Codeforces Round #803 (Div. 2)

A. XOR Mixup

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

There is an array a with $n-1$ integers. Let x be the bitwise XOR of all elements of the array. The number x is added to the end of the array a (now it has length n), and then the elements are shuffled.

You are given the newly formed array a. What is x? If there are multiple possible values of x, you can output any of them.

Input

The input consists of multiple test cases. The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains an integer n ($2 \le n \le 100$) — the number of integers in the resulting array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 127$) — the elements of the newly formed array a.

Additional constraint on the input: the array a is made by the process described in the statement; that is, some value of x exists.

Output

For each test case, output a single integer — the value of x, as described in the statement. If there are multiple possible values of x, output any of them.

Example

input
4 4 4 3 2 5 5 6 1 10 7 10 6 6 6 6 6 6 6 3 100 100 0
output
3 7 6 0

Note

In the first test case, one possible array a is $a = [2, 5, 4]$. Then $x = 2 \oplus 5 \oplus 4 = 3$ ($\oplus$ denotes the bitwise XOR), so the new array is $[2, 5, 4, 3]$. Afterwards, the array is shuffled to form $[4, 3, 2, 5]$.

In the second test case, one possible array a is $a = [1, 10, 6, 10]$. Then $x = 1 \oplus 10 \oplus 6 \oplus 10 = 7$, so the new array is $[1, 10, 6, 10, 7]$. Afterwards, the array is shuffled to form $[6, 1, 10, 7, 10]$.

In the third test case, all elements of the array are equal to 6, so $x = 6$.

In the fourth test case, one possible array a is $a = [100, 100]$. Then $x = 100 \oplus 100 = 0$, so the new array is $[100, 100, 0]$. Afterwards, the array is shuffled to form $[100, 100, 0]$. (Note that after the shuffle, the array can remain the same.)

B. Rising Sand

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

There are n piles of sand where the i-th pile has a_i blocks of sand. The i-th pile is called *too tall* if $1 < i < n$ and $a_i > a_{i-1} + a_{i+1}$. That is, a pile is too tall if it has more sand than its two neighbours combined. (Note that piles on the ends of the array cannot be too tall.)

You are given an integer k. An operation consists of picking k consecutive piles of sand and adding one unit of sand to them all. Formally, pick $1 \le l, r \le n$ such that $r - l + 1 = k$. Then for all $l \le i \le r$, update $a_i \leftarrow a_i + 1$.

What is the **maximum** number of piles that can simultaneously be too tall after some (possibly zero) operations?

Input
The input consists of multiple test cases. The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and k ($3 \le n \le 2 \cdot 10^5$; $1 \le k \le n$) — the number of piles of sand and the size of the operation, respectively.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the sizes of the piles.

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

Output
For each test case, output a single integer — the maximum number of piles that are simultaneously too tall after some (possibly zero) operations.

Example

input
3 5 2 2 9 2 4 1 4 4 1 3 2 1 3 1 1 3 1
output
2 0 1

Note
In the first test case, we can perform the following three operations:

- Add one unit of sand to piles 1 and 2: $[3, 10, 2, 4, 1]$.
- Add one unit of sand to piles 4 and 5: $[3, 10, 2, 5, 2]$.
- Add one unit of sand to piles 3 and 4: $[3, 10, 3, 6, 2]$.

Now piles 2 and 4 are too tall, so in this case the answer is 2. It can be shown that it is impossible to make more than 2 piles too tall.
In the second test case, any operation will increase all piles by 1 unit, so the number of too tall piles will always be 0.

In the third test case, we can increase any pile by 1 unit of sand. It can be shown that the maximum number of too tall piles is 1.

C. 3SUM Closure

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

You are given an array a of length n. The array is called *3SUM-closed* if for all distinct indices i, j, k, the sum $a_i + a_j + a_k$ is an element of the array. More formally, a is 3SUM-closed if for all integers $1 \le i < j < k \le n$, there exists some integer $1 \le l \le n$ such that $a_i + a_j + a_k = a_l$.

Determine if a is 3SUM-closed.

Input
The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains an integer n ($3 \le n \le 2 \cdot 10^5$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) — the elements of the array.

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

Output
For each test case, output "YES" (without quotes) if a is 3SUM-closed and "NO" (without quotes) otherwise.

You can output "YES" and "NO" in any case (for example, strings "yEs", "yes" and "Yes" will be recognized as a positive response).

Example

input
4 3 -1 0 1 5 1 -2 -2 1 -3

```
6
0 0 0 0 0 0
4
-1 2 -3 4
```

output

```
YES
NO
YES
NO
```

Note

In the first test case, there is only one triple where $i = 1$, $j = 2$, $k = 3$. In this case, $a_1 + a_2 + a_3 = 0$, which is an element of the array ($a_2 = 0$), so the array is 3SUM-closed.

In the second test case, $a_1 + a_4 + a_5 = -1$, which is not an element of the array. Therefore, the array is not 3SUM-closed.

In the third test case, $a_i + a_j + a_k = 0$ for all distinct i, j, k, and 0 is an element of the array, so the array is 3SUM-closed.

D. Fixed Point Guessing

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

This is an interactive problem.

Initially, there is an array $a = [1, 2, \ldots, n]$, where n is an odd positive integer. The jury has selected $\frac{n-1}{2}$ **disjoint** pairs of elements, and then the elements in those pairs are swapped. For example, if $a = [1, 2, 3, 4, 5]$, and the pairs $1 \leftrightarrow 4$ and $3 \leftrightarrow 5$ are swapped, then the resulting array is $[4, 2, 5, 1, 3]$.

As a result of these swaps, exactly one element will not change position. You need to find this element.

To do this, you can ask several queries. In each query, you can pick two integers l and r ($1 \le l \le r \le n$). In return, you will be given the elements of the subarray $[a_l, a_{l+1}, \ldots, a_r]$ **sorted in increasing order**.

Find the element which did not change position. You can make at most $\mathbf{15}$ queries.

The array a is fixed before the interaction and does not change after your queries.

Recall that an array b is a subarray of the array a if b can be obtained from a by deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end.

Input

Each test contains multiple test cases. The first line contains an integer t ($1 \le t \le 500$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains an integer n ($3 \le n < 10^4$; n is odd) — the length of the array a.

After reading the first line of each test case, you should begin the interaction.

It is guaranteed that the sum of n over all test cases does not exceed 10^4.

Interaction

For each test case, begin the interaction by reading the integer n.

To make a query, output "`? l r`" ($1 \le l \le r \le n$) without quotes. Afterwards, you should read in $r - l + 1$ integers — the integers $a_l, a_{l+1}, \ldots, a_r$, in increasing order. You can make at most 15 such queries in a single test case.

If you receive the integer -1 instead of an answer or the integer n, it means your program has made an invalid query, has exceed the limit of queries, or has given incorrect answer on the previous test case. Your program must terminate immediately to receive a `Wrong Answer` verdict. Otherwise you can get an arbitrary verdict because your solution will continue to read from a closed stream.

When you are ready to give the final answer, output "`! x`" ($1 \le x \le n$) without quotes — the element that did not change position. Giving this answer does not count towards the limit of 15 queries. Afterwards, your program must continue to solve the remaining test cases, or exit if all test cases have been solved.

After printing a query do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

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

Hacks

To make a hack, use the following format. The first line must contain an integer t ($1 \le t \le 500$) — the number of test cases. The description of the test cases follows.

The first line of each test case must contain an integer n ($3 \le n < 10^4$; n is odd) — the length of the array a.

The second line of each test case must contain n space-separated integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements of a. The array a should be the result of $\frac{n-1}{2}$ disjoint swaps on the array $[1, 2, \ldots, n]$.

Example

input
2 5 1 2 4 5 1 3 5 3 1
output
  ? 1 4 ? 3 5 ! 2 ? 1 1 ! 1

Note

In the first test, the interaction proceeds as follows.

Solution	Jury	Explanation
	2	There are 2 test cases.
	5	In the first test case, the hidden array is $[4, 2, 5, 1, 3]$, with length 5.
? 1 4	1 2 4 5	The solution requests the subarray $[4, 2, 5, 1]$ in increasing order, and the jury responds with $[1, 2, 4, 5]$.
? 3 5	1 3 5	The solution requests the subarray $[5, 1, 3]$ in increasing order, and the jury responds with $[1, 3, 5]$.
! 2		The solution has somehow determined that $a_2 = 2$, and outputs it. Since the output is correct, the jury continues to the next test case.
	3	In the second test case, the hidden array is $[1, 3, 2]$, with length 3.
? 1 1	1	The solution requests the number $[1]$ only, and the jury responds with $[1]$.
! 1		The solution has determined that $a_1 = 1$, and outputs it. Since the output is correct and there are no more test cases, the jury and the solution exit.

Note that the line breaks in the example input and output are for the sake of clarity, and do not occur in the real interaction.

E. PermutationForces II

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

You are given a permutation a of length n. Recall that permutation is an array consisting of n distinct integers from 1 to n in arbitrary order.

You have a strength of s and perform n moves on the permutation a. The i-th move consists of the following:

- Pick two integers x and y such that $i \le x \le y \le \min(i + s, n)$, and swap the positions of the integers x and y in the permutation a. Note that you **can** select $x = y$ in the operation, in which case no swap will occur.

You want to turn a into another permutation b after n moves. However, some elements of b are missing and are replaced with -1 instead. Count the number of ways to replace each -1 in b with some integer from 1 to n so that b is a permutation and it is

possible to turn a into b with a strength of s.

Since the answer can be large, output it modulo $998\,244\,353$.

Input

The input consists of multiple test cases. The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and s ($1 \le n \le 2 \cdot 10^5$; $1 \le s \le n$) — the size of the permutation and your strength, respectively.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements of a. All elements of a are distinct.

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le n$ or $b_i = -1$) — the elements of b. All elements of b that are not equal to -1 are distinct.

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

Output

For each test case, output a single integer — the number of ways to fill up the permutation b so that it is possible to turn a into b using a strength of s, modulo $998\,244\,353$.

Example

input
6 3 1 2 1 3 3 -1 -1 3 2 2 1 3 3 -1 -1 4 1 1 4 3 2 4 3 1 2 6 4 4 2 6 3 1 5 6 1 5 -1 3 -1 7 4 1 3 6 2 7 4 5 2 5 -1 -1 -1 4 -1 14 14 1 2 3 4 5 6 7 8 9 10 11 12 13 14 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

output
1 2 0 2 12 331032489

Note

In the first test case, $a = [2, 1, 3]$. There are two possible ways to fill out the -1s in b to make it a permutation: $[3, 1, 2]$ or $[3, 2, 1]$. We can make a into $[3, 1, 2]$ with a strength of 1 as follows:

$$[2, 1, 3] \xrightarrow[x=1,\, y=1]{} [2, 1, 3] \xrightarrow[x=2,\, y=3]{} [3, 1, 2] \xrightarrow[x=3,\, y=3]{} [3, 1, 2].$$

It can be proven that it is impossible to make $[2, 1, 3]$ into $[3, 2, 1]$ with a strength of 1. Thus only one permutation b satisfies the constraints, so the answer is 1.

In the second test case, a and b the same as the previous test case, but we now have a strength of 2. We can make a into $[3, 2, 1]$ with a strength of 2 as follows:

$$[2, 1, 3] \xrightarrow[x=1,\, y=3]{} [2, 3, 1] \xrightarrow[x=2,\, y=3]{} [3, 2, 1] \xrightarrow[x=3,\, y=3]{} [3, 2, 1].$$

We can still make a into $[3, 1, 2]$ using a strength of 1 as shown in the previous test case, so the answer is 2.

In the third test case, there is only one permutation b. It can be shown that it is impossible to turn a into b, so the answer is 0.

F. Equal Reversal

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

There is an array a of length n. You may perform the following operation on it:

- Choose two indices l and r where $1 \le l \le r \le n$ and $a_l = a_r$. Then, reverse the subsegment from the l-th to the r-th element, i. e. set $[a_l, a_{l+1}, \ldots, a_{r-1}, a_r]$ to $[a_r, a_{r-1}, \ldots, a_{l+1}, a_l]$.

You are also given another array b of length n which is a permutation of a. Find a sequence of at most n^2 operations that transforms array a into b, or report that no such sequence exists.

Input
Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 500$) — the length of array a and b.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — elements of the array a.

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le n$) — elements of the array b.

It is guaranteed that b is a permutation of a.

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

Output
For each test case, output "NO" (without quotes) if it is impossible to turn a into b using at most n^2 operations.

Otherwise, output "YES" (without quotes). Then output an integer k ($0 \le k \le n^2$) denoting the number of operations you will perform. Note that you don't have to minimize the number of operations.

Afterwards, output k lines. The i-th line should contain two integers l_i and r_i ($1 \le l_i \le r_i \le n$) — the left and right indices for the i-th operation.

You can output "YES" and "NO" in any case (for example, strings "yEs", "yes" and "Yes" will be recognized as a positive response).

If there are multiple possible sequences of operations, you may output any of them.

Example

input
5 8 1 2 4 3 1 2 1 1 1 1 3 4 2 1 2 1 7 1 2 3 1 3 2 3 1 3 2 3 1 2 3 3 1 1 2 1 2 1 2 1 2 2 1 1 1 1
output
YES 2 5 8 1 6 YES 2 1 4 3 6 NO NO YES 0

Note
In the first test case, we can perform the following operations:

$$[1,2,4,3,1,2,1,1] \xrightarrow[l=5,\, r=8]{} [1,2,4,3,1,1,2,1] \xrightarrow[l=1,\, r=6]{} [1,1,3,4,2,1,2,1].$$

In the second test case, we can perform the following operations:

$$[1,2,3,1,3,2,3] \xrightarrow[l=1,\, r=4]{} [1,3,2,1,3,2,3] \xrightarrow[l=3,\, r=6]{} [1,3,2,3,1,2,3].$$

It can be proven that it is impossible to turn a into b in the third and fourth test cases.

G. Long Binary String

time limit per test: 2 seconds
memory limit per test: 256 megabytes

input: standard input
output: standard output

There is a binary string t of length 10^{100}, and initally all of its bits are 0. You are given a binary string s, and perform the following operation some times:

- Select some substring of t, and replace it with its XOR with s.[†]

After several operations, the string t has exactly two bits 1; that is, there are exactly two distinct indices p and q such that the p-th and q-th bits of t are 1, and the rest of the bits are 0.
Find the lexicographically largest[‡] string t satisfying these constraints, or report that no such string exists.

[†] Formally, choose an index i such that $0 \le i \le 10^{100} - |s|$. For all $1 \le j \le |s|$, if $s_j = 1$, then toggle t_{i+j}. That is, if $t_{i+j} = 0$, set $t_{i+j} = 1$. Otherwise if $t_{i+j} = 1$, set $t_{i+j} = 0$.

[‡] A binary string a is lexicographically larger than a binary string b of the same length if in the first position where a and b differ, the string a has a bit 1 and the corresponding bit in b is 0.

Input
The only line of each test contains a single binary string s ($1 \le |s| \le 35$).

Output
If no string t exists as described in the statement, output -1. Otherwise, output the integers p and q ($1 \le p < q \le 10^{100}$) such that the p-th and q-th bits of the lexicographically maximal t are 1.

Examples

input
1
output
1 2

input
001
output
3 4

input
1111
output
1 5

input
00000
output
-1

input
00000111110000011111000001111101010
output
6 37452687

Note
In the first test, you can perform the following operations.

$$00000\ldots \to 10000\ldots \to 11000\ldots$$

In the second test, you can perform the following operations.

$$00000\ldots \to 00100\ldots \to 00110\ldots$$

In the third test, you can perform the following operations.

$$00000\ldots \to 11110\ldots \to 10001\ldots$$

It can be proven that these strings t are the lexicographically largest ones.

In the fourth test, you can't make a single bit 1, so it is impossible.