Sorting Permutation Unit

Problem

You may have heard of Google's Tensor Processing Units, which are used to build neural networks. However, there is one research area that is even deeper and more important than machine learning: sorting!

We are working on a special new chip called the Sorting Permutation Unit, which is very fast at applying permutations to arrays of integers. Formally, a permutation is an ordering of the first n positive integers

$p_1, p_2, ..., p_n$

and applying it to an array of n integers

$a_1, a_2, ..., a_n$

yields the new array

$a_{p_1}, a_{p_2}, ..., a_{p_n}$.

For example, applying the permutation 3, 1, 2, 4 to the array 99, 234, 45, 800 would yield the new array 45, 99, 234, 800.

However, permutations are expensive to represent in the hardware, so the unit can only have access to at most **P** distinct permutations. We need your help figuring out what those permutations should be!

Given **K** arrays of **N** integers each, you must first specify up to **P** permutations (of size **N**) of your choice. Then, for each of those **K** input arrays, you must provide one sequence of up to **S** instructions (each of which is a permutation from your specified set). When the instructions in this sequence are applied, in the given order, to the array, they must yield an array sorted in nondecreasing order. In each of your **K** sequences of instructions, you may use each of your **P** permutations zero or more times (not necessarily consecutively).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with four integers **P**, **S**, **K**, and **N**: the maximum number of permutations allowed, the maximum number of instructions you are allowed to use to sort each array, the number of arrays, and the number of integers in each array. Then, there are **K** more lines of **N** integers A_{i1}, A_{i2}, .., A_{iN} each, where the j-th integer on the i-th line, A_{ij}, represents the j-th value of the i-th array.

Output

For each test case, first output the following, in this order:

- One line containing `Case #x:`, where `x` is the test case number (starting from 1).
- One line containing one integer P', where 1 ≤ P' ≤ **P**: the number of permutations you have chosen to use.

- P' lines, the i-th of which contains **N** integers p_{i1} p_{i2} ... p_{iN}, where `pij` is the j-th element of the i-th permutation.

Then, output **K** more lines. The i-th of these gives the instructions that you will apply to the i-th array given in the input. Each such line must begin with one integer S', where $0 \le S' \le S$, and must continue with S' integers X_1, X_2, ..., $X_{S'}$, where $1 \le X_k \le P'$ for all k. Here, X_k represents that the k-th instruction you apply to the i-th array is the X_k-th permutation (counting starting from 1) in your list of permutations. These instructions must yield an array with the elements of the i-th input array, sorted in nondecreasing order.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.
$1 \le$ **T** ≤ 10.
S $= 450$.
$1 \le$ **K** ≤ 30.
$2 \le$ **N** ≤ 50.
$1 \le$ **A$_{ij}$** ≤ 1000, for all i and j.

Test set 1 (Visible)

P = 20.

Test set 2 (Hidden)

P = 5.

Sample

Sample Input	Sample Output
2 20 450 4 3 10 10 11 17 4 1000 999 998 997 10 10 11 20 450 5 5 1 2 3 4 5 2 3 4 5 1 3 4 5 1 2 4 5 1 2 3 5 1 2 3 4	Case #1: 2 3 1 2 2 1 3 0 1 2 2 2 1 1 2 Case #2: 1 5 1 2 3 4 0 1 1 2 1 1 3 1 1 1 4 1 1 1 1

In Sample Case #1, we can define up to **P** = 20 permutations. One viable strategy uses only these two:

1. 3 1 2

2. 2 1 3

We can handle the four arrays as follows:

- `10 10 11`: This is already sorted in nondecreasing order, so we do not need to do anything.
- `17 4 1000`: We can apply permutation #2 to yield 4 17 1000.
- `999 998 997`: One option is to first apply permutation #2 to turn the array into 998 999 997, then apply permutation #1 to turn the array into 997 998 999.
- `10 10 11`: This is the same as the first array. Applying permutation #2 also yields array sorted in nondecreasing order. (But we could have used the line `0` as we did before.)

In Sample Case #2, notice that we can use the same permutation instruction more than once on the same array, if desired.