

## D. Seating Plan (seatingplan)

A la ceremonia de clausura de esta EGOI asistirán  $N$  invitados importantes. Todos ellos deben sentarse en la primera fila en un orden muy específico que corresponde a todos los matices del protocolo diplomático. Determinar el orden correcto de los asientos le tomó a Noemi dos noches sin dormir.

Veronica está supervisando la ceremonia de clausura. Una de sus muchas responsabilidades es asegurarse de que los asientos de la primera fila tengan los carteles con los nombres correctos. Solo hay un pequeño problema: Noemi nunca le dijo el orden correcto de los asientos, y ahora no hay forma de encontrar a Noemi. Por suerte, Dorka, la fotógrafa, tiene una aplicación que podría ser útil.

Dorka tuvo que preparar sus cámaras para poder tomar algunas fotos específicas de los invitados en la primera fila. Para la configuración, necesitaba saber qué tan ancha sería cada foto, así que Noemi le hizo una aplicación que le proporciona rápidamente la información que necesita. Veronica ahora quiere usar la aplicación para encontrar la asignación de asientos correcta.

Los  $N$  invitados importantes están numerados del 0 al  $N - 1$ . Los asientos de la primera fila también están numerados del 0 al  $N - 1$ , de izquierda a derecha. Para cada  $I$  ( $0 \leq I \leq N - 1$ ), sea  $g_I$  el invitado que se supone debe sentarse en el asiento  $I$ , y sea  $s_I$  el asiento en el que se supone debe sentarse el invitado  $I$ .

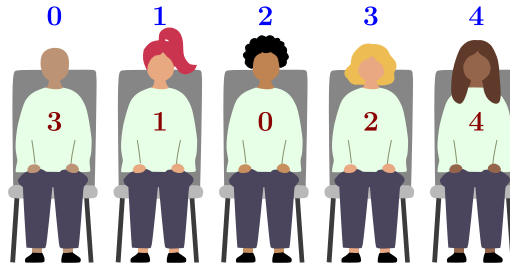


Figura 1: Una fila con cinco invitados. Para esta fila,  $g = [3, 1, 0, 2, 4]$  y  $s = [2, 1, 3, 0, 4]$ .

La aplicación funciona de la siguiente manera:

- Dorka ingresa los números  $I$ ,  $J$ ,  $K$  de exactamente tres invitados diferentes.
- La aplicación le dice el número mínimo de invitados que serán visibles si los tres invitados seleccionados están en la foto. Formalmente, la aplicación mostrará el valor  $\max(s_I, s_J, s_K) - \min(s_I, s_J, s_K) + 1$ .

Por ejemplo, mira la situación mostrada en Figura 1:

- Los invitados  $I = 0$ ,  $J = 2$  y  $K = 4$  están en los asientos  $s_I = 2$ ,  $s_J = 3$  y  $s_K = 4$ . Si Dorka los selecciona, la aplicación mostrará el valor  $\max(2, 3, 4) - \min(2, 3, 4) + 1 = 3$ .

En otras palabras, la foto más estrecha que contiene a los invitados 0, 2 y 4 tiene solo a estos tres invitados.

- Los invitados  $I = 0$ ,  $J = 4$  y  $K = 3$  están en los asientos  $s_I = 2$ ,  $s_J = 4$  y  $s_K = 0$ . Si Dorka los selecciona, la aplicación mostrará el valor  $\max(2, 4, 0) - \min(2, 4, 0) + 1 = 5$ .

En otras palabras, una foto que contiene a los tres invitados dados debe contener a los 5 invitados.

Ayuda a Veronica a determinar el orden correcto de los asientos usando la aplicación de Dorka. Más específicamente, tu programa debería determinar y mostrar la secuencia  $g_0, g_1, \dots, g_{N-1}$ . Siempre hay exactamente dos respuestas correctas (una siendo la inversa de la otra), y puedes mostrar cualquiera de ellas. Tu puntuación dependerá del número de consultas que tu solución realice a la aplicación.

## Implementación



Este es un problema interactivo. Tu programa utilizará la entrada y salida estándar para comunicarse con un evaluador en el formato descrito a continuación.

Tu programa debe comenzar leyendo una línea de entrada que contenga un número entero positivo  $T$ , el número de casos de prueba que siguen.

Para cada caso de prueba, tu programa debe comenzar leyendo una línea de entrada que contenga un número entero positivo  $N$ , el número de asientos, que también es el número de invitados.

Para realizar una consulta, tu programa debe mostrar una línea de la forma «?  $I J K$ », donde  $0 \leq I, J, K \leq N - 1$  son tres números **distintos**.

Después de realizar una consulta, tu programa debe leer una línea que contenga un número entero positivo, la respuesta a tu consulta.

Para responder con un orden de asientos correcto, tu programa debe mostrar una línea de la forma «!  $g_0 \dots g_{N-1}$ ».

Después de resolver todos los  $T$  casos de prueba, tu programa debería terminar normalmente.

Ten en cuenta que el evaluador oficial utilizado en CMS para probar tu solución puede ser **adaptativo**. Es decir, para algunos casos de prueba, la permutación de los invitados no se determina de antemano. En su lugar, el evaluador puede decidir qué permutación usar dependiendo de las consultas ya realizadas por tu programa.

**Limpieza del búfer (Flushing).** Si no estás usando las plantillas proporcionadas, asegúrate de limpiar la salida estándar después de imprimir cada línea, o de lo contrario tu programa podría ser juzgado como *Incorrecto*. En Python, esto sucede automáticamente si usas `input()` para leer líneas, y puedes usar `print(..., flush=True)` para forzarlo a limpiar. En C++, `cout << endl;` limpia el búfer además de imprimir un salto de línea; si usas `printf`, usa `fflush(stdout)`.

## Restricciones

- $1 \leq T \leq 10$ .
- $N$  será 5 (solo ejemplo), 8, 40, o 2000.
- Para cada caso de prueba, puedes realizar como máximo 10 000 consultas.

## Puntuación

Tu programa será probado en varios casos de prueba agrupados en subtareas. Para obtener la puntuación de una subtarea, debes resolver correctamente todas las pruebas que contiene.

- **Subtask 0** [ 0 puntos]: Ejemplo ( $N = 5$ ).
- **Subtask 1** [ 9 puntos]:  $N = 8$ .
- **Subtask 2** [11 puntos]:  $N = 2000$ , y los invitados 0 y 1 se sientan uno al lado del otro.
- **Subtask 3** [15 puntos]:  $N = 40$ .
- **Subtask 4** [65 puntos]:  $N = 2000$ .

Para las subtareas 1 y 2, cualquier solución que resuelva correctamente todos los casos de prueba obtendrá todos los puntos.

Para las subtareas 3 y 4, tu solución debe resolver correctamente todos los casos de prueba para obtener puntos, y tu puntuación dependerá de  $Q_s$ , el mayor número de consultas que tu solución necesitó realizar para resolver un caso de prueba. Sea  $X_s = \max(1, Q_s/N)$ . Las puntuaciones para las subtareas 3 y 4 se calculan entonces de la siguiente manera:

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

El valor de  $\text{score}_s$  se redondea al número entero más cercano por subtarea, y tu puntuación total es la suma de estas. Para obtener la puntuación máxima, necesitas resolver la subtarea 3 en un máximo de 55 consultas y la subtarea 4 en un máximo de 2597 consultas. Los valores de ejemplo de  $Q_s$  y las puntuaciones para las subtareas 3 y 4 se muestran a continuación.

$Q_s$	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

## Ejemplos de entrada/salida

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

## Explicación

La entrada de ejemplo contiene un caso de prueba ( $T = 1$ ) con  $N = 5$  invitados. La configuración oculta de invitados en este caso de prueba corresponde a Figura 1.

La primera consulta realizada por la solución de ejemplo es 0, 2, 4. La respuesta 3 a esta consulta nos dice que estos invitados se sientan, en algún orden desconocido, en tres asientos consecutivos uno al lado del otro.

La respuesta 3 a la segunda consulta nos dice lo mismo sobre los invitados 3, 0 y 1.

Ahora podemos deducir que el invitado 0 debe sentarse en el medio, con los invitados 2 y 4 a un lado y los invitados 1 y 3 al otro lado.

Después de la tercera consulta, ya podemos estar seguros de que los invitados deben sentarse en el orden [3, 1, 0, 2, 4] o en el orden inverso [4, 2, 0, 1, 3]. Podemos mostrar cualquiera de los órdenes.

## Plantillas de código y detalles de evaluación en CMS

Recomendamos encarecidamente usar las plantillas de código proporcionadas para C++ y Python. Estas verifican si la comunicación con el evaluador fue exitosa y terminan correctamente cuando no lo es.

Si no usas las plantillas proporcionadas, en los casos en que tu solución sea incorrecta, CMS podría mostrar el veredicto incorrecto. Por ejemplo, en lugar de «Output isn't correct» podrías recibir «Execution killed by signal» o «Execution timed out (wall clock limit exceeded)».

También recomendamos la herramienta de prueba (ver más abajo) para probar tu solución localmente antes de enviarla. La herramienta de prueba verifica las salidas de tu solución e informa sobre el uso de consultas no válidas.

### Herramienta de prueba

Para facilitar la prueba de tu solución, proporcionamos una herramienta sencilla que puedes descargar desde CMS. El uso de la herramienta es opcional. Ten en cuenta que el evaluador oficial en CMS es diferente de la herramienta de prueba.

Para usar la herramienta, necesitas un archivo de entrada. Puedes usar la entrada de ejemplo proporcionada `seatingplan.input0.txt` o crear la tuya propia. El archivo de entrada debe comenzar con una línea que tenga el número  $T$  de casos de prueba y luego debe tener dos líneas por caso de prueba: una línea con el número  $N$  y luego una línea con los números  $g_0, g_1, \dots, g_{N-1}$ .

Para programas en Python, digamos `seatingplan.py` (normalmente ejecutado como `pypy3 seatingplan.py`) ejecuta la herramienta de prueba de la siguiente manera:

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

Para programas en C++, primero compila tu solución:

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

y luego ejecuta la herramienta de prueba:

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