D. Seating Plan (seatingplan)

The closing ceremony of this EGOI will be attended by N important guests. These all need to be seated in the front row in a very specific order that corresponds to all the nuances of the diplomatic protocol. Determining the correct seating order took Noemi two sleepless nights.

Veronica is supervising the closing ceremony. One of her many responsibilities is to make sure that the front-row seats have the correct name signs. There is only one tiny problem: Noemi never told her the correct seating order, and now Noemi is nowhere to be found. Luckily, Dorka the photographer has an app that might be useful.

Dorka had to prepare her cameras so that she could take some specific photos of guests in the first row. For the setup, she needed to know how wide each photo would be, so Noemi made her an app that quickly outputs the information she needs. Veronica now wants to use the app to find the correct seat assignment.

The N important guests are numbered from 0 to $N-1$. The seats in the front row are also numbered from 0 to $N-1$, left to right. For each I ($0 \leq I \leq N-1$), let g_I denote the guest who is supposed to sit in seat I, and let s_I denote the seat in which guest I is supposed to sit.

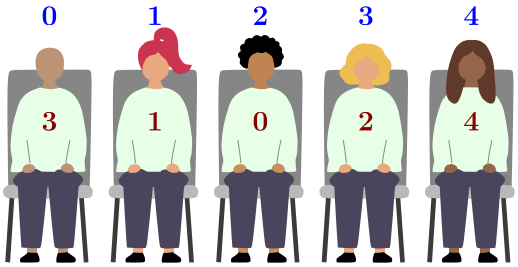

Figure 1: A row with five guests. For this row, $g = [3, 1, 0, 2, 4]$ and $s = [2, 1, 3, 0, 4]$.

The app works as follows:

- Dorka enters the numbers I, J, K of exactly three different guests.
- The app tells her the minimum number of guests that will be visible if all three selected guests are in the photo. Formally, the app will display the value $\max(s_I, s_J, s_K) - \min(s_I, s_J, s_K) + 1$.

For example, see the situation shown in Figure 1:

- Guests $I = 0$, $J = 2$, and $K = 4$ are in seats $s_I = 2$, $s_J = 3$, and $s_K = 4$. If Dorka selects them, the app will display the value $\max(2, 3, 4) - \min(2, 3, 4) + 1 = 3$.

  In other words, the narrowest photo that contains guests 0, 2, and 4 has just these three guests.

- Guests $I = 0$, $J = 4$, and $K = 3$ are in seats $s_I = 2$, $s_J = 4$, and $s_K = 0$. If Dorka selects them, the app will display the value $\max(2, 4, 0) - \min(2, 4, 0) + 1 = 5$.

  In other words, a photo that contains the three given guests must contain all 5 guests.

Help Veronica determine the correct seating order using Dorka's app. More specifically, your program should determine and output the sequence $g_0, g_1, ..., g_{N-1}$. There are always exactly two correct answers (one being the reverse of the other), and you may output either of them. Your score will depend on the number of queries to the app your solution makes.

Implementation

⇨ This is an interactive problem. Your program will use standard input and output to communicate with a grader in the format described below.

Your program should start by reading one line of input containing a positive integer T, the number of test cases that follow.

For each test case, your program should start by reading one line of input containing a positive integer N, the number of seats, which is also the number of guests.

To make a query, your program should output a line of the form "? I J K", where $0 \le I, J, K \le N-1$ are three **distinct** numbers.

After making a query, your program should read one line containing one positive integer, the answer to your query.

To answer with a correct seating order, your program should output a line of the form "! g_0 … g_{N-1}".

After solving all T test cases your program should terminate normally.

Note that the official grader used in CMS to test your solution may be **adaptive**. That is, for some test cases, the permutation of guests is not determined in advance. Instead, the grader may decide which of the remaining permutations it uses depending on the queries already asked by your program.

Flushing. If you are not using the provided templates, make sure to flush standard output after printing each line, or else your program might get judged as *Not correct*. In Python, this happens automatically if you use `input()` to read lines, and you can use `print(..., flush=True)` to force it to flush. In C++, `cout << endl;` flushes in addition to printing a newline; if using printf, use `fflush(stdout)`.

Constraints

- $1 \le T \le 10$.
- N will be 5 (example only), 8, 40, or 2000.
- For each test case, you may make at most 10 000 queries.

Scoring

Your program will be tested on several test cases grouped into subtasks. To obtain the score for a subtask, you must correctly solve all the tests it contains.

- **Subtask 0 [0 points]**: Example ($N = 5$).
- **Subtask 1 [9 points]**: $N = 8$.
- **Subtask 2 [11 points]**: $N = 2000$, and guests 0 and 1 sit next to each other.
- **Subtask 3 [15 points]**: $N = 40$.
- **Subtask 4 [65 points]**: $N = 2000$.

For subtasks 1 and 2 any solution that correctly solves all test cases will be awarded all the points.

For subtasks 3 and 4 your solution must correctly solve all test cases to score any points, and your score will depend on Q_s, the largest number of queries your solution needed to make to solve a test case. Let $X_s = \max(1, Q_s/N)$. The scores for subtasks 3 and 4 are then computed as follows:

$$\text{score}_3 = \min\left(15, 3 + \frac{19}{X_s^{1.5}}\right), \qquad \text{score}_4 = \min\left(65, 3 + \frac{91}{X_s^{1.5}}\right)$$

The value of score_s is rounded to the nearest integer per subtask, and your total score is the sum of these. In order to get a full score, you need to solve subtask 3 in at most 55 queries and subtask 4 in at most 2597 queries. Example values of Q_s and scores for subtasks 3 and 4 are shown below.

Q_s	55	56	60	70	80	100	150	10000
score_3	15	14	13	11	10	8	6	3

Q_s	2597	2800	3000	4000	5000	6000	8000	10000
score_4	65	58	53	35	26	21	14	11

Examples

Grader	Solution
`1` `5`	
	`? 0 2 4`
`3`	
	`? 3 0 1`
`3`	
	`? 0 4 3`
`5`	
	`! 3 1 0 2 4`

Explanation

The example input contains one test case ($T = 1$) with $N = 5$ guests. The hidden configuration of guests in this test case corresponds to Figure 1.

The first query made by the example solution is 0, 2, 4. The answer 3 to this query tells us that these guests sit, in some unknown order, in three consecutive seats next to each other.

The answer 3 to the second query tells us the same about the guests 3, 0 and 1.

We can now deduce that guest 0 must sit in the middle, with guests 2 and 4 on one side and guests 1 and 3 on the other side.

After the third query, we can already be sure that the guests must sit either in the order $[3, 1, 0, 2, 4]$ or in the reversed order $[4, 2, 0, 1, 3]$. We may output either of the orders.

Code Templates and Evaluation Details in CMS

We strongly recommend using the provided code templates for C++ and Python. These check whether communication with the grader was successful and terminate gracefully when it was not.

If you do not use the provided templates, in cases where your solution is incorrect, CMS might display the wrong verdict. For example, instead of "Output isn't correct" you may receive "Execution killed by signal" or "Execution timed out (wall clock limit exceeded)".

We also recommend testing tool (see below) to test your solution locally before submitting it. The testing tool checks the outputs of your solution and reports use of invalid queries.

Testing Tool

To facilitate the testing of your solution, we provide a simple tool that you can download from CMS. The tool is optional to use. Note that the official grader on CMS is different from the testing tool.

To use the tool, you need an input file. You can use the provided example input `seatingplan.input0.txt` or make your own. The input file should start with a line that has the number T of test cases and then it should have two lines per test case: one line with the number N and then one line with the numbers $g_0, g_1, ..., g_{N-1}$.

For Python programs, say `seatingplan.py` (normally run as `pypy3 seatingplan.py`) run the testing tool as follows:

```
python3 testing_tool.py pypy3 seatingplan.py < seatingplan.input0.txt
```

For C++ programs, first compile your solution:

```
g++ -DEVAL -std=gnu++20 -O2 -pipe -static -s -o seatingplan seatingplan.cpp
```

and then run the testing tool:

```
python3 testing_tool.py ./seatingplan < seatingplan.input0.txt
```