A. MEX Destruction

1 second, 256 megabytes

Evirir the dragon snuck into a wizard's castle and found a mysterious contraption, and their playful instincts caused them to play with (destroy) it...

Evirir the dragon found an array $a_1, a_2, \ldots, a_n$ of n non-negative integers.

In one operation, they can choose a non-empty subarray* b of a and replace it with the integer $\operatorname{mex}(b)^\dagger$. They want to use this operation any number of times to make a only contain zeros. It can be proven that this is always possible under the problem constraints.

What is the minimum number of operations needed?

*An array c is a subarray of an array d if c can be obtained from d by the deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end.

†The minimum excluded (MEX) of a collection of integers $f_1, f_2, \ldots, f_k$ is defined as the smallest non-negative integer x which does not occur in the collection f.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 200$). The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 50$), the length of a.

The second line of each test case contains n space-separated integers, $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 100$).

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

Output

For each test case, output a single integer on a line, the minimum number of operations needed to make a contain only zeros.

input

```
10
4
0 1 2 3
6
0 0 0 0 0 0
5
1 0 1 0 1
5
3 1 4 1 5
4
3 2 1 0
7
9 100 0 89 12 2 3
4
0 3 9 0
7
0 7 0 2 0 7 0
1
0
2
0 1
```

output

```
1
0
2
1
1
2
1
2
0
1
```

In the first test case, Evirir can choose the subarray $b = [1, 2, 3]$ and replace it with $\operatorname{mex}(1, 2, 3) = 0$, changing a from $[0, \underline{1, 2, 3}]$ to $[0, 0]$ (where the chosen subarray is underlined). Therefore, the answer is 1.

In the second test case, a already contains only 0s, so no operation is needed.

In the third test case, Evirir can change a as follows: $[1, \underline{0, 1, 0, 1}] \to [\underline{1, 2}] \to [0]$. Here, $\operatorname{mex}(0, 1, 0, 1) = 2$ and $\operatorname{mex}(1, 2) = 0$.

In the fourth test case, Evirir can choose b to be the entire array a, changing a from $[\underline{3, 1, 4, 1, 5}]$ to $[0]$.

B. pspspsps

1 second, 256 megabytes

Cats are attracted to pspspsps, but Evirir, being a dignified dragon, is only attracted to pspspsps with oddly specific requirements...

Given a string $s = s_1 s_2 \ldots s_n$ of length n consisting of characters p, s, and . (dot), determine whether a permutation* p of length n exists, such that for all integers i ($1 \le i \le n$):

- If s_i is p, then $[p_1, p_2, \ldots, p_i]$ forms a permutation (of length i);
- If s_i is s, then $[p_i, p_{i+1}, \ldots, p_n]$ forms a permutation (of length $n - i + 1$);
- If s_i is ., then there is no additional restriction.

*A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 500$), the length of s.

The second line of each test case contains a string s of length n that consists of the characters p, s, and ..

It is guaranteed that the sum of n over all test cases does not exceed 5000.

Output

For each test case, output YES or NO on a line. Output YES if there is such a permutation and NO otherwise.

You can output the answer in any case (upper or lower). For example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as positive responses.

input

```
9
4
s.sp
6
pss..s
5
ppppp
2
sp
4
.sp.
8
psss....
1
.
8
pspspsps
20
....................
```

output

```
YES
NO
YES
YES
NO
NO
YES
NO
YES
```

For the first test case, one permutation that works is $p = [3, 4, 1, 2]$. The restrictions are as follows:

- $s_1 =$ s: $[p_1, p_2, p_3, p_4] = [3, 4, 1, 2]$ forms a permutation.
- $s_2 =$.: No additional restriction.
- $s_3 =$ s: $[p_3, p_4] = [1, 2]$ forms a permutation.
- $s_4 =$ p: $[p_1, p_2, p_3, p_4] = [3, 4, 1, 2]$ forms a permutation.

For the second test case, it can be proven that there is no permutation that satisfies all restrictions.

For the third test case, one permutation that satisfies the constraints is $p = [1, 2, 3, 4, 5]$.

C. MEX Cycle

2 seconds, 256 megabytes

Evirir the dragon has many friends. They have 3 friends! That is one more than the average dragon.

You are given integers n, x, and y. There are n dragons sitting in a circle. The dragons are numbered $1, 2, \ldots, n$. For each i ($1 \le i \le n$), dragon i is friends with dragon $i-1$ and $i+1$, where dragon 0 is defined to be dragon n and dragon $n+1$ is defined to be dragon 1. Additionally, dragons x and y are friends with each other (if they are already friends, this changes nothing). Note that all friendships are mutual.

Output n non-negative integers $a_1, a_2, \ldots, a_n$ such that for each dragon i ($1 \le i \le n$), the following holds:

- Let $f_1, f_2, \ldots, f_k$ be the friends of dragon i. Then $a_i = \operatorname{mex}(a_{f_1}, a_{f_2}, \ldots, a_{f_k})$.*

*The minimum excluded (MEX) of a collection of integers $c_1, c_2, \ldots, c_m$ is defined as the smallest non-negative integer t which does not occur in the collection c.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first and only line of each test case contains three integers n, x, y ($3 \le n \le 2 \cdot 10^5$, $1 \le x < y \le n$).

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 n space-separated non-negative integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) on a line that satisfy the condition in the statement. If there are multiple solutions, print any of them. It can be proven that under the problem constraints, a solution with $0 \le a_i \le 10^9$ always exists.

input

```
7
5 1 3
4 2 4
6 3 5
7 3 6
3 2 3
5 1 5
6 2 5
```

output

```
0 2 1 0 1
1 2 1 0
1 2 0 1 2 0
0 1 2 0 1 0 1
2 0 1
1 0 2 1 0
0 1 2 0 2 1
```

For the first test case:

- $i = 1$: Dragon 1's friends are dragons $2, 3, 5$. $\operatorname{mex}(a_2, a_3, a_5) = \operatorname{mex}(2, 1, 1) = 0 = a_1$, so the condition for dragon 1 is satisfied.
- $i = 2$: Dragon 2's friends are dragons $1, 3$. $\operatorname{mex}(a_1, a_3) = \operatorname{mex}(0, 1) = 2 = a_2$.
- $i = 3$: Dragon 3's friends are dragons $1, 2, 4$. $\operatorname{mex}(a_1, a_2, a_4) = \operatorname{mex}(0, 2, 0) = 1 = a_3$.
- $i = 4$: Dragon 4's friends are dragons $3, 5$. $\operatorname{mex}(a_3, a_5) = \operatorname{mex}(1, 1) = 0 = a_4$.
- $i = 5$: Dragon 5's friends are dragons $1, 4$. $\operatorname{mex}(a_1, a_4) = \operatorname{mex}(0, 0) = 1 = a_5$.

D. Shift + Esc

2.5 seconds, 512 megabytes

After having fun with a certain contraption and getting caught, Evirir the dragon decides to put their magical skills to good use — warping reality to escape fast!

You are given a grid with n rows and m columns of non-negative integers and an integer k. Let (i, j) denote the cell in the i-th row from the top and j-th column from the left ($1 \le i \le n$, $1 \le j \le m$). For every cell (i, j), the integer $a_{i,j}$ is written on the cell (i, j).

You are initially at $(1, 1)$ and want to go to (n, m). You may only move down or right. That is, if you are at (i, j), you can only move to $(i+1, j)$ or $(i, j+1)$ (if the corresponding cell exists).

Before you begin moving, you may do the following operation any number of times:

- Choose an integer i between 1 and n and cyclically shift row i to the left by 1. Formally, simultaneously set $a_{i,j}$ to $a_{i,(j \bmod m)+1}$ for all integers j ($1 \le j \le m$).

Note that you may not do any operation after you start moving.
After moving from $(1, 1)$ to (n, m), let x be the number of operations you have performed before moving, and let y be the sum of the integers written on visited cells (including $(1, 1)$ and (n, m)). Then the *cost* is defined as $kx + y$.

Find the minimum cost to move from $(1, 1)$ to (n, m).

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line contains three space-separated integers n, m, and k ($1 \le n, m \le 200$, $0 \le k \le 10^9$).

Then, n lines follow. The i-th line contains m space-separated integers, $a_{i,1}, a_{i,2}, \ldots, a_{i,m}$ ($0 \le a_{i,j} \le 10^9$).

It is guaranteed that the sum of $n \cdot m$ over all test cases does not exceed $5 \cdot 10^4$.

Output

For each test case, output a single integer, the minimum cost to move from $(1, 1)$ to (n, m).

input

```
5
3 3 100
3 4 9
5 2 4
0 101 101
3 4 1
10 0 0 10
0 0 10 0
10 10 0 10
1 1 3
4
3 2 3
1 2
3 6
5 4
10 10 14
58 49 25 12 89 69 8 49 71 23
45 27 65 59 36 100 73 23 5 84
82 91 54 92 53 15 43 46 11 65
61 69 71 87 67 72 51 42 55 80
1 64 8 54 61 70 47 100 84 50
86 93 43 51 47 35 56 20 33 61
100 59 5 68 15 55 69 8 8 60
33 61 20 79 69 51 23 24 56 28
67 76 3 69 58 79 75 10 65 63
6 64 73 79 17 62 55 53 61 58
```

output

```
113
6
4
13
618
```

In the first test case, the minimum cost of 113 can be achieved as follows:

1. Cyclically shift row 3 once. The grid now becomes
$$\begin{bmatrix} 3 & 4 & 9 \\ 5 & 2 & 4 \\ 101 & 101 & 0 \end{bmatrix}.$$
2. Move as follows: $(1, 1) \to (1, 2) \to (2, 2) \to (2, 3) \to (3, 3)$.

$x = 1$ operation is done before moving. The sum of integers on visited cells is $y = 3 + 4 + 2 + 4 + 0 = 13$. Therefore, the cost is $kx + y = 100 \cdot 1 + 13 = 113$.

In the second test case, one can shift row 1 once, row 2 twice, and row 3 thrice. Then, the grid becomes

$$\begin{bmatrix} 0 & 0 & 10 & 10 \\ 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 \end{bmatrix}.$$

$x = 6$ operations were done before moving, and there is a path of cost $y = 0$. Therefore, the cost is $6 \cdot 1 + 0 = 6$.

E. Broken Queries

2 seconds, 256 megabytes

You, a wizard whose creation was destroyed by a dragon, are determined to hunt it down with a magical AOE tracker. But it seems to be toyed with...

This is an interactive problem.

There is a hidden binary array a of length n (**$\mathbf{n}$ is a power of 2**) and a hidden integer k $(2 \le k \le n - 1)$. The array a contains **exactly one 1** (and all other elements are 0). For two integers l and r ($1 \le l \le r \le n$), define the range sum $s(l, r) = a_l + a_{l+1} + \cdots + a_r$.

You have a magical device that takes ranges and returns range sums, but it returns the opposite result when the range has length at least k. Formally, in one query, you can give it a pair of integers $[l, r]$ where $1 \le l \le r \le n$, and it will return either 0 or 1 according to the following rules:

- If $r - l + 1 < k$, it will return $s(l, r)$.
- If $r - l + 1 \ge k$, it will return $1 - s(l, r)$.

Find k using at most 33 queries.

The device is **not** adaptive. It means that the hidden a and k are fixed before the interaction and will not change during the interaction.

Interaction

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

The first line of each test case contains one positive integer n ($4 \le n \le 2^{30}$) — the length of the hidden array. **It is guaranteed that $\mathbf{n}$ is a power of 2**; that is, $n = 2^m$ for some non-negative integer m.

You can make queries in the following way — print one line of the form "**?** l r" where $1 \le l \le r \le n$. After that, read a single integer: 0 or 1, as described in the statement.

If you want to print the answer k, output "**!** k". Then, the interaction continues with the next test case.

Printing the answer does **not** count towards the number of queries made.

After printing each query do not forget to output the end of line and flush* the output. Otherwise, you will get `Idleness limit exceeded` verdict.

If, at any interaction step, you read -1 instead of valid data, your solution must exit immediately. This means that your solution will receive `Wrong answer` because of an invalid query or any other mistake. Failing to exit can result in an arbitrary verdict because your solution will continue to read from a closed stream.

Hacks

The format of the hacks should be the following: the first line should contain one integer t ($1 \le t \le 100$) — the number of test cases. The description of the test cases should follow.

The first and only line of each test case should contain three integers n, p, and k ($4 \le n \le 2^{30}$, $1 \le p \le n$, $2 \le k \le n - 1$) — the length of the hidden array a, the position of the only 1 in a, and the hidden k. n must be a power of 2.

*To flush, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `sys.stdout.flush()` in Python;
- see the documentation for other languages.

input

```
2
8

0

0

1

0

4

1

0
```

output

```
? 3 5

? 1 8

? 4 8

? 3 8

! 6

? 3 3

? 3 4

! 2
```

In the first test case, $k = 6$ and the 1 in the hidden array is at index 6, so $a = [0, 0, 0, 0, 0, 1, 0, 0]$.

- For the query 3 5, since $5 - 3 + 1 = 3 < k$, the device answers correctly. Since 6 is not contained in the range $[3, 5]$, the device answers 0.
- For the query 1 8, since $8 - 1 + 1 = 8 \geq k$, the device answers 0 incorrectly.
- For the query 4 8, since $8 - 4 + 1 = 5 < k$, the device answers 1 correctly.
- For the query 3 8, since $8 - 3 + 1 = 6 \geq k$, the device answers 0 incorrectly.

The example solution then outputs 6 as the answer, which is correct.
In the second test case, $k = 2$ and the 1 in the hidden array is at index 3, so $a = [0, 0, 1, 0]$.

Note that the example solution may not have enough information to determine k above; this is only an example.

F. MEX OR Mania

4 seconds, 1024 megabytes

An integer sequence $b_1, b_2, \ldots, b_n$ is *good* if $\operatorname{mex}(b_1, b_2, \ldots, b_n) - (b_1 | b_2 | \ldots | b_n) = 1$. Here, $\operatorname{mex(c)}$ denotes the MEX* of the collection c, and $|$ is the bitwise OR operator.

Shohag has an integer sequence $a_1, a_2, \ldots, a_n$. He will perform the following q updates on a:

- i x — increase a_i by x.

After each update, help him find the length of the longest good subarray† of a.

*The minimum excluded (MEX) of a collection of integers $c_1, c_2, \ldots, c_k$ is defined as the smallest non-negative integer y which does not occur in the collection c.

†An array d is a subarray of an array f if d can be obtained from f by the 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 the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains two space-separated integers n and q ($1 \le n, q \le 10^5$).

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le n$).

The next q lines of each test case are of the following form:

- i x ($1 \le i, x \le n$) — it means you should increase a_i by x.

It is guaranteed that the sum of n over all test cases doesn't exceed 10^5 and the sum of q doesn't exceed 10^5.

Output

For each test case, output q lines — on the i-th line output the length of the longest good subarray of a after the i-th update.

input

```
2
6 3
0 0 1 0 1 0
6 1
3 2
6 3
3 1
1 3 1
1 1
```

output

```
6
3
2
0
```

In the first test case, after the first update, the array becomes $[0, 0, 1, 0, 1, 1]$, and here the whole array is good because $\operatorname{mex}([0, 0, 1, 0, 1, 1]) - (0|0|1|0|1|1) = 2 - 1 = 1$.

After the second update, the array becomes $[0, 0, 3, 0, 1, 1]$, and here the subarray $[0, 1, 1]$ has the maximum length among all the good subarrays.

Finally, after the third update, the array becomes $[0, 0, 3, 0, 1, 4]$, and here the subarrays $[0, 0]$ and $[0, 1]$ both have the maximum length among all the good subarrays.