BHCSI Algorithms Program: Stones

You love stones. They come in all shapes and sizes. They are especially fun to throw at your younger siblings. Recently, you discovered a huge stone quarry with more stones than you thought even existed. Naturally, you've been spending a lot of time at the quarry, and have started inventing several new games to help you pass the time. Your favorite game is one where you find a row of stones and try to find the longest sequence of (not necessarily contiguous) stones which are strictly increasing in size (for the purpose of this problem, the *size* of a stone is a single positive integer). Of course, you are not allowed to change the order of stones, or the game would be trivial. The game is pretty easy when there are only a few stones, but as the number of stones grows, the game becomes harder for you to play. You decide to write a computer program to help you play optimally.

The Problem

Given the number of stones in a row and the size of each stone, find the longest sequence of stones which are strictly increasing in size. Because there are several rows of stones in the quarry, your program should be able to handle multiple rows of stones.

Input Format

Input will begin with a single positive integer T indicating the number of test cases. This will be followed by exactly T test cases. Each test case will consist of exactly 2 lines. The first line will contain only a positive integer $N \leq 100$ indicating the number of stones in the row. The following line will contain exactly N positive integers, all ≤ 100, giving the size of each stone in the row.

Output Format

For each row of stones in the input file, you should output a single line of the following format:

`Row #`*`x`*`:` *`s`*

where x is the test case number starting at 1, and s is the maximum number of stones you can choose satisfying the rules of the game.

[*Sample Input/Output on next page*]

Sample Input

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

Sample Output

```
Row #1: 4
Row #2: 1
Row #3: 3
Row #4: 3
Row #5: 3
Row #6: 2
```

Sample Output Explanations

Row #1: You can use all the stones
Row #2: No matter which stone you start with, you can only use one stone
Row #3: Remember that the stones must be in *strictly increasing* order of size
Row #4: You can get 3 stones by choosing (1,4,5), (1,2,5), or (1,2,3)
Row #5: Even though you could choose 8, you're better off skipping it to choose (2,6,7)
Row #6: Be careful! It's not always optimal to choose the first stone in the row. The best is (1,2)

Grading Details

Your assignment will be graded upon the three following criteria: correctness, use of dynamic programming, and programming style.