Problem Cool Rotation

C header	`coolrot.h`
C++ header	`coolrot.h`

Little Square has decided to buy some cake for Little Triangle, from the InfO(1) bakery. This cake is very strangely constructed: it is formed out of N slices, each with a certain number of chocolates on top of it. The number of chocolates on each slice is a distinct integer from 0 to $N-1$. Little Triangle's taste in cake is rather *eccentric*: the degree to which they like a cake is determined by the number of (possibly non-adjacent) pairs of slices where the left slice has more chocolates than the right slice. Thus she will like a cake with the fewest number of such pairs of slices more than one with more such pairs. In case they are presented with two cakes that are the same with respect to this criterion, they prefer the one that is smaller in lexicographical order.

Little Square wants Little Triangle to like the cake they get for them as much as possible. To do this they can use some operations that the bakery can do. The InfO(1) bakery is very understanding, and does not care about the number of operations. Each operation is determined by two integers, d and x, where d is a divisor of N, and $0 < x < \frac{N}{d}$. The operation consists of splitting the cake into blocks of d slices, and moving the first x blocks to the end of the cake. Some examples of operations follow:

$$[0,1,2,3,4,5] \xrightarrow[d=2]{x=1} [2,3,4,5,0,1]$$
$$[0,1,2,3,4,5] \xrightarrow[d=3]{x=1} [3,4,5,0,1,2]$$
$$[0,1,2,3,4,5] \xrightarrow[d=1]{x=3} [3,4,5,0,1,2]$$

Unfortunately, depending on the day, only certain types operations are permitted; each day only certain values for d are allowed. Can you help Little Square perform operations so as to get the best possible cake according to Little Triangle's criteria, on each of the Q days in question?

Interaction protocol

The contestant must implement two functions, `init` and `query`, with the following signature.

```
void init(int n, const int a[], int q);
```

```
void query(int m, const int ds[]);
```

The contestant is allowed to use the following function.

```
void update(int d, int x);
```

The `init` function will be called exactly once, before any of the calls to `query`. The function will be supplied with `n`, the length of the cake, `a`, an array containing the number of chocolates on each slice (indexed from 0), and `q`, the number of days we are interested in.

The `query` function, called precisely `q` times, is supplied with `m`, the number of d values that are allowed in that day, and `ds`, a sorted array (indexed from 0) containing `m` distinct divisors of n that represents the allowed d values in that day. Using the `update` function (which corresponds to an operation done by the InfO(1) bakery), the cake slices must be put into the optimal arrangement during the call to `query`.

All calls to `query` are independent. Thus operations done during one call to `query` do not carry over to the next.

The sample grader reads, on the first line, N and Q, on the second line the array `a`, and then the description of the Q queries. Each query is formed of two lines. On the first line of a query you find the m value for that query (the number of allowed values of d), and on the second line those values of d.

For each query, the sample grader outputs the operations done using `update`, each on a separate line, and at the end the sequence obtained.

Attention! The contestant should not implement the main function.

Constraints

- $2 \leq N \leq 100.000$
- $1 \leq Q \leq 100.000$
- $1 \leq \sum m \leq 2.000.000$
- $1 \leq ds[i] \leq N$, for every i, where $0 \leq i < m$
- $ds[i]$ is a divisor of N, for every i, where $0 \leq i < m$
- The grader given to the contestant is not necessarily the same with the grader used for scoring
- **For each update the following restriction must be satisfied:** $0 < x < \frac{N}{d}$

Subtask 1 (12 points)

- $N \leq 1.000$
- $Q = 1$
- $m = 1$

Subtask 2 (19 points)

- $m = 1$

Subtask 3 (11 points)

- $m = 2$

Subtask 4 (12 points)

- $m = 3$

Subtask 5 (11 points)

- $Q = 1$

Subtask 6 (35 points)

- No further constraints.

Examples

input	output

6 2
1 3 4 0 2 5
2
3 2
1
2
	3 1
	0 2 5 1 3 4
	1 3 4 0 2 5

Explanation

In the first testcase, it is optimal to do one operation with $d = 3, x = 1$, leading to sequence $0, 2, 5, 1, 3, 4$.
In the second testcase, it is optimal to do no operations, leading to sequence $1, 3, 4, 0, 2, 5$