Superpowertree

Ben just learned about range-queries. He was so fascinated of them that he even wanted to create his own problem involving range-queries. He was thinking about the power function: let pow(a, b) = a^b. But this function was a) too boring and b) not a function over a range. So he just took an array of size n namely a_0, ..., $a_{(n - 1)}$ and defined the superpower of a range:

superpower(l, r) = pow(pow(... pow(pow(a_l, $a_{(l + 1)}$), $a_{(l + 2)}$) ..., $a_{(r - 1)}$), a_r).

Or more formally:

superpower(x, x) = a_x

superpower(l, r) = pow(superpower(l, r - 1), a_r)   if l != r

He then suddenly realised that his problem also needs updates to be a good query-problem. So he decided to add normal updates of the form "change the number at position k to v". But while trying small values for his arrays, he realized that already the results for these small values were getting really big, so he decided he only wants them modulo m. But as Ben is not as expierienced in making problems as in not-listening to his teacher (who is btw. talking about associativity of operations for some data structures right now), he doesn't know that taking prime numbers as modulos is the normal way of doing it. So he just chooses any number m but with the restriction that it's coprime to all the numbers in his array.

Suddenly he realised that the problem invented by himself was too difficult for him to solve as his own solution didn't work. Help Ben!

Input

On the first line, there are three numbers n, q, m: the size of the array, the number of queries and the number for the modulo.

Then on the second line, n numbers follow: a_0, a_1, ..., $a_{(n - 1)}$

Then each each of the next q lines consists of a char and two numbers:

if the char is q, then the two numbers l and r follow: Ben wants to know superpower(l, r) modulo m.

if the char is u, then the two numbers k and v follow: Ben wants to update a_k to v.

Output

For each query of type q, output the superpower of the range modulo m.

Constraints

1 <= n, q <= 1e5

2 <= m <= 1e15

At any time, all the values in the array are <= 1e9 and are coprime to m.

Edit (6.5.2020):

I know that with m <= 1e15, there may be overflows if not handled correctly. In C++ in gcc, there is the __int128 type which is essentially a 128-bit integer. The master solution uses __int128 everywhere, so you can use it without running into TLE.

Example

Input:

```
10 10 77
2 6 3 9 5 4 8 3 4 5
q 0 9
q 3 6
q 2 7
u 1 4
u 4 9
q 0 8
q 4 7
q 4 8
u 5 5
q 2 7
```

Output:

```
1
23
1
36
64
71
1
```