D. 座位安排 (seatingplan)

時間限制: 4 秒
空間限制: 1024 MiB

本屆 EGOI 閉幕禮將有 N 位貴賓出席。必須按照細微的外交禮儀，以非常特定的順序安排他們在前排就坐。Noemi 歷經兩個不眠之夜才確定正確的座位順序。

Veronica 負責監督閉幕禮。她的眾多職責包括確保前排座位有正確標名。但有個小問題：Noemi 從未告訴她正確的座位順序，而且現在找不到 Noemi 了。幸運的是，攝影師 Dorka 有個 app，或者可以派上用場。

Dorka 必須準備相機，以便拍攝前排賓客的特定照片。為了設置，她需要知道每張照片有多闊，所以 Noemi 為她做了一個能快速輸出所需資訊的 app。Veronica 現在想用這個 app 來找出正確的座位安排。

N 位貴賓編號從 0 到 $N-1$。前排座位也自左至右編號，從 0 到 $N-1$。對於每個 I $(0 \le I \le N-1)$，令 g_I 表示應該坐在座位 I 的賓客（guest），令 s_I 表示賓客 I 應該坐的座位（seat）。

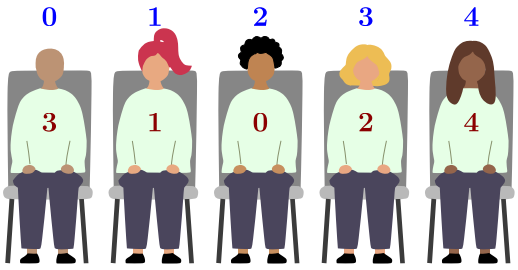

图 1　五位賓客的一排。對於這一排，$g = [3, 1, 0, 2, 4]$，而 $s = [2, 1, 3, 0, 4]$。

App 的運作方式如下：

- Dorka 輸入恰好三個不同賓客的編號 I, J, K。
- App 會告訴她，若要拍攝到三位所選賓客，照片中可見的最少賓客數。以符號表示，app 會顯示數值 $\max(s_I, s_J, s_K) - \min(s_I, s_J, s_K) + 1$。

例如，請參見 图 1 所示的情況：

- 賓客 $I = 0$, $J = 2$, $K = 4$ 分別在座位 $s_I = 2$, $s_J = 3$, $s_K = 4$。若 Dorka 選擇他們，則 app 顯示的值為 $\max(2, 3, 4) - \min(2, 3, 4) + 1 = 3$。

  換言之，包含賓客 0, 2, 4 的最窄照片就祇有這三位賓客。

- 賓客 $I = 0$, $J = 4$, $K = 3$ 分別在座位 $s_I = 2$, $s_J = 4$, $s_K = 0$。若 Dorka 選擇他們，則 app 顯示的值為 $\max(2, 4, 0) - \min(2, 4, 0) + 1 = 5$。

  換言之，包含這三位指定賓客的照片必須包含全部 5 位賓客。

請幫助 Veronica 用 Dorka 的 app 確定正確的座位順序。具體而言，你的程式應確定並輸出序列 $g_0, g_1, \ldots, g_{N-1}$。正確答案總是恰有兩組（其中一組是另一組的反轉），你可以輸出其中任何一組。你的分數取決於你的解法用 app 詢問的次數。

實作須知

⇨ 本題為交互題。你的程式將使用標準輸入和輸出與評測機通訊，格式如下描述。

你的程式應首先讀入一行，包含正整數 T，即後續的測試用例數。

對於每個測試用例，你的程式應先讀入一行，包含正整數 N，即座位數量，亦是賓客數量。

如欲詢問，你的程式應輸出一行，格式為 "? I J K"，其中 $0 \le I, J, K \le N-1$ 是三個 **不同** 的編號。

進行詢問後，你的程式應讀入一行，包含一個正整數，即對你詢問的回答。

如欲回答正確的座位順序，你的程式應輸出一行，格式為 "! g_0 … g_{N-1}"。

解決全部 T 個測試用例後，你的程式應正常終止。

請注意, CMS 上測試你解法的正式評測系統可能是 **自適應的**。換言之, 某些測試用例中, 賓客的排列並非預先確定。相反, 評測系統可能因應你程式已經提出的詢問, 來決定採用哪一個尚存可能的排列。

沖刷緩衝區 (Flushing)。 若你不使用提供的模板，請確保在輸出每一行後沖刷標準輸出，否則你的程式可能會被判為「不正確」(*Not correct*)。 Python 中，每次你使用 `input()` 讀入一行時，此事都會自動發生，你亦可以用 `print(..., flush=True)` 使之沖刷。 C++ 中，`cout << endl;` 除了列印換行符外還會沖刷緩衝；若用 printf，請使用 `fflush(stdout);`。

限制條件

- $1 \le T \le 10$。
- N 必定是 5（僅限範例）、8、40 或 2000。
- 對於每個測試用例，你最多可以詢問 10 000 次。

評分方式

你的程式將以多個測資用例進行評測，測資用例分成若干個子任務。要獲得某個子任務的分數，你必須正確解出其中全部測試。

- **子任務 0 [0 分]**: 範例 ($N = 5$)。
- **子任務 1 [9 分]**: $N = 8$。
- **子任務 2 [11 分]**: $N = 2000$，且賓客 0 和 1 相鄰而坐。
- **子任務 3 [15 分]**: $N = 40$。
- **子任務 4 [65 分]**: $N = 2000$。

子任務 1 和 2 中，正確解決所有測試用例的任何解法皆可獲全部分數。

子任務 3 和 4 中，你的解法必須正確解決所有測試用例才能得分，且你的分數取決於 Q_s，即你的解法解決一個測試用例所需的詢問次數的最大值。設 $X_s = \max(1, Q_s/N)$。子任務 3 和 4 的分數以下式計算:

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

每個子任務 score_s 的值四捨五入到最接近的整數，而你的總分為這些分數的總和。 為取得滿分，你需要以最多 55 次詢問解決子任務 3，並以最多 2597 次詢問解決子任務 4。 Q_s 值與子任務 3、4 的得分舉例如下。

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

範例

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

說明

範例輸入包含一個測試用例 ($T = 1$)，有 $N = 5$ 位賓客。 此測試用例的賓客隱藏配置對應 图 1。

範例解法的第一個詢問是 0, 2, 4。 此詢問的回答是 3，告訴我們這些賓客以某種未知順序坐在相鄰的三個座位上。

第二個詢問的回答 3 亦告訴我們關於賓客 3, 0 和 1 的同樣資訊。

我們現在可以推斷出賓客 0 必須坐在中間，賓客 2 和 4 在一側，賓客 1 和 3 在另一側。

在第三次詢問之後，我們已經可以確定賓客一定是按順序 $[3, 1, 0, 2, 4]$ 或反轉順序 $[4, 2, 0, 1, 3]$ 就坐。我們可以輸出兩者之一。

代碼範本與 CMS 測評細節

我們強烈鼓勵使用所提供的 C++ 和 Python 代碼範本。範本有檢測是否成功與評測機交互，當檢測到失敗時亦會徐徐終止（terminate gracefully）。

若你未使用提供的範本，則萬一你的解法有誤時，CMS 顯示的判定未必準確。例如，應判為「輸出不正確」(“Output isn't correct”) 的解法，可能錯判成「被訊號終止運行」(“Execution killed by signal”) 或「運行超時（現實時間超限）」(“Execution timed out (wall clock limit exceeded)”)。

亦建議提交前先使用測試工具（見下節）本地測試你的解答。測試工具會檢查你解法的輸出內容，亦會就不合規的詢問報錯。

測試工具

為方便你測試自己的解答, 我們提供一個簡單工具, 可從 CMS 下載。你可自由選用該工具。留意 CMS 的正式評測機與此測試工具有差異。

要使用該工具，需要有輸入檔案。你可以使用提供的範例輸入 `seatingplan.input0.txt` 或自行預備。輸入檔起首應為一行，包含測試用例數 T，然後每個測試用例佔兩行：一行是 N，然後一行是 $g_0, g_1, \ldots, g_{N-1}$。

對於 Python 程式，設為 `seatingplan.py`（平常以 `pypy3 seatingplan.py` 運行），運行測試工具的方式如下：

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

對於 C++ 程式，先編譯你的解答:

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

然後運行測試工具:

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