Sponsored by Telegram

Codeforces Round #648 (Div. 2)

A. Matrix Game

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

Ashish and Vivek play a game on a matrix consisting of rows and columns, where they take turns claiming cells. Unclaimed cells are represented by , while claimed cells are represented by . The initial state of the matrix is given. There can be some claimed cells in the initial state.

In each turn, a player must claim a cell. A cell may be claimed if it is unclaimed and does not share a row or column with any other already claimed cells. When a player is unable to make a move, he loses and the game ends.

If Ashish and Vivek take turns to move and Ashish goes first, determine the winner of the game if both of them are playing optimally.

Optimal play between two players means that both players choose the best possible strategy to achieve the best possible outcome for themselves.

Input

The first line consists of a single integer — the number of test cases. The description of the test cases follows.

The first line of each test case consists of two space-separated integers , — the number of rows and columns in the matrix.

The following lines consist of integers each, the -th integer on the -th line denoting .

Output

For each test case if Ashish wins the game print "`Ashish`" otherwise print "`Vivek`" (without quotes).

Example

input

```
4
2 2
0 0
0 0
2 2
0 0
0 1
2 3
1 0 1
1 1 0
3 3
1 0 0
0 0 0
1 0 0
```

output

```
Vivek
Ashish
Vivek
Ashish
```

Note

For the first case: One possible scenario could be: Ashish claims cell , Vivek then claims cell . Ashish can neither claim cell , nor cell as cells and are already claimed. Thus Ashish loses. It can be shown that no matter what Ashish plays in this case, Vivek will win.

For the second case: Ashish claims cell , the only cell that can be claimed in the first move. After that Vivek has no moves left.

For the third case: Ashish cannot make a move, so Vivek wins.

For the fourth case: If Ashish claims cell , Vivek will have no moves left.

B. Trouble Sort

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

Ashish has elements arranged in a line.

These elements are represented by two integers — the value of the element and — the type of the element (there are only two possible types: and). He wants to sort the elements in non-decreasing values of .

He can perform the following operation any number of times:

- Select any two elements and such that and swap them. That is, he can only swap two elements of different types in one move.

Tell him if he can sort the elements in non-decreasing values of after performing any number of operations.

Input

The first line contains one integer — the number of test cases. The description of the test cases follows.

The first line of each test case contains one integer — the size of the arrays.

The second line contains integers — the value of the -th element.

The third line containts integers — the type of the -th element.

Output

For each test case, print "Yes" or "No" (without quotes) depending on whether it is possible to sort elements in non-decreasing order of their value.

You may print each letter in any case (upper or lower).

Example

input

```
5
4
10 20 20 30
0 1 0 1
3
3 1 2
0 1 1
4
2 2 4 8
1 1 1 1
3
5 15 4
0 0 0
4
20 10 100 50
1 0 0 1
```

output

```
Yes
Yes
Yes
No
Yes
```

Note

For the first case: The elements are already in sorted order.

For the second case: Ashish may first swap elements at positions and , then swap elements at positions and .

For the third case: The elements are already in sorted order.

For the fourth case: No swap operations may be performed as there is no pair of elements and such that . The elements cannot be sorted.

For the fifth case: Ashish may swap elements at positions and , then elements at positions and .

C. Rotation Matching

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

After the mysterious disappearance of Ashish, his two favourite disciples Ishika and Hriday, were each left with one half of a secret message. These messages can each be represented by a permutation of size . Let's call them and .

Note that a permutation of elements is a sequence of numbers , in which every number from to appears exactly once.

The message can be decoded by an arrangement of sequence and , such that the number of matching pairs of elements between them is maximum. A pair of elements and is said to match if:

- , that is, they are at the same index.
-

His two disciples are allowed to perform the following operation any number of times:

- choose a number and cyclically shift one of the permutations to the left or right times.

A single cyclic shift to the left on any permutation is an operation that sets simultaneously. Likewise, a single cyclic shift to the right on any permutation is an operation that sets simultaneously.

Help Ishika and Hriday find the maximum number of pairs of elements that match after performing the operation any (possibly zero) number of times.

Input

The first line of the input contains a single integer — the size of the arrays.

The second line contains integers , , ..., — the elements of the first permutation.

The third line contains integers , , ..., — the elements of the second permutation.

Output

Print the maximum number of matching pairs of elements after performing the above operations some (possibly zero) times.

Examples

input

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

output

```
5
```

input

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

output

```
1
```

input

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

output

```
2
```

Note

For the first case: can be shifted to the right by . The resulting permutations will be and .

For the second case: The operation is not required. For all possible rotations of and , the number of matching pairs won't exceed .

For the third case: can be shifted to the left by . The resulting permutations will be and . Positions and have matching pairs of elements. For all possible rotations of and , the number of matching pairs won't exceed .

D. Solve The Maze

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

Vivek has encountered a problem. He has a maze that can be represented as an grid. Each of the grid cells may represent the following:

- Empty — '.'
- Wall — '#'
- Good person — 'G'
- Bad person — 'B'

The only escape from the maze is at cell .

A person can move to a cell only if it shares a side with their current cell and does not contain a wall. Vivek wants to block some of the empty cells by replacing them with walls in such a way, that all the good people are able to escape, while none of the bad people are able to. A cell that initially contains 'G' or 'B' **cannot be blocked** and **can be travelled through**.

Help him determine if there exists a way to replace some (zero or more) empty cells with walls to satisfy the above conditions.

It is guaranteed that the cell is empty. Vivek can also block this cell.

Input

The first line contains one integer — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers , — the number of rows and columns in the maze.

Each of the next lines contain characters. They describe the layout of the maze. If a character on a line equals '`.`', the corresponding cell is empty. If it equals '#', the cell has a wall. 'G' corresponds to a good person and 'B' corresponds to a bad person.

Output

For each test case, print "`Yes`" if there exists a way to replace some empty cells with walls to satisfy the given conditions. Otherwise print "No"

You may print every letter in any case (upper or lower).

Example

input

```
6
1 1
.
1 2
G.
2 2
#B
G.
2 3
G.#
B#.
3 3
#B.
#..
GG.
2 2
#B
B.
```

output

```
Yes
Yes
No
No
Yes
Yes
```

Note

For the first and second test cases, all conditions are already satisfied.

For the third test case, there is only one empty cell , and if it is replaced with a wall then the good person at will not be able to escape.

For the fourth test case, the good person at cannot escape.

For the fifth test case, Vivek can block the cells and .

For the last test case, Vivek can block the destination cell .

E. Maximum Subsequence Value

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ridhiman challenged Ashish to find the maximum valued subsequence of an array of size consisting of positive integers.

The value of a non-empty subsequence of elements of is defined as over all integers such that at least elements of the subsequence have the -th bit set in their binary representation (value has the -th bit set in its binary representation if — is equal to).

Recall that is a subsequence of , if can be obtained by deleting some(possibly zero) elements from .

Help Ashish find the maximum value he can get by choosing some subsequence of .

Input

The first line of the input consists of a single integer — the size of .

The next line consists of space-separated integers — the elements of the array .

Output

Print a single integer — the maximum value Ashish can get by choosing some subsequence of .

Examples

input

```
3
2 1 3
```

output

```
3
```

input

```
3
3 1 4
```

output

```
7
```

input

```
1
1
```

output

```
1
```

input

```
4
7 7 1 1
```

output

```
7
```

Note

For the first test case, Ashish can pick the subsequence of size . The binary representation of is `10` and that of is `11`. Since is equal to , the value of the subsequence is (both and have -st bit set in their binary representation and has -th bit set in its binary representation). Note that he could also pick the subsequence or .

For the second test case, Ashish can pick the subsequence with value .

For the third test case, Ashish can pick the subsequence with value .

For the fourth test case, Ashish can pick the subsequence with value .

F. Swaps Again

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ayush, Ashish and Vivek are busy preparing a new problem for the next Codeforces round and need help checking if their test cases are valid.

Each test case consists of an integer and two arrays and , of size . If after some (possibly zero) operations described below, array can be transformed into array , the input is said to be `valid`. Otherwise, it is `invalid`.

An operation on array is:

- select an integer −
- swap the prefix of length with the suffix of length

For example, if array initially is , after performing an operation with , it is transformed into .

Given the set of test cases, help them determine if each one is `valid` or `invalid`.

Input

The first line contains one integer — the number of test cases. The description of each test case is as follows.

The first line of each test case contains a single integer — the size of the arrays.

The second line of each test case contains integers , , ..., — elements of array .

The third line of each test case contains integers , , ..., — elements of array .

Output

For each test case, print "`Yes`" if the given input is `valid`. Otherwise print "`No`".

You may print the answer in any case.

Example

input

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

output

```
yes
yes
No
yes
No
```

Note

For the first test case, we can swap prefix with suffix to get .

For the second test case, is already equal to .

For the third test case, it is impossible since we cannot obtain in .

For the fourth test case, we can first swap prefix with suffix to obtain . Now we can swap prefix with suffix to obtain .

For the fifth test case, it is impossible to convert to .

G. Secure Password

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem.

Ayush devised yet another scheme to set the password of his lock. The lock has slots where each slot can hold any non-negative integer. The password is a sequence of integers, -th element of which goes into the -th slot of the lock.

To set the password, Ayush comes up with an array of integers each in the range . He then sets the -th element of as the bitwise OR of all integers in the array except .

You need to guess the password. To make a query, you can choose a non-empty subset of indices of the array and ask the **bitwise OR** all elements of the array with index in this subset. **You can ask no more than 13 queries**.

Input

The first line of input contains one integer — the number of slots in the lock.

Interaction

To ask a query print a single line:

- In the beginning print "`? c` " (without quotes) where denotes the size of the subset of indices being queried, followed by **distinct** space-separated integers in the range .

For each query, you will receive an integer — the bitwise OR of values in the array among all the indices queried. If the subset of indices queried is invalid or you exceeded the number of queries then you will get . In this case, you should terminate the program immediately.

When you have guessed the password, print a single line "`!` " (without quotes), followed by space-separated integers — the password sequence.

Guessing the password does **not** count towards the number of queries asked.

The interactor is not adaptive. The array does not change with queries.

After printing a query do not forget to output the end of the 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 the documentation for other languages.

Hacks

To hack the solution, use the following test format:

On the first line print a single integer — the number of slots in the lock. The next line should contain space-separated integers in the range — the array .

Example

input

```
3
1
2
4
```

output

```
? 1 1
? 1 2
? 1 3
! 6 5 3
```

Note

The array in the example is . The first element of the password is bitwise OR of and , the second element is bitwise OR of and and the third element is bitwise OR of and . Hence the password sequence is .