A. Log Chopping

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

There are n logs, the i-th log has a length of a_i meters. Since chopping logs is tiring work, errorgorn and maomao90 have decided to play a game.

errorgorn and maomao90 will take turns chopping the logs with **errorgorn chopping first**. On his turn, the player will pick a log and chop it into 2 pieces. If the length of the chosen log is x, and the lengths of the resulting pieces are y and z, then y and z have to be **positive integers**, and $x = y + z$ must hold. For example, you can chop a log of length 3 into logs of lengths 2 and 1, but not into logs of lengths 3 and 0, 2 and 2, or 1.5 and 1.5.

The player who is unable to make a chop will be the loser. Assuming that both errorgorn and maomao90 play optimally, who will be the winner?

Input

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

The first line of each test case contains a single integer n $(1 \leq n \leq 50)$ — the number of logs.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 50)$ — the lengths of the logs.

Note that there is no bound on the sum of n over all test cases.

Output

For each test case, print "`errorgorn`" if errorgorn wins or "`maomao90`" if maomao90 wins. (Output without quotes).

Example

input

```
2
4
2 4 2 1
1
1
```

output

```
errorgorn
maomao90
```

Note

In the first test case, errorgorn will be the winner. An optimal move is to chop the log of length 4 into 2 logs of length 2. After this there will only be 4 logs of length 2 and 1 log of length 1.

After this, the only move any player can do is to chop any log of length 2 into 2 logs of length 1. After 4 moves, it will be maomao90's turn and he will not be able to make a move. Therefore errorgorn will be the winner.

In the second test case, errorgorn will not be able to make a move on his first turn and will immediately lose, making maomao90 the winner.

B. I love AAAB

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

Let's call a string **good** if its length is at least 2 and all of its characters are $\texttt{A}$ except for the last character which is $\texttt{B}$. The good strings are $\texttt{AB}, \texttt{AAB}, \texttt{AAAB}, \ldots$. Note that $\texttt{B}$ is **not** a good string.

You are given an initially empty string s_1.

You can perform the following operation any number of times:

- Choose any position of s_1 and insert some good string in that position.

Given a string s_2, can we turn s_1 into s_2 after some number of operations?

Input

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

The first line of each test case contains a single string s_2 ($1 \le |s_2| \le 2 \cdot 10^5$).

It is guaranteed that s_2 consists of only the characters A and B.

It is guaranteed that the sum of $|s_2|$ over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print "YES" (without quotes) if we can turn s_1 into s_2 after some number of operations, 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
AABAB
ABB
AAAAAAAAB
A
```

output

```
YES
NO
YES
NO
```

Note

In the first test case, we transform s_1 as such: $\varnothing \to$ AAB $\to$ AABAB.

In the third test case, we transform s_1 as such: $\varnothing \to$ AAAAAAAAB.

In the second and fourth test case, it can be shown that it is impossible to turn s_1 into s_2.

C. Unequal Array

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. We define the **equality** of the array as the number of indices $1 \le i \le n-1$ such that $a_i = a_{i+1}$. We are allowed to do the following operation:

- Select two integers i and x such that $1 \le i \le n-1$ and $1 \le x \le 10^9$. Then, set a_i and a_{i+1} to be equal to x.

Find the minimum number of operations needed such that the equality of the array is less than or equal to 1.

Input

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

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

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

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

Output

For each test case, print the minimum number of operations needed.

Example

input

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

output

```
2
1
```

```
2
0
```

Note

In the first test case, we can select $i = 2$ and $x = 2$ to form $[1, 2, 2, 1, 1]$. Then, we can select $i = 3$ and $x = 3$ to form $[1, 2, 3, 3, 1]$.

In the second test case, we can select $i = 3$ and $x = 100$ to form $[2, 1, 100, 100, 2]$.

D. Cyclic Rotation

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 any number of times:

- Choose two indices l and r where $1 \le l < r \le n$ and $a_l = a_r$. Then, set $a[l \dots r] = [a_{l+1}, a_{l+2}, \dots, a_r, a_l]$.

You are also given another array b of length n which is a permutation of a. Determine whether it is possible to transform array a into an array b using the above operation some number of times.

Input

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

The first line of each test case contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the length of array a and b.

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

The third line of each test case contains n integers $b_1, b_2, \dots, b_n$ ($1 \le b_i \le n$) — 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 $2 \cdot 10^5$

Output

For each test case, print "YES" (without quotes) if it is possible to transform array a to b, 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

```
5
5
1 2 3 3 2
1 3 3 2 2
5
1 2 4 2 1
4 2 2 1 1
5
2 4 5 5 2
2 2 4 5 5
3
1 2 3
1 2 3
3
1 1 2
2 1 1
```

output

```
YES
YES
NO
YES
NO
```

Note

In the first test case, we can choose $l = 2$ and $r = 5$ to form $[1, 3, 3, 2, 2]$.

In the second test case, we can choose $l = 2$ and $r = 4$ to form $[1, 4, 2, 2, 1]$. Then, we can choose $l = 1$ and $r = 5$ to form $[4, 2, 2, 1, 1]$.

In the third test case, it can be proven that it is not possible to transform array a to b using the operation.

E. notepad.exe

time limit per test: 1 second
memory limit per test: 256 megabytes

input: standard input
output: standard output

This is an interactive problem.

There are n words in a text editor. The i-th word has length l_i ($1 \le l_i \le 2000$). The array l is hidden and only known by the grader.

The text editor displays words in lines, splitting each two words in a line with at least one space. Note that a line does not have to end with a space. Let the height of the text editor refer to the number of lines used. For the given **width**, the text editor will display words in such a way that the height is minimized.

More formally, suppose that the text editor has **width** w. Let a be an array of length $k+1$ where $1 = a_1 < a_2 < \ldots < a_{k+1} = n+1$. a is a **valid** array if for all $1 \le i \le k$, $l_{a_i} + 1 + l_{a_i+1} + 1 + \ldots + 1 + l_{a_{i+1}-1} \le w$. Then the **height** of the text editor is the minimum k over all valid arrays.

Note that if $w < \max(l_i)$, the text editor cannot display all the words properly and will crash, and the height of the text editor will be 0 instead.

You can ask $n + 30$ queries. In one query, you provide a width w. Then, the grader will return the height h_w of the text editor when its width is w.

Find the minimum area of the text editor, which is the minimum value of $w \cdot h_w$ over all w for which $h_w \neq 0$.

The lengths are fixed in advance. In other words, **the interactor is not adaptive**.

Input

The first and only line of input contains a single integer n ($1 \le n \le 2000$) — the number of words on the text editor.

It is guaranteed that the hidden lengths l_i satisfy $1 \le l_i \le 2000$.

Interaction

Begin the interaction by reading n.

To make a query, print "? w" (without quotes, $1 \le w \le 10^9$). Then you should read our response from standard input, that is, h_w.

If your program has made an invalid query or has run out of tries, the interactor will terminate immediately and your program will get a verdict `Wrong answer`.

To give the final answer, print "! $area$" (without the quotes). Note that giving this answer is not counted towards the limit of $n + 30$ queries.

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

The first line of input must contain a single integer n ($1 \le n \le 2000$) — the number of words in the text editor.

The second line of input must contain exactly n space-separated integers $l_1, l_2, \ldots, l_n$ ($1 \le l_i \le 2000$).

Example

input
6
0
4
2

output
? 1
? 9
? 16
! 32

Note

In the first test case, the words are $\{\texttt{glory}, \texttt{to}, \texttt{ukraine}, \texttt{and}, \texttt{anton}, \texttt{trygub}\}$, so $l = \{5, 2, 7, 3, 5, 6\}$.

If $w = 1$, then the text editor is not able to display all words properly and will crash. The height of the text editor is $h_1 = 0$, so the grader will return 0.

If $w = 9$, then a possible way that the words will be displayed on the text editor is:

- `glory__to`
- `ukraine__`
- `and_anton`
- `__trygub_`

The height of the text editor is $h_9 = 4$, so the grader will return 4.

If $w = 16$, then a possible way that the words will be displayed on the text editor is:

- `glory_to_ukraine`
- `and_anton_trygub`

The height of the text editor is $h_{16} = 2$, so the grader will return 2.

We have somehow figured out that the minimum area of the text editor is 32, so we answer it.

F1. Array Shuffling

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

oolimry has an array a of length n which he really likes. Today, you have changed his array to b, a permutation of a, to make him sad.

Because oolimry is only a duck, he can only perform the following operation to restore his array:

- Choose two integers i, j such that $1 \leq i, j \leq n$.
- Swap b_i and b_j.

The **sadness** of the array b is the minimum number of operations needed to transform b into a.

Given the array a, find any array b which is a permutation of a that has the maximum sadness over all permutations of the array a.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \leq t \leq 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \leq n \leq 2 \cdot 10^5$) — the length of the array.

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

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

Output

For each test case, print n integers $b_1, b_2, \ldots, b_n$ — describing the array b. If there are multiple answers, you may print any.

Example

input

```
2
2
2 1
4
1 2 3 3
```

output

```
1 2
3 3 2 1
```

Note

In the first test case, the array $[1, 2]$ has sadness 1. We can transform $[1, 2]$ into $[2, 1]$ using one operation with $(i, j) = (1, 2)$.

In the second test case, the array $[3, 3, 2, 1]$ has sadness 2. We can transform $[3, 3, 2, 1]$ into $[1, 2, 3, 3]$ with two operations with $(i, j) = (1, 4)$ and $(i, j) = (2, 3)$ respectively.

F2. Checker for Array Shuffling

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

oolimry has an array a of length n which he really likes. Today, you have changed his array to b, a permutation of a, to make him sad.

Because oolimry is only a duck, he can only perform the following operation to restore his array:

- Choose two integers i, j such that $1 \le i, j \le n$.
- Swap b_i and b_j.

The **sadness** of the array b is the minimum number of operations needed to transform b into a.

Given the arrays a and b, where b is a permutation of a, determine if b has the maximum sadness over all permutations of a.

Input

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

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the array.

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

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le n$) — the 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 $2 \cdot 10^5$.

Output

For each test case, print "AC" (without quotes) if b has the maximum sadness over all permutations of a, and "WA" (without quotes) otherwise.

Example

input

```
4
2
2 1
1 2
4
1 2 3 3
3 3 2 1
2
2 1
2 1
4
1 2 3 3
3 2 3 1
```

output

```
AC
AC
WA
WA
```

Note

In the first test case, the array $[1, 2]$ has sadness 1. We can transform $[1, 2]$ into $[2, 1]$ using one operation with $(i, j) = (1, 2)$.

In the second test case, the array $[3, 3, 2, 1]$ has sadness 2. We can transform $[3, 3, 2, 1]$ into $[1, 2, 3, 3]$ with two operations with $(i, j) = (1, 4)$ and $(i, j) = (2, 3)$ respectively.

In the third test case, the array $[2, 1]$ has sadness 0.

In the fourth test case, the array $[3, 2, 3, 1]$ has sadness 1.

G. Cross Xor

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

There is a grid with r rows and c columns, where the square on the i-th row and j-th column has an integer $a_{i,j}$ written on it. Initially, all elements are set to 0. We are allowed to do the following operation:

- Choose indices $1 \le i \le r$ and $1 \le j \le c$, then replace all values on the same row or column as (i, j) with the value xor 1. In other words, for all $a_{x,y}$ where $x = i$ or $y = j$ or both, replace $a_{x,y}$ with $a_{x,y}$ xor 1.

You want to form grid b by doing the above operations a finite number of times. However, some elements of b are missing and are replaced with '?' instead.

Let k be the number of '?' characters. Among all the 2^k ways of filling up the grid b by replacing each '?' with '0' or '1', count the number of grids, that can be formed by doing the above operation a finite number of times, starting from the grid filled with 0. As this number can be large, output it modulo 998244353.

Input

The first line contains two integers r and c ($1 \le r, c \le 2000$) — the number of rows and columns of the grid respectively.

The i-th of the next r lines contain c characters $b_{i,1}, b_{i,2}, \ldots, b_{i,c}$ ($b_{i,j} \in \{0, 1, ?\}$).

Output

Print a single integer representing the number of ways to fill up grid b modulo 998244353.

Examples

input
```
3 3
?10
1??
010
```
output
```
1
```

input
```
2 3
000
001
```
output
```
0
```

input
```
1 1
?
```
output
```
2
```

input
```
6 9
1101011?0
001101?00
101000110
001011010
0101?01??
00?1000?0
```
output
```
8
```

Note

In the first test case, the only way to fill in the **?**s is to fill it in as such:

0	1	0
1	1	1
0	1	0

This can be accomplished by doing a single operation by choosing $(i, j) = (2, 2)$.

In the second test case, it can be shown that there is no sequence of operations that can produce that grid.

H. Zigu Zagu

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

You have a binary string a of length n consisting only of digits 0 and 1.

You are given q queries. In the i-th query, you are given two indices l and r such that $1 \le l \le r \le n$.

Let $s = a[l, r]$. You are allowed to do the following operation on s:

1. Choose two indices x and y such that $1 \le x \le y \le |s|$. Let t be the substring $t = s[x, y]$. Then for all $1 \le i \le |t| - 1$, the condition $t_i \neq t_{i+1}$ has to hold. Note that $x = y$ is always a valid substring.
2. Delete the substring $s[x, y]$ from s.

For each of the q queries, find the minimum number of operations needed to make s an empty string.

Note that for a string s, $s[l, r]$ denotes the subsegment $s_l, s_{l+1}, \ldots, s_r$.

Input

The first line contains two integers n and q ($1 \le n, q \le 2 \cdot 10^5$) — the length of the binary string a and the number of queries respectively.

The second line contains a binary string a of length n ($a_i \in \{0, 1\}$).

Each of the next q lines contains two integers l and r ($1 \le l \le r \le n$) — representing the substring of each query.

Output

Print q lines, the i-th line representing the minimum number of operations needed for the i-th query.

Examples

input

```
5 3
11011
2 4
1 5
3 5
```

output

```
1
3
2
```

input

```
10 3
1001110110
1 10
2 5
5 10
```

output

```
4
2
3
```

Note

In the first test case,

1. The substring is 101, so we can do one operation to make the substring empty.
2. The substring is 11011, so we can do one operation on $s[2, 4]$ to make 11, then use two more operations to make the substring empty.
3. The substring is 011, so we can do one operation on $s[1, 2]$ to make 1, then use one more operation to make the substring empty.

I. PermutationForces

time limit per test: 5 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You have a permutation p of integers from 1 to n.

You have a strength of s and will perform the following operation some times:

- Choose an index i such that $1 \le i \le |p|$ and $|i - p_i| \le s$.
- For all j such that $1 \le j \le |p|$ and $p_i < p_j$, update p_j to $p_j - 1$.
- Delete the i-th element from p. Formally, update p to $[p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n]$.

It can be shown that no matter what i you have chosen, p will be a permutation of integers from 1 to $|p|$ after all operations.

You want to be able to transform p into the empty permutation. Find the minimum strength s that will allow you to do so.

Input

The first line of input contains a single integer n ($1 \le n \le 5 \cdot 10^5$) — the length of the permutation p.

The second line of input conatains n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$) — the elements of the permutation p.

It is guaranteed that all elements in p are distinct.

Output

Print the minimum strength s required.

Examples

input

```
3
3 2 1
```

output

```
1
```

input

```
1
1
```

output

```
0
```

input

```
10
1 8 4 3 7 10 6 5 9 2
```

output

```
1
```

Note

In the first test case, the minimum s required is 1.

Here is how we can transform p into the empty permutation with $s = 1$:

- In the first move, you can only choose $i = 2$ as choosing any other value of i will result in $|i - p_i| \leq s$ being false. With $i = 2$, p will be changed to $[2, 1]$.
- In the second move, you choose $i = 1$, then p will be changed to $[1]$.
- In the third move, you choose $i = 1$, then p will be changed to $[\,]$.

It can be shown that with $s = 0$, it is impossible to transform p into the empty permutation.