

## D. Sitzordnung (seatingplan)

An der Abschlussfeier dieser EGOI nehmen  $N$  wichtige Gäste teil. Sie alle müssen in der vordersten Reihe in einer ganz bestimmten Reihenfolge sitzen, die allen Feinheiten des diplomatischen Protokolls entspricht. Noemi hat zwei schlaflose Nächte damit verbracht, die richtige Sitzordnung zu bestimmen.

Veronica beaufsichtigt die Abschlussfeier. Eine ihrer vielen Aufgaben ist es, sicherzustellen, dass auf den Plätzen in der ersten Reihe die richtigen Namensschilder stehen. Es gibt nur ein kleines Problem: Noemi hat ihr die richtige Sitzordnung nie verraten und jetzt ist Noemi nirgends zu finden. Zum Glück hat Dorka, die Fotografin, eine App, die nützlich sein könnte.

Dorka musste ihre Kameras vorbereiten, um bestimmte Fotos von den Gästen in der ersten Reihe zu machen. Für die Einrichtung musste sie wissen, wie breit jedes Foto sein würde, also hat Noemi ihr eine App gemacht, die schnell die benötigten Informationen ausgibt. Veronica möchte die App nun nutzen, um die korrekte Sitzverteilung herauszufinden.

Die  $N$  wichtigen Gäste sind von 0 bis  $N - 1$  nummeriert. Die Plätze in der vordersten Reihe sind ebenfalls von 0 bis  $N - 1$  von links nach rechts nummeriert. Für jedes  $I$  ( $0 \leq I \leq N - 1$ ) sei  $g_I$  der Gast, der auf Platz  $I$  sitzen soll, und  $s_I$  der Platz, an dem Gast  $I$  sitzen soll.

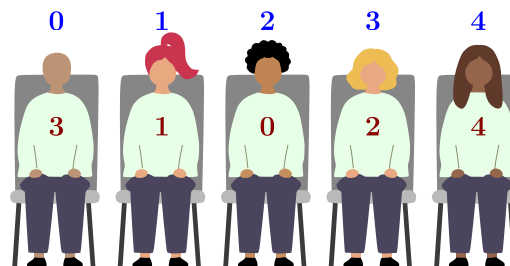


Abbildung 1: Eine Reihe mit fünf Gästen. Für diese Reihe gilt  $g = [3, 1, 0, 2, 4]$  und  $s = [2, 1, 3, 0, 4]$ .

Die App funktioniert so:

- Dorka gibt die Nummern  $I$ ,  $J$ ,  $K$  von genau drei verschiedenen Gästen ein.
- Die App zeigt ihr die minimale Anzahl an Gästen an, die sichtbar sind, wenn alle drei ausgewählten Gäste auf dem Foto zu sehen sind. Formell zeigt die App den Wert  $\max(s_I, s_J, s_K) - \min(s_I, s_J, s_K) + 1$  an.

Zum Beispiel, siehe die Situation in Abbildung 1:

- Die Gäste  $I = 0$ ,  $J = 2$  und  $K = 4$  sitzen auf den Plätzen  $s_I = 2$ ,  $s_J = 3$  und  $s_K = 4$ . Wenn Dorka diese auswählt, zeigt die App den Wert  $\max(2, 3, 4) - \min(2, 3, 4) + 1 = 3$  an.

Mit anderen Worten: Das schmalste Foto, das die Gäste 0, 2 und 4 enthält, besteht genau aus diesen drei Gästen.

- Die Gäste  $I = 0$ ,  $J = 4$  und  $K = 3$  sitzen auf den Plätzen  $s_I = 2$ ,  $s_J = 4$  und  $s_K = 0$ . Wenn Dorka diese auswählt, zeigt die App den Wert  $\max(2, 4, 0) - \min(2, 4, 0) + 1 = 5$  an.

Mit anderen Worten: Ein Foto, das die drei angegebenen Gäste enthält, muss alle 5 Gäste enthalten.

Hilf Veronica dabei, die richtige Sitzordnung mit Dorkas App zu bestimmen. Genauer gesagt soll dein Programm die Sequenz  $g_0, g_1, \dots, g_{N-1}$  bestimmen und ausgeben. Es gibt immer genau zwei richtige

Antworten (eine ist die Umkehrung der anderen), und du kannst eine der beiden ausgeben. Deine Punktzahl hängt von der Anzahl der Anfragen ab, die deine Lösung an die App stellt.

## Implementierung



Dies ist ein interaktives Problem. Dein Programm kommuniziert über die Standardein- und -ausgabe mit einem Grader, wie unten beschrieben.

Dein Programm sollte mit dem Einlesen einer Zeile beginnen, die eine positive Ganzzahl  $T$  enthält, die Anzahl der folgenden Testfälle.

Für jeden Testfall sollte dein Programm mit dem Einlesen einer Zeile beginnen, die eine positive Ganzzahl  $N$  enthält, die Anzahl der Plätze (und damit auch der Gäste).

Um eine Anfrage zu stellen, sollte dein Programm eine Zeile der Form „?  $I J K$ “ ausgeben, wobei  $0 \leq I, J, K \leq N - 1$  drei **verschiedene** Nummern sind.

Nachdem du eine Anfrage gestellt hast, sollte dein Programm eine Zeile mit einer positiven Ganzzahl einlesen, die Antwort auf deine Anfrage.

Um die richtige Sitzordnung zu liefern, sollte dein Programm eine Zeile der Form „!  $g_0 \dots g_{N-1}$ “ ausgeben.

Nachdem alle  $T$  Testfälle gelöst sind, sollte dein Programm normal enden.

Beachte, dass der offizielle Grader, der im CMS zum Testen deiner Lösung verwendet wird, **adaptiv** sein kann. Das bedeutet, dass bei einigen Testfällen die Permutation der Gäste nicht im Voraus festgelegt ist. Stattdessen kann der Grader je nachdem, welche Anfragen dein Programm bereits gestellt hat, entscheiden, welche der verbleibenden Permutationen er verwendet.

**Flushing (Puffer leeren).** Falls du die bereitgestellten Vorlagen nicht verwendest, stelle sicher, dass du den Puffer der Standardausgabe nach dem Drucken jeder Zeile leerst (flush), sonst könnte dein Programm als *Nicht korrekt* bewertet werden. In Python passiert das automatisch, wenn du `input()` zum Lesen von Zeilen verwendest, und du kannst `print(..., flush=True)` verwenden, um es zu erzwingen. In C++ leert `cout << endl;` den Puffer zusätzlich zum Drucken eines Zeilenumbruchs; falls du `printf` verwendest, nutze `fflush(stdout)`.

## Einschränkungen

- $1 \leq T \leq 10$ .
- $N$  wird 5 (nur Beispiel), 8, 40 oder 2000 sein.
- Für jeden Testfall darfst du maximal 10000 Anfragen stellen.

## Punktevergabe

Deine Lösung wird an mehreren Testfällen getestet, die in Teilaufgaben gruppiert sind. Um die Punktzahl für eine Teilaufgabe zu erhalten, musst du alle darin enthaltenen Tests korrekt lösen.

- **Teilaufgabe 0 [ 0 Punkte]:** Beispiel ( $N = 5$ ).
- **Teilaufgabe 1 [ 9 Punkte]:**  $N = 8$ .
- **Teilaufgabe 2 [11 Punkte]:**  $N = 2000$ , und die Gäste 0 und 1 sitzen nebeneinander.
- **Teilaufgabe 3 [15 Punkte]:**  $N = 40$ .
- **Teilaufgabe 4 [65 Punkte]:**  $N = 2000$ .

Für die Teilaufgaben 1 und 2 erhält jede Lösung, die alle Testfälle korrekt löst, die volle Punktzahl.

Für die Teilaufgaben 3 und 4 muss deine Lösung alle Testfälle korrekt lösen, um Punkte zu erhalten. Deine Punktzahl hängt dann von  $Q_s$  ab, der höchsten Anzahl an Anfragen, die deine Lösung für

einen Testfall benötigt hat. Sei  $X_s = \max(1, Q_s/N)$ . Die Punktzahlen für die Teilaufgaben 3 und 4 werden dann wie folgt berechnet:

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

Der Wert von  $\text{score}_s$  wird pro Teilaufgabe auf die nächste ganze Zahl gerundet, und deine Gesamtpunktzahl ist die Summe dieser Werte. Um die volle Punktzahl zu erhalten, musst du Teilaufgabe 3 in maximal 55 Anfragen und Teilaufgabe 4 in maximal 2597 Anfragen lösen. Beispielwerte für  $Q_s$  und die entsprechenden Punktzahlen für Teilaufgabe 3 und 4 sind unten dargestellt.

$Q_s$	55	56	60	70	80	100	150	10000
$\text{score}_3$	15	14	13	11	10	8	6	3

$Q_s$	2597	2800	3000	4000	5000	6000	8000	10000
$\text{score}_4$	65	58	53	35	26	21	14	11

## Beispiele

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

## Erklärung

Die Beispiel-Eingabe enthält einen Testfall ( $T = 1$ ) mit  $N = 5$  Gästen. Die versteckte Konfiguration der Gäste in diesem Testfall entspricht Abbildung 1.

Die erste Anfrage der Beispiel-Lösung ist 0, 2, 4. Die Antwort 3 auf diese Anfrage verrät uns, dass diese Gäste in irgendeiner unbekannten Reihenfolge auf drei nebeneinanderliegenden Plätzen sitzen.

Die Antwort 3 auf die zweite Anfrage verrät uns dasselbe über die Gäste 3, 0 und 1.

Wir können nun ableiten, dass Gast 0 in der Mitte sitzen muss, mit den Gästen 2 und 4 auf der einen Seite und den Gästen 1 und 3 auf der anderen Seite.

Nach der dritten Anfrage können wir sicher sein, dass die Gäste entweder in der Reihenfolge [3, 1, 0, 2, 4] oder in der umgekehrten Reihenfolge [4, 2, 0, 1, 3] sitzen müssen. Wir können eine der beiden Reihenfolgen ausgeben.

## Code-Vorlagen und Bewertungsdetails im CMS

Wir empfehlen dringend, die bereitgestellten Code-Vorlagen für C++ und Python zu verwenden. Diese überprüfen, ob die Kommunikation mit dem Grader erfolgreich war, und beenden das Programm ordnungsgemäß, falls nicht.

Falls du die bereitgestellten Vorlagen nicht verwendest, zeigt das CMS bei einer falschen Lösung möglicherweise das falsche Urteil an. Zum Beispiel erhältst du statt „Output isn’t correct“ möglicherweise „Execution killed by signal“ oder „Execution timed out (wall clock limit exceeded)“.

Wir empfehlen außerdem das Test-Tool (siehe unten), um deine Lösung lokal zu testen, bevor du sie einreichst. Das Test-Tool überprüft die Ausgaben deiner Lösung und meldet, falls ungültige Anfragen gestellt wurden.

## Test-Tool

Um das Testen deiner Lösung zu erleichtern, stellen wir ein einfaches Tool zur Verfügung, das du vom CMS herunterladen kannst. Die Verwendung des Tools ist optional. Beachte, dass sich der offizielle Grader im CMS vom Test-Tool unterscheidet.

Um das Tool zu verwenden, benötigst du eine Eingabedatei. Du kannst das bereitgestellte Beispiel `seatingplan.input0.txt` verwenden oder eine eigene erstellen. Die Eingabedatei sollte mit einer Zeile beginnen, die die Anzahl  $T$  der Testfälle enthält, gefolgt von zwei Zeilen pro Testfall: eine Zeile mit der Anzahl  $N$  und eine Zeile mit den Nummern  $g_0, g_1, \dots, g_{N-1}$ .

Für Python-Programme, z. B. `seatingplan.py` (normalerweise ausgeführt als `pypy3 seatingplan.py`), führe das Test-Tool wie folgt aus:

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

Für C++-Programme, kompiliere zuerst deine Lösung:

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

und führe dann das Test-Tool aus:

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