Bonus Problem #1:
String to Palindrome

In this problem you are asked to convert a string into a palindrome with minimum number of operations. The operations are described below:

Here you'd have the ultimate freedom. You are allowed to:

- Add any character at any position
- Remove any character from any position
- Replace any character at any position with another character

Every operation you do on the string would count for a unit cost. You'd have to keep that as low as possible.

For example, to convert "abccda" you would need at least two operations if we allowed you only to add characters. But when you have the option to replace any character you can do it with only one operation. We hope you'd be able to use this feature to your advantage.

Input:

The input file contains several test cases. The first line of the input gives you the number of test cases, T ($1 \leq$ T ≤ 10). Then T test cases will follow, each in one line. The input for each test case consists of a string containing lower case letters only. You can safely assume that the length of this string will not exceed 1000 characters.

Output:

For each set of input print the test case number first. Then print the minimum number of characters needed to turn the given string into a palindrome.

Sample Input:

```
6
tanbirahmed
shahriarmanzoor
monirulhasan
syedmonowarhossain
sadrulhabibchowdhury
mohammadsajjadhossain
```

Sample Output:

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

Bonus Problem #2:
History Grading

Many problems in Computer Science involve maximizing some measure according to constraints.

Consider a history exam in which students are asked to put several historical events into chronological order. Students who order all the events correctly will receive full credit, but how should partial credit be awarded to students who incorrectly rank one or more of the historical events?

Some possibilities for partial credit include:

1 point for each event whose rank matches its correct rank
1 point for each event in the longest (not necessarily contiguous) sequence of events which are in the correct order relative to each other.

For example, if four events are correctly ordered 1 2 3 4 then the order 1 3 2 4 would receive a score of 2 using the first method (events 1 and 4 are correctly ranked) and a score of 3 using the second method (event sequences 1 2 4 and 1 3 4 are both in the correct order relative to each other).

In this problem you are asked to write a program to score such questions using the second method.

Given the correct chronological order of n events 1, 2, . . ., n as c1, c2, . . . cn where $1 \leq$ ci $\leq$ n denotes the ranking of event i in the correct chronological order and a sequence of student responses r1, r2, . . . rn where $1 \leq$ ri $\leq$ n denotes the chronological rank given by the student to event i; determine the length of the longest (not necessarily contiguous) sequence of events in the student responses that are in the correct chronological order relative to each other.

Input:

The first line of the input will consist of one integer n indicating the number of events with $2 \leq n \leq 20$. The second line will contain n integers, indicating the correct chronological order of n events. The remaining lines will each consist of n integers with each line representing a student's chronological ordering of the n events. All lines will contain n numbers in the range [1...n], with each number appearing exactly once per line, and with each number separated from other numbers on the same line by one or more spaces.

Output:

For each student ranking of events your program should print the score for that ranking. There should be one line of output for each student ranking.

Sample Input 1:

```
4
4 2 3 1
```

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

Sample Output 1:

```
1
2
3
```

Sample Input 2:

```
10
3 1 2 4 9 5 10 6 8 7
1 2 3 4 5 6 7 8 9 10
4 7 2 3 10 6 9 1 5 8
3 1 2 4 9 5 10 6 8 7
2 10 1 3 8 4 9 5 7 6
```

Sample Output 2:

```
6
5
10
9
```

Bonus Problem #3:
Alignment

In the army, a platoon is composed by n soldiers. During the morning inspection, the soldiers are aligned in a straight line in front of the captain. The captain is not satisfied with the way his soldiers are aligned; it is true that the soldiers are aligned in order by their code number: 1 , 2 , 3 , . . . , n , but they are not aligned by their height. The captain asks some soldiers to get out of the line, as the soldiers that remain in the line, without changing their places, but getting closer, to form a new line, where each soldier can see by looking lengthwise the line at least one of the line's extremity (left or right). A soldier see an extremity if there isn't any soldiers with a higher or equal height than his height between him and that extremity.

Write a program that, knowing the height of each soldier, determines the minimum number of soldiers which have to get out of line.

Input:

On the first line of the input is written the number of the soldiers n. On the second line is written a series of n floating numbers with at most 5 digits precision and separated by a space character. The k-th number from this line represents the height of the soldier who has the code k (1 <= k <= n).

There are some restrictions:
$2 \leq n \leq 1000$
The height are floating numbers from the interval [0.5, 2.5]

Output:

Print a single integer the number of distinct good substrings of string s.

Sample Input:

```
8
1.86 1.86 1.30621 2 1.4 1 1.97 2.2
```

Sample Output:

```
4
```