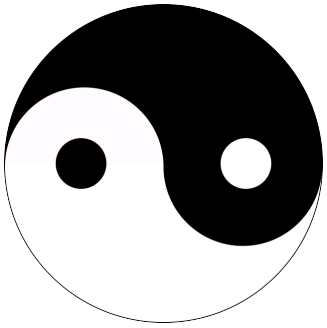
Array Partition (partition)



Giorgio is working on a research paper which involves the **partition** step of the Quicksort algorithm. The partition step works like this: we choose some element of the array (it doesn't matter which one), which will then be called *pivot* element. Then we move all the array elements in such a way that, in the end:

- all the elements on the left of the pivot are smaller than it;
- all the elements on the right of the pivot are greater than it.

For example, let's take this array: $2, 6, 1, 5, 7, 8, 9, 4, 3$. Let's say we choose the second element as pivot (the one with value 6). On the left, there is only one element with value 2, which is smaller than 6, so the "left partitioning" is already satisfied. On the right, however, there are some elements which are *not* grater than 6, so we should move them in order to obtain a correct partition.

So, a valid partition (among many) of the array is this: $2, 3, 4, 1, 5, 6, 9, 7, 8$.

For the purpose of Giorgio's research, we need to know the number of *out-of-place elements* for each possible pivot. In the example above, with pivot 6, there are 0 out-of-place elements on the left and 4 on the right ($1, 5, 4$ and 3), for a total of 4 out-of-place elements. If the pivot 5 was chosen, there would be $1 + 2 = 3$ out-of-place elements. Help Giorgio compute the number of out-of-place elements for each possible pivot.

> ☞ Among the attachments of this task you may find a template file `partition.*` with a sample incomplete implementation.

Input

The first line contains the only integer N, the size of Giorgio's array. The second line contains N distinct integers V_i, the elements of the array.

Output

You need to write a single line with N space-separated integers, the i-th of which represents the number of out-of-place elements when the i-th pivot is chosen.

Constraints

- $1 \leq N \leq 100\,000$.
- $1 \leq V_i \leq N$ for each $i = 0 \ldots N - 1$.
- All array elements are distinct.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**35 points**]: $N \leq 10$.
- **Subtask 3** [**50 points**]: $N \leq 1000$.
- **Subtask 4** [**10 points**]: No additional limitations.

Examples

input.txt	output.txt
9 2 6 1 5 7 8 9 4 3	1 4 2 3 2 2 2 6 6
3 3 2 1	2 2 2

Explanation

The **first sample case** is the example which was previously described.

In the **second sample case**, "every other element" is out-of-place, regardless of which pivot is chosen.