E: Sorting Algorithm

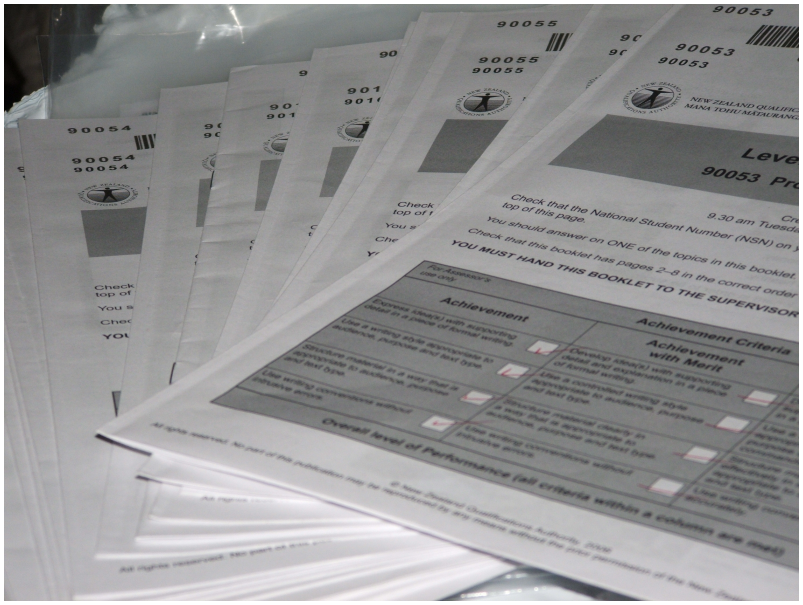
Polimi's researchers and PhD students are developing a sorting algorithm for exam papers that should make the grading process much faster for the professors. There is a long sequence of N papers to be graded, each of those being related to a specific subject. There are S total subjects.

The algorithm works like this: one paper is removed from the *start* of the original sequence, and is moved to the *end* of a new (initially empty) sequence. The next step is the sorting step: the paper will be repeatedly swapped with the adjacent paper, moving it towards the *start* of the new sequence, until either:

- The paper reaches the beginning of the new sequence, or
- The adjacent paper is related to the same subject, so the two papers will just get stacked on top of each other, ending the sorting step.

This process is repeated until no papers are left in the original sequence.

You can see how this novel method might speed things up, since it will produce at most S stacks of papers, one for each subject, all ready to be handed over to the professor!



In order to estimate the computational complexity of the algorithm, the researchers want to analyze how many swaps are needed. Given the initial sequence of unsorted papers, compute the **total number of swaps** that will be executed by the algorithm.

Input

The first line contains two space-separated integers N and S, respectively: the number of papers and the number of subjects. The second line contains N integers s_i, each of them used to indicate a subject.

Output

You need to write a single line with an integer: the total number of swaps required by the algorithm.

Constraints

- $1 \leq N \leq 100\,000$.
- $1 \leq S \leq 1\,000\,000\,000$.
- $0 \leq s_i < S$ for each $i = 0 \ldots N-1$.
- Once two papers are stacked together, they become like one single paper for all practical purposes (i.e. a stack of papers can be swapped as if it was one single paper).

Scoring

Your program will be tested against several testcases, and will be considered **correct** only if it will solve all of them correctly.

Examples

input	output
5 3 2 2 1 0 1	4
17 2 1 1 0 0 1 0 1 0 0 1 1 1 0 1 1 0 1	7

In the **first example**, the algorithm requires: 0 swaps for the first paper, 0 swaps for the second one (because it's related to the same subject), then 1 swap for the third one, 2 swaps for the fourth, and only 1 swap for the fifth.