

## Bit Recovery (B)

Memory limit: 1024 MB      Time limit: 1.00 s

This task is interactive. After printing each line, you should flush the output buffer. You can use `cout << flush` in C++, `System.out.flush()` in Java, and `sys.stdout.flush()` in Python. You must strictly follow the instructions in the *Interaction* section; otherwise **you may receive verdicts like *wrong answer, time limit exceeded, or others***.

Since all the tasks have mythical dwarves as their main characters, it may be worth finally thinking about something tangible.

There is a hidden sequence  $A$  of length  $N$  consisting of integers in the range  $[0, 2^N - 1]$ . Your task is to recover it using a limited number of queries. In each query, you provide a sequence  $B$  of length  $N$  with elements in the range  $[0, 2^N - 1]$ . The response is computed as follows:

- Sequence  $C$  is created where the  $i$ -th element of  $C$  is the bitwise **xor**<sup>1</sup> of the  $i$ -th elements of  $A$  and  $B$ . We denote **xor** by  $\oplus$ .
- Set  $S$  is constructed as the set of all values obtainable by **xoring** some subset of elements of  $C$ . In particular, for the empty subset, the **xor** is 0.
- The answer to the query is  $|S|$ .

For example, if  $A = (1, 4, 3)$  and  $B = (0, 4, 7)$ , then  $C = (1 \oplus 0, 4 \oplus 4, 3 \oplus 7) = (1, 0, 4)$  and  $S = \{0, 1, 4, 5\}$ , so the answer is 4.

### Interaction

The sequence  $A$  is **fixed** at the start and does not depend on the queries made.

First, read a line containing one integer  $N$ . Then you may ask queries.

To ask a query, print a line containing `?` followed by  $N$  integers in the range  $[0, 2^N - 1]$ . In response, read one integer being the answer.

When ready, print `!` followed by the  $N$  integers of the hidden sequence. Then your program should exit without further output.

Remember to flush after each query and after writing the answer. Put spaces between numbers and the `?`, `!` symbols.

### Limits

$1 \leq N \leq 60$ ,

you may use at most 4000 queries.

<sup>1</sup>The bitwise **xor** of two numbers  $x$  and  $y$  has bit  $i$  set if and only if exactly one of  $x$  and  $y$  has bit  $i$  set. For example,  $5 \oplus 3 = (101)_2 \oplus (011)_2 = (110)_2 = 6$ . In C++ and Python, **xor** is the operator `^`.

## Sample interaction

The first sample test has  $N = 3$  and  $A = (1, 3, 4)$ :

Input	Output	Explanation
3		$N = 3$
4	? 1 2 3	$B = (1, 2, 3)$ $C = (1 \oplus 1, 3 \oplus 2, 4 \oplus 3) = (0, 1, 7)$ $S = \{0, 1, 6, 7\}$ , so the answer is 4
8	? 0 0 0	$B = (0, 0, 0)$ $C = (1 \oplus 0, 3 \oplus 0, 4 \oplus 0) = (1, 3, 4)$ $S = \{0, 1, \dots, 7\}$ , so the answer is 8
	! 1 3 4	The hidden sequence is $A = (1, 3, 4)$

## Local testing

In the *Files* section you can find **B.zip** containing sample tests and a grader. To test your solution, compile it, then pass the test name and your executable to `./grader`:

```
./grader [test] [executable]
```

For example: `./grader 0b.in ./abc`

Before the first run, you may need to make `grader` executable. This can be done using the command:

```
chmod +x grader
```

The sample grader is **not guaranteed** to behave identically to the official one. However, neither is adaptive.