs exactly once.
- For each i, l, r, the mean price of items from l to r on i-th shelf is an integer.

Find out if such an arrangement is possible, and if it is, give any example of such arrangement.

Input
The first line contains a single integer t ($1 \le t \le 500$) — the number of test cases.

The first and only line of each test case contains two integers n and k ($1 \le n, k \le 500$) — the number of shelves and length of each shelf, respectively.

It is guaranteed that the sum n over all test cases does not exceed 500 and the sum k over all test cases does not exceed 500.

Output
Print the answer for each test case.

If such an arrangement exists, print "YES" on a single line. After that, print any example on n lines of k numbers each, one line per shelf. Each number from 1 to $n \cdot k$ must occur exactly once in the output.

If no good arrangement exists, print a single word "NO" on its own line.

Example

input
4
1 1
2 2
3 3
3 1

output
YES
1
YES
1 3
2 4
NO
YES
1
2
3

D. Finding Zero

SHALL WE PLAY A GAME?

This is an interactive problem.

We picked an array of whole numbers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) and concealed **exactly one** zero in it! Your goal is to find the location of this zero, that is, to find i such that $a_i = 0$.

You are allowed to make several queries to guess the answer. For each query, you can think up three distinct indices i, j, k, and we will tell you the value of $\max(a_i, a_j, a_k) - \min(a_i, a_j, a_k)$. In other words, we will tell you the difference between the maximum and the minimum number among a_i, a_j and a_k.

You are allowed to make no more than $2 \cdot n - 2$ queries, and after that you have two tries to guess where the zero is. That is, you have to tell us two numbers i and j and you win if $a_i = 0$ or $a_j = 0$.

Can you guess where we hid the zero?

Note that the array in each test case is fixed beforehand and will not change during the game. In other words, the interactor is not adaptive.

Input
Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 500$). Description of the test cases follows.

The first and only line of each test case contains an integer n ($4 \leq n \leq 1000$) — the length of the array that we picked.

It is guaranteed that the sum of n over all test cases does not exceed 3000.

Interaction

For each test case, the interaction starts with reading n.

To make a query, print "? i j k" (without quotes, $1 \leq i, j, k \leq n$, indices must be distinct). Then you should read our response from standard input, that is, $\max(a_i, a_j, a_k) - \min(a_i, a_j, a_k)$.

If the response is -1, it means your program has made an invalid query or has run out of tries. Your program must terminate immediately after reading -1, and you will get a verdict `Wrong answer`. Otherwise you may get any verdict, because the program will continue reading from the closed stream. Note that if the query is correct, the answer will never be -1 because $\max(a_i, a_j, a_k) - \min(a_i, a_j, a_k) \geq 0$.

To give the final answer, print "! i j" (without the quotes). Printing the same number twice (that is, $i = j$) is allowed. Note that giving this answer is not counted towards the limit of $2 \cdot n - 2$ queries. After that, your program must continue to solve the remaining test cases, or exit if all test cases have been solved.

After printing a query, don't forget to output line feed and flush the output buffer. Otherwise you will get the verdict `Idleness limit exceeded`. To flush the buffer, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- Read documentation for other languages.

Hacks

The first line must contain an integer t ($1 \leq t \leq 500$) — the count of test cases.

The first line of each test case must contain an integer n ($4 \leq n \leq 1000$) — the length of the hidden array.

The second line of each test case must contain n integers separated by spaces — $a_1, a_2, \ldots, a_n$ ($0 \leq a_i \leq 10^9$). There must also be **exactly one** zero in this array.

The sum of n over all test cases must not exceed 3000.

Example

input
1
4
2
3
3
2

output
? 1 2 3
? 2 3 4
? 3 4 1
? 4 1 2
! 2 3

Note
Array from sample: $[1, 2, 0, 3]$.

E. Fair Share

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Even a cat has things it can do that AI cannot.

— Fei-Fei Li

You are given m arrays of positive integers. Each array is of even length.

You need to split all these integers into two **equal** multisets L and R, that is, each element of each array should go into one of two multisets (but not both). Additionally, for each of the m arrays, **exactly half** of its elements should go into L, and the rest should go into R.

Give an example of such a division or determine that no such division exists.

Input

The first line contains an integer m ($1 \le m \le 10^5$) — the number of arrays.

The next $2 \cdot m$ lines contain descriptions of the arrays.

For each array, the first line contains an even integer n ($2 \le n \le 2 \cdot 10^5$) — the length of the array. The second line consists of n space-separated integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — array elements.

It is guaranteed that the sum of n over all arrays does not exceed $2 \cdot 10^5$.

Output

If the answer exists, print "YES", and then print m lines.

On each line, for each element, print the letter "L" or "R" (capitalized, without spaces), depending on which multiset the element should go into.

If there is no answer, print "NO" on the only line.

Example

input
3 2 1 2 4 1 2 3 3 6 1 1 2 2 3 3

output
YES RL LRLR RLLRRL

Note

In the first array, we add the first element to R and the second to L. Now $L = \{2\}$, and $R = \{1\}$.

In the second array, we add the first and third elements to L and the rest to R. Now $L = \{1, 2, 3\}$ and $R = \{1, 2, 3\}$.

In the third array, we add elements 2, 3, and 6 to L, and others — to R. As a result, $L = R = \{1, 1, 2, 2, 3, 3\}$.

F. Fibonacci Additions

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

> One of my most productive days was throwing away 1,000 lines of code.
>
> — Ken Thompson

Fibonacci addition is an operation on an array X of integers, parametrized by indices l and r. Fibonacci addition increases X_l by F_1, increases X_{l+1} by F_2, and so on up to X_r which is increased by F_{r-l+1}.

F_i denotes the i-th Fibonacci number ($F_1 = 1, F_2 = 1, F_i = F_{i-1} + F_{i-2}$ for $i > 2$), and **all operations are performed modulo MOD**.

You are given two arrays A and B of the same length. We will ask you to perform several Fibonacci additions on these arrays with different parameters, and after each operation you have to report whether arrays A and B are equal modulo MOD.

Input

The first line contains 3 numbers n, q and MOD ($1 \le n, q \le 3 \cdot 10^5, 1 \le MOD \le 10^9 + 7$) — the length of the arrays, the number of operations, and the number modulo which all operations are performed.

The second line contains n numbers — array A ($0 \le A_i < MOD$).

The third line also contains n numbers — array B ($0 \le B_i < MOD$).

The next q lines contain character c and two numbers l and r ($1 \le l \le r \le n$) — operation parameters. If c is "A", Fibonacci addition is to be performed on array A, and if it is is "B", the operation is to be performed on B.

Output

After each operation, print "YES" (without quotes) if the arrays are equal and "NO" otherwise. Letter case does not matter.

Examples

input
3 5 3 2 2 1 0 0 0 A 1 3 A 1 3 B 1 1 B 2 2 A 3 3
output
YES NO NO NO YES

input
5 3 10 2 5 0 3 5 3 5 8 2 5 B 2 3 B 3 4 A 1 2
output
NO NO YES

Note

Explanation of the test from the condition:

- Initially $A = [2, 2, 1]$, $B = [0, 0, 0]$.
- After operation "A 1 3": $A = [0, 0, 0]$, $B = [0, 0, 0]$ (addition is modulo 3).
- After operation "A 1 3": $A = [1, 1, 2]$, $B = [0, 0, 0]$.
- After operation "B 1 1": $A = [1, 1, 2]$, $B = [1, 0, 0]$.
- After operation "B 2 2": $A = [1, 1, 2]$, $B = [1, 1, 0]$.
- After operation "A 3 3": $A = [1, 1, 0]$, $B = [1, 1, 0]$.