Problem H. Hack

Time limit: 10 seconds

Heidi is analyzing a peculiar device. This device takes an a as input and computes $a^d \pmod n$ using the following pseudocode and some integers d and n stored in this device:

```
1 modPow(a, d, n) {
2   r = 1;
3   for (i = 0; i < 60; ++i) {
4     if ((d & (1 << i)) != 0) {
5       r = r * a % n;
6     }
7     a = a * a % n;
8   }
9 }
```

Note that the pseudocode assumes arbitrary sized integers, `<<` denotes bitwise shift left, `&` denotes bitwise and, and `%` denotes modulo.

The device does *not* tell Heidi the result of the computation. However, Heidi can measure how long does the computation take. She knows that only multiplication modulo n (lines 5 and 7 in the above pseudocode) takes any measurable amount of time, all other lines can be assumed to take 0 nanoseconds. Moreover, she knows that it takes $(\text{bits}(x) + 1) \cdot (\text{bits}(y) + 1)$ nanoseconds to multiply x by y modulo n, where $\text{bits}(x)$ is the number of bits in the binary representation of x without leading zeros, or more formally $\text{bits}(x) = \lceil \log_2 (x + 1) \rceil$.

Heidi knows the integer n but does not know the integer d. She wants to find d by feeding the device different integers a as input and measuring the time the computation takes for each a.

She knows that n and d were chosen in the following way: first, two prime numbers p and q with 30 bits in binary representation (in other words, between 2^{29} and $2^{30} - 1$) were picked independently and uniformly at random. Then the number n was computed as $n = p \cdot q$. Then the number $m = \phi(n) = (p - 1) \cdot (q - 1)$ was computed. Then d was picked uniformly at random between 1 and $m - 1$ inclusive, such that it is coprime with m.

Interaction Protocol

First, the testing system writes the integer n — the modulo used by the device. Note that n and the hidden number d are guaranteed to have been generated according to the procedure described above.

Your solution shall print requests of two types:

- "`?` a" tells to feed a as input to the device. a must be an integer between 0 and $n - 1$ inclusive. The testing system responds with the time it took the device to compute `modPow`(a, d, n) in nanoseconds.
- "`!` d" tells the value of d that your program has determined.

Don't forget to flush the output after each request!

Your solution must issue exactly one request of the second type, which must be the last request, and the solution must terminate gracefully after issuing it.

Your solution is allowed to issue at most 30 000 requests of the first type.

Your solution will be run on 30 testcases, working with one (n, d) pair per run. For each testcase the numbers n and d are fixed and were generated using the procedure described above. The example below was *not* generated in that manner and thus will *not* be used for testing your solution; it only serves to illustrate the input/output format and provide a sanity check for your calculation of the computation time.

Examples

input	output
15 980 293	? 3 ? 8 ! 5

Notes

In the first request in the example case, the following multiplications are done by the device when computing `modPow`$(3, 5, 15)$:

- $1 \cdot 3 \pmod{15} = 3$, taking 6 nanoseconds
- $3 \cdot 3 \pmod{15} = 9$, taking 9 nanoseconds
- $9 \cdot 9 \pmod{15} = 6$, taking 25 nanoseconds
- $3 \cdot 6 \pmod{15} = 3$, taking 12 nanoseconds
- $6 \cdot 6 \pmod{15} = 6$, taking 16 nanoseconds
- $6 \cdot 6 \pmod{15} = 6$, taking 16 nanoseconds
- $6 \cdot 6 \pmod{15} = 6$, taking 16 nanoseconds
- ... (55 more repetitions of the last multiplication)

The computation takes $6 + 9 + 25 + 12 + 58 \cdot 16 = 980$ nanoseconds.

A positive integer is *prime* if it has exactly two divisors: 1 and itself. Two positive integers are *coprime* if their greatest common divisor is 1.

Here are a few first values of the function bits(x):

- bits$(0) = 0$
- bits$(1) = 1$
- bits$(2) = 2$
- bits$(3) = 2$
- bits$(4) = 3$
- ...