Colorful Lanterns

Keza is organizing this year's Festival of Lights. Lanterns come in N colors, numbered from 0 to $N-1$, and her warehouse contains $A[i]$ lanterns of color i.

At the festival, Keza will hang all the lanterns one by one on a single long rope, in an order of her choice.

Each time she hangs a lantern, if the rope now holds at least one lantern of **every** color from 0 to $N-1$, the crowd cheers, and they immediately take every lantern on the rope home as a souvenir, except the lantern that was just hung, which remains on the rope. Otherwise, nothing happens and Keza simply continues hanging lanterns.

For example, suppose $N = 3$. The following describes the rope after a possible sequence of hangings:

- $[] \to$ hangs a color 0 lantern $\to [0]$
- $[0] \to$ hangs a color 0 lantern $\to [0, 0]$
- $[0, 0] \to$ hangs a color 2 lantern $\to [0, 0, 2]$
- $[0, 0, 2] \to$ hangs a color 1 lantern $\to [1]$

The **beauty** of the festival is the total number of cheers. Keza would like to know what is the maximum possible total beauty over all possible hanging orders.

While Keza is planning the ordering of the lanterns, suppliers keep revising the warehouse. There will be Q updates, the j-th of which changes the number of lanterns of color $X[j]$ to $V[j]$. For the initial warehouse contents, and after each update, Keza wants to know the maximum possible beauty over all hanging orders.

Implementation Details

You should implement the following procedures:

```
int64 init(int32[] A)
```

- A: an array of length N describing the initial warehouse; there are $A[i]$ lanterns of color i.
- This procedure is called exactly once, before any call to `update`.
- It should return the maximum possible beauty for the initial warehouse contents.

```
int64 update(int32 X, int32 V)
```

- X: the color whose number of lanterns changes.
- V: the new number of lanterns of color X.
- This procedure is called Q times.
- It should return the maximum possible beauty for the current warehouse contents.

Note that even if you solve only the subtasks without updates (with $Q = 0$), your solution should still contain the `update` procedure, but it can be empty.

Constraints

- $2 \leq N \leq 200\,000$
- $0 \leq Q \leq 200\,000$
- $1 \leq A[i] \leq 10^9$ for $0 \leq i < N$
- $0 \leq X[j] \leq N - 1$ for $0 \leq j < Q$
- $1 \leq V[j] \leq 10^9$ for $0 \leq j < Q$

Subtasks

Subtask	Score	Additional Constraints
1	20	$Q = 0$, $N \leq 6$, $A[i] \leq 6$ for $0 \leq i < N$
2	20	$Q = 0$
3	20	$N, Q \leq 2\,000$
4	20	$A[i], V[j] \leq 10^6$ for $0 \leq i < N$ and $0 \leq j < Q$
5	20	No additional constraints.

Examples

Example 1

Consider the following call:

```
init([2, 1, 3])
```

Here $N = 3$: the warehouse has 2 lanterns of color 0, 1 lantern of color 1, and 3 lanterns of color 2.

One optimal order for Keza is the following. She hangs lanterns of colors 2, 0, 1: the rope now holds all three colors, the crowd cheers, and only the last lantern (color 1) stays on the rope. She then hangs 0 and 2: all colors again, a second cheer, and only the color 2 lantern stays. Finally she

hangs the remaining lantern of color 2: nothing happens. The beauty is 2, and no order achieves more, so `init` should return 2.

Then the grader makes the following call:

```
update(1, 2)
```

The warehouse becomes $[2, 2, 3]$, and the maximum possible beauty is now 3, so `update` should return 3.

Then the grader makes the following call:

```
update(0, 1)
```

The warehouse becomes $[1, 2, 3]$, and `update` should return 2.

Sample Grader

Input format:

```
N Q
A[0] A[1] ... A[N-1]
X[0] V[0]
X[1] V[1]
...
X[Q-1] V[Q-1]
```

Output format:

```
I
R[0]
R[1]
...
R[Q-1]
```

Here I denotes the return value of `init`, while R denotes the return values of the calls to `update`, in order.