D. Serega and Fun

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Serega loves fun. However, everyone has fun in the unique manner. Serega has fun by solving query problems. One day Fedor came up with such a problem.

You are given an array a consisting of n positive integers and queries to it. The queries can be of two types:

1. Make a unit cyclic shift to the right on the segment from l to r (both borders inclusive). That is rearrange elements of the array in the following manner:
$$a[l], a[l+1], ..., a[r-1], a[r] \rightarrow a[r], a[l], a[l+1], ..., a[r-1].$$
2. Count how many numbers equal to k are on the segment from l to r (both borders inclusive).

Fedor hurried to see Serega enjoy the problem and Serega solved it really quickly. Let's see, can you solve it?

Input

The first line contains integer n $(1 \le n \le 10^5)$ — the number of elements of the array. The second line contains n integers $a[1], a[2], ..., a[n]$ $(1 \le a[i] \le n)$.

The third line contains a single integer q $(1 \le q \le 10^5)$ — the number of queries. The next q lines contain the queries.

As you need to respond to the queries online, the queries will be **encoded**. A query of the first type will be given in format: $1\ l'_i\ r'_i$. A query of the second type will be given in format: $2\ l'_i\ r'_i\ k'_i$. All the number in input are integer. They satisfy the constraints: $1 \le l'_i, r'_i, k'_i \le n$.

To decode the queries from the data given in input, you need to perform the following transformations:

$$l_i = ((l'_i + lastans - 1)\ mod\ n) + 1;\ r_i = ((r'_i + lastans - 1)\ mod\ n) + 1;\ k_i = ((k'_i + lastans - 1)\ mod\ n) + 1.$$

Where $lastans$ is the last reply to the query of the 2-nd type (initially, $lastans = 0$). If after transformation l_i is greater than r_i, you must swap these values.

Output

For each query of the 2-nd type print the answer on a single line.

Examples

input

```
7
6 6 2 7 4 2 5
7
1 3 6
2 2 4 2
2 2 4 7
2 2 2 5
1 2 6
1 1 4
2 1 7 3
```

output

```
2
1
0
0
```

input

```
8
8 4 2 2 7 7 8 8
8
1 8 8
2 8 1 7
1 8 1
1 7 3
2 8 8 3
1 1 4
1 2 7
1 4 5
```

output

```
2
0
```