Thanks to MikeMirzayanov

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

... for beautiful Codeforces and Polygon platforms. And thanks to XTX Markets for supporting Codeforces Global Rounds. And also big thanks to **dario2994** who was an author of Codeforces Global Round 11 and who has permitted to reuse his problem `https://codeforces.com/contest/1427/problem/D`. Yep, that's exactly the same problem apart from that doing it in n operations is kinda boring, don't you agree?

You are given a deck of n cards numbered from 1 to n (not necessarily in this order in the deck). You have to sort the deck by repeating the following operation.

Choose $2 \le k \le n$ and split the deck in k nonempty contiguous parts $D_1, D_2, \ldots, D_k$ (D_1 contains the first $|D_1|$ cards of the deck, D_2 contains the following $|D_2|$ cards and so on). Then reverse the order of the parts, transforming the deck into $D_k, D_{k-1}, \ldots, D_2, D_1$ (so, the first $|D_k|$ cards of the new deck are D_k, the following $|D_{k-1}|$ cards are D_{k-1} and so on). The internal order of each packet of cards D_i is unchanged by the operation.

You have to obtain a sorted deck (that is, a deck where the first card is 1, the second is 2 and so on) performing at most 120 operations. It can be proven that it is always possible to sort the deck performing at most 120 operations under the limitations of this problem.

Examples of operation: the following are three examples of valid operations (on three decks with different sizes).

- If the deck is `[3 6 2 1 4 5 7]` (so 3 is the first card and 7 is the last card), we may apply the operation with $k = 4$ and D_1 =`[3 6]`, D_2 =`[2 1 4]`, D_3 =`[5]`, D_4 =`[7]`. Doing so, the deck becomes `[7 5 2 1 4 3 6]`.
- If the deck is `[3 1 2]`, we may apply the operation with $k = 3$ and D_1 =`[3]`, D_2 =`[1]`, D_3 =`[2]`. Doing so, the deck becomes `[2 1 3]`.
- If the deck is `[5 1 2 4 3 6]`, we may apply the operation with $k = 2$ and D_1 =`[5 1]`, D_2 =`[2 4 3 6]`. Doing so, the deck becomes `[2 4 3 6 5 1]`.

Input

The first line of the input contains one integer n ($1 \le n \le 20\,000$) — the number of cards in the deck.

The second line contains n integers $c_1, c_2, \ldots, c_n$ — the cards in the deck. The first card is c_1, the second is c_2 and so on.

It is guaranteed that for all $i = 1, \ldots, n$ there is exactly one $j \in \{1, \ldots, n\}$ such that $c_j = i$.

Output

On the first line, print the number q of operations you perform (it must hold that $0 \le q \le 120$).

Then, print q lines, each describing one operation.

To describe an operation, print on a single line the number k of parts you are going to split the deck in, followed by the sizes of the k parts: $|D_1|, |D_2|, \ldots, |D_k|$.

It must hold that $2 \le k \le n$, and $|D_i| \ge 1$ for all $i = 1, \ldots, k$, and $|D_1| + |D_2| + \ldots + |D_k| = n$.

It can be proven that it is always possible to sort the deck performing at most 120 operations under the limitations of this problem. If there are several ways to sort the deck you can output any one of them. Note that you don't have to minimize q.

Examples

standard input	standard output
4 3 1 2 4	2 3 1 2 1 2 1 3
6 6 5 4 3 2 1	1 6 1 1 1 1 1 1
1 1	0