Code Jam 2018 - Qualification Round

Trouble Sort

Problem

Deep in Code Jam's secret algorithm labs, we devote countless hours to wrestling with one of the most complex problems of our time: efficiently sorting a list of integers into non-decreasing order. We have taken a careful look at the classic bubble sort algorithm, and we are pleased to announce a new variant.

The basic operation of the standard bubble sort algorithm is to examine a pair of adjacent numbers, and reverse that pair if the left number is larger than the right number. But our algorithm examines a group of *three* adjacent numbers, and if the leftmost number is larger than the rightmost number, it reverses that entire group. Because our algorithm is a "triplet bubble sort", we have named it Trouble Sort for short.

```
TroubleSort(L): // L is a 0-indexed list of integers
  let done := false
  while not done:
    done = true
    for i := 0; i < len(L)-2; i++:
      if L[i] > L[i+2]:
        done = false
        reverse the sublist from L[i] to L[i+2], inclusive
```

For example, for L = 5 6 6 4 3, Trouble Sort would proceed as follows:

- First pass:
  - inspect 5 6 6, do nothing: 5 6 6 4 3
  - inspect 6 6 4, see that 6 > 4, reverse the triplet: 5 4 6 6 3
  - inspect 6 6 3, see that 6 > 3, reverse the triplet: 5 4 3 6 6
- Second pass:
  - inspect 5 4 3, see that 5 > 3, reverse the triplet: 3 4 5 6 6
  - inspect 4 5 6, do nothing: 3 4 5 6 6
  - inspect 5 6 6, do nothing: 3 4 5 6 6
- Then the third pass inspects the three triplets and does nothing, so the algorithm terminates.

We were looking forward to presenting Trouble Sort at the Special Interest Group in Sorting conference in Hawaii, but one of our interns has just pointed out a problem: it is possible that Trouble Sort does not correctly sort the list! Consider the list 8 9 7, for example.

We need your help with some further research. Given a list of **N** integers, determine whether Trouble Sort will successfully sort the list into non-decreasing order. If it will not, find the index (counting starting from 0) of the first sorting error after the algorithm has finished: that is, the first value that is larger than the value that comes directly after it when the algorithm is done.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of two lines: one line with an integer **N**, the number of values in the list, and then another line with **N** integers $\mathbf{V_i}$, the list of values.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is `OK` if Trouble Sort correctly sorts the list, or the index (counting starting from 0) of the first sorting error, as described above.

Limits

$1 \le \mathbf{T} \le 100$.
$0 \le \mathbf{V_i} \le 10^9$, for all i.
Memory limit: 1GB.

Test set 1 (Visible)

$3 \le \mathbf{N} \le 100$.
Time limit (for the entire test set): 10 seconds.

Test set 2 (Hidden)

$3 \le \mathbf{N} \le 10^5$.
Time limit (for the entire test set): 20 seconds.

Special Note

Notice that test set 2 for this problem has a large amount of input, so using a non-buffered reader might lead to slower input reading. In addition, keep in mind that certain languages have a small input buffer size by default.

Sample

Sample Input

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

Sample Output

```
Case #1: OK
Case #2: 1
```

Sample Case #1 is similar to the first one described in the problem statement. Trouble Sort correctly sorts this list, so the answer is `OK`.

Sample Case #2 is the second one described in the problem statement. Trouble Sort does not correctly sort this list, since it terminates with the list 7 9 8. The 9 is the first value in the list that is larger than the next value, so the index of the first sorting error is 1.