# Ogre Sort

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 2 seconds |
| Memory limit: | 512 megabytes |

What happens if you cast an ogre curse on a stick of RAM? It turns out that it can circlularly move arbitrary sequences of data in an efficient manner, but only at night.

Find the minimum cost (and a valid sequence of moves) needed to sort a permutation $v$ of length $n$. All elements of the permutation are indexed from 1 to $n$.

The only permitted type of move allows you to take an element from some position $x$ and insert it at another position $y$, shifting all elements in between by one. The cost of such a move is $y$.

Formally, a move takes an element valued $t$ from position $x$, "freeing" the index $x$. We then shift the remaining elements in $v$, such that the "free" position becomes $y$. We then put $t$ in the free position at index $y$.

For example, if we have a permutation [4, 3, 2, 1], some of the possible moves:

- $x = 2$, $y = 4$, the resulting permutation is [4, 2, 1, 3], the cost of the move is 4.
- $x = 2$, $y = 1$, the resulting permutation is [3, 4, 2, 1], the cost of the move is 1.

## Input

The first line contains an integer $n$ — the length of the permutation.

The second line contains $n$ integers $v_1, v_2, \ldots, v_n$ — the values of the permutation.

## Constraints

$1 \leq n \leq 3 \cdot 10^5$,

$1 \leq v_i \leq n$,

$v_i \neq v_j$ for all $1 \leq i < j \leq n$.

## Output

On the first line, print two numbers *min_cost* and *len_moves* — the minimum cost needed to sort the permutation and the length of the proposed sequence of moves respectively.

The next *len_moves* lines should each contain two integers $x_k$, $y_k$ each, signifying that the $k$-th operation should move the element from position $x_k$ to position $y_k$ ($1 \leq k \leq len\_moves$, $1 \leq x_k, y_k \leq n$).

If several possible sequences of moves exist, you can print any of them. Note that you don't need to minimize the number of moves, only the total cost. It's guaranteed that there always exist a sequence of moves to sort a permutation.

## Examples

| standard input | standard output |
|---|---|
| 4<br>1 2 4 3 | 3 1<br>4 3 |
| 5<br>2 4 1 3 5 | 3 2<br>4 2<br>4 1 |
| 3<br>1 2 3 | 0 0 |

## Note

- For the first example, the only move effectively swaps the elements on positions 3 and 4.
- For the second example, the permutation resulting after the first move is $[2, 3, 4, 1, 5]$.
- For the third example, the permutation is already sorted, so the optimal cost is zero, and no moves are needed.