Codeforces Round #771 (Div. 2)

A. Reverse

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

You are given a permutation $p_1, p_2, \ldots, p_n$ of length n. You have to choose two integers l, r ($1 \le l \le r \le n$) and reverse the subsegment $[l, r]$ of the permutation. The permutation will become $p_1, p_2, \ldots, p_{l-1}, p_r, p_{r-1}, \ldots, p_l, p_{r+1}, p_{r+2}, \ldots, p_n$.

Find the lexicographically smallest permutation that can be obtained by performing **exactly** one reverse operation on the initial permutation.

Note that for two distinct permutations of equal length a and b, a is lexicographically smaller than b if at the first position they differ, a has the smaller element.

A permutation 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 a single integer t ($1 \le t \le 500$) — the number of test cases. 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 the permutation.

The second line of each test case contains n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$) — the elements of the permutation.

Output

For each test case print the lexicographically smallest permutation you can obtain.

Example

input
4 1 1 3 2 1 3 4 1 4 2 3 5 1 2 3 4 5
output
1 1 2 3 1 2 4 3 1 2 3 4 5

Note

In the first test case, the permutation has length 1, so the only possible segment is $[1, 1]$. The resulting permutation is $[1]$.

In the second test case, we can obtain the identity permutation by reversing the segment $[1, 2]$. The resulting permutation is $[1, 2, 3]$.

In the third test case, the best possible segment is $[2, 3]$. The resulting permutation is $[1, 2, 4, 3]$.

In the fourth test case, there is no lexicographically smaller permutation, so we can leave it unchanged by choosing the segment $[1, 1]$. The resulting permutation is $[1, 2, 3, 4, 5]$.

B. Odd Swap Sort

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

You are given an array $a_1, a_2, \ldots, a_n$. You can perform operations on the array. In each operation you can choose an integer i ($1 \le i < n$), and swap elements a_i and a_{i+1} of the array, if $a_i + a_{i+1}$ is odd.

Determine whether it can be sorted in non-decreasing order using this operation any number of times.

Input
Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. Description of the test cases follows.

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

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the 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 "Yes" or "No" depending on whether you can or can not sort the given array.

You may print each letter in any case (for example, "YES", "Yes", "yes", "yEs" will all be recognized as positive answer).

Example

input
4 4 1 6 31 14 2 4 2 5 2 9 6 7 10 3 6 6 6
output
Yes No No Yes

Note
In the first test case, we can simply swap 31 and 14 ($31 + 14 = 45$ which is odd) and obtain the non-decreasing array $[1, 6, 14, 31]$.

In the second test case, the only way we could sort the array is by swapping 4 and 2, but this is impossible, since their sum $4 + 2 = 6$ is even.

In the third test case, there is no way to make the array non-decreasing.

In the fourth test case, the array is already non-decreasing.

C. Inversion Graph

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

You are given a permutation $p_1, p_2, \ldots, p_n$. Then, an undirected graph is constructed in the following way: add an edge between vertices i, j such that $i < j$ if and only if $p_i > p_j$. Your task is to count the number of connected components in this graph.

Two vertices u and v belong to the same connected component if and only if there is at least one path along edges connecting u and v.

A permutation 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 a single integer t ($1 \le t \le 10^5$) — the number of test cases. Description of the test cases follows.

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

The second line of each test case contains n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$) — the elements of the permutation.

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 one integer k — the number of connected components.

Example

input
6

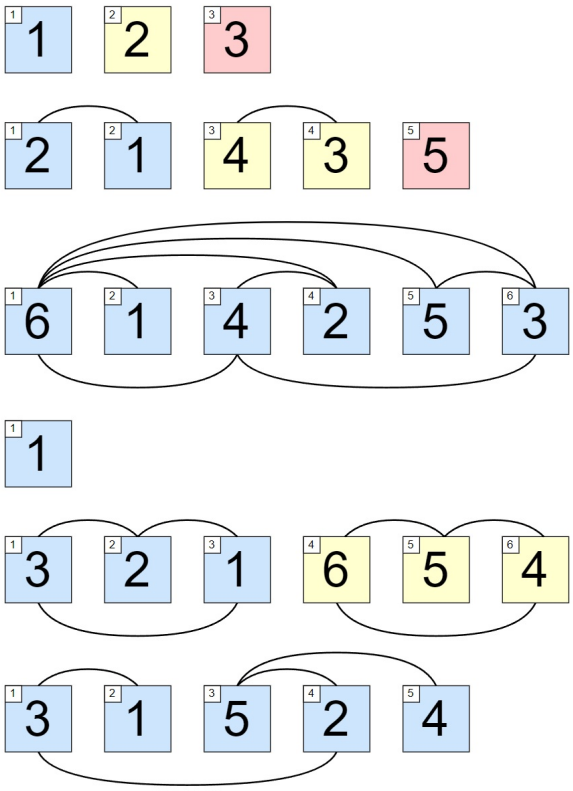
```
3
1 2 3
5
2 1 4 3 5
6
6 1 4 2 5 3
1
1
6
3 2 1 6 5 4
5
3 1 5 2 4
```

output

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

Note

Each separate test case is depicted in the image below. The colored squares represent the elements of the permutation. For one permutation, each color represents some connected component. The number of distinct colors is the answer.



D. Big Brush

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

You found a painting on a canvas of size $n \times m$. The canvas can be represented as a grid with n rows and m columns. Each cell has some color. Cell (i, j) has color $c_{i,j}$.

Near the painting you also found a brush in the shape of a 2×2 square, so the canvas was surely painted in the following way: initially, no cell was painted. Then, the following painting operation has been performed some number of times:

- Choose two integers i and j ($1 \leq i < n$, $1 \leq j < m$) and some color k ($1 \leq k \leq nm$).
- Paint cells (i, j), $(i + 1, j)$, $(i, j + 1)$, $(i + 1, j + 1)$ in color k.

All cells must be painted at least once. A cell can be painted multiple times. In this case, its final color will be the last one.

Find any sequence of at most nm operations that could have led to the painting you found or state that it's impossible.

Input

The first line of input contains two integers n and m ($2 \leq n, m \leq 1000$) — the dimensions of the canvas.

On the i-th of the next n lines of input, there will be m integers. The j-th of them is $a_{i,j}$ ($1 \leq a_{i,j} \leq nm$) — the color of cell (i, j).

Output

If there is no solution, print a single integer -1.

Otherwise, on the first line, print one integer q ($1 \le q \le nm$) — the number of operations.

Next, print the operations in order. On the k-th of the next q lines, print three integers i, j, c ($1 \le i < n$, $1 \le j < m$, $1 \le c \le nm$) — the description of the k-th operation.

If there are multiple solutions, print any.

Examples

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

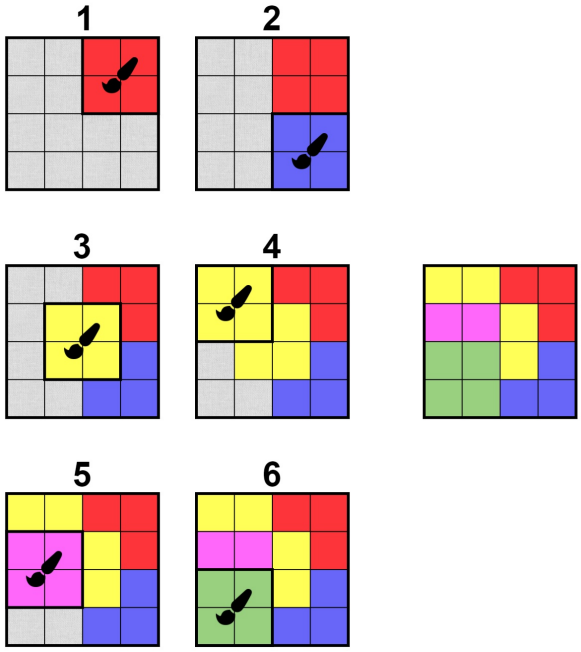
output
6 1 3 3 3 3 4 2 2 5 1 1 5 2 1 1 3 1 2

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

output
-1

Note

In the first test case, the solution is not unique. Here's one of them:



In the second test case, there is no way one could obtain the given painting, thus the answer is -1.

E. Colorful Operations

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have an array $a_1, a_2, \ldots, a_n$. Each element initially has value 0 and color 1. You are also given q queries to perform:

- `Color` l r c: Change the color of elements $a_l, a_{l+1}, \cdots, a_r$ to c ($1 \le l \le r \le n$, $1 \le c \le n$).
- `Add` c x: Add x to values of all elements a_i ($1 \le i \le n$) of color c ($1 \le c \le n$, $-10^9 \le x \le 10^9$).
- `Query` i: Print a_i ($1 \le i \le n$).

Input

The first line of input contains two integers n and q ($1 \leq n, q \leq 10^6$) — the length of array a and the number of queries you have to perform.

Each of the next q lines contains the query given in the form described in the problem statement.

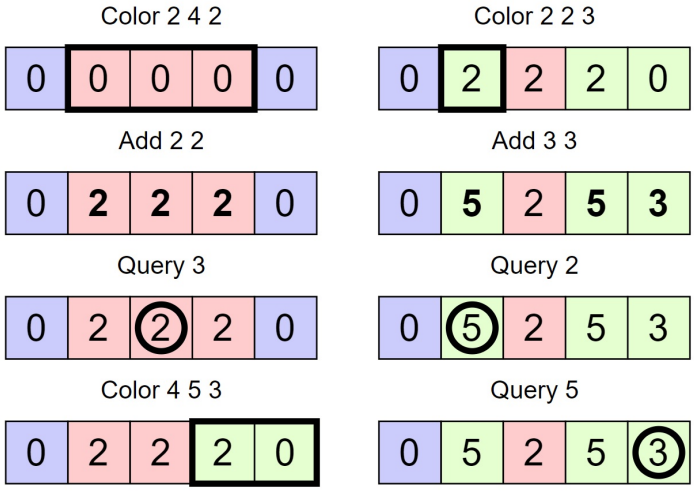
Output
Print the answers to the queries of the third type on separate lines.

Examples

input
5 8 Color 2 4 2 Add 2 2 Query 3 Color 4 5 3 Color 2 2 3 Add 3 3 Query 2 Query 5
output
2 5 3

input
2 7 Add 1 7 Query 1 Add 2 4 Query 2 Color 1 1 1 Add 1 1 Query 2
output
7 7 8

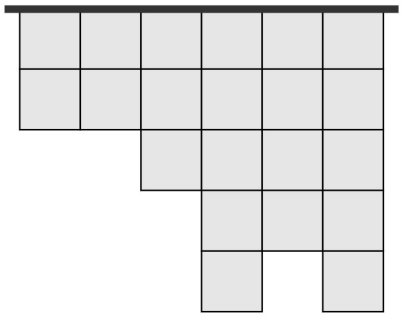
Note
The first sample test is explained below. Blue, red and green represent colors 1, 2 and 3 respectively.



F. Two Posters

time limit per test: 2 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You want to advertise your new business, so you are going to place two posters on a billboard in the city center. The billboard consists of n vertical panels of width 1 and varying integer heights, held together by a horizontal bar. The i-th of the n panels has height h_i.



Initially, all panels hang down from the bar (their top edges lie on it), but before placing the two posters, you are allowed to move each panel up by any integer length, as long as it is still connected to the bar (its bottom edge lies below or on it).

After the moves are done, you will place two posters: one below the bar and one above it. They are not allowed to go over the bar and they must be positioned completely inside of the panels.

What is the maximum total area the two posters can cover together if you make the optimal moves? **Note that you can also place a poster of 0 area. This case is equivalent to placing a single poster.**

Input

The first line of input contains one integer n ($1 \le n \le 10^4$) — the number of vertical panels.

The second line of input contains n integers $h_1, h_2, \ldots, h_n$ ($1 \le h_i \le 10^{12}$) — the heights of the n vertical panels.

Output

Print a single integer — the maximum total area the two posters can cover together.

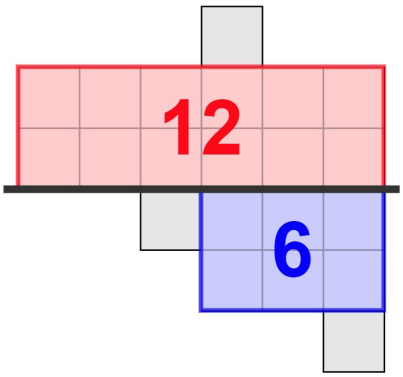
Examples

input
6 2 2 3 5 4 5
output
18

input
1 1
output
1

Note

In the first sample test, we can choose an upper poster with area 12 and a lower poster of area 6 as in the image below.



In the second sample test, we can cover the whole billboard using a single poster.