1. ¿Cuál fue lo PRIMERO que miré cuando vi el código? ¿Por qué empecé ahí?

Lo primero que miré fue cuántas veces se ejecutaba la operación principal del código, más que el resultado lógico.
Esto se nota claramente en el Reto 4, donde comparé explícitamente el número de operaciones entre una función O(n) y otra $O(n^2)$, incluso construyendo una tabla con valores concretos de n.

Empecé ahí porque entendí que dos códigos pueden dar el mismo resultado, pero lo que realmente los diferencia es cuántas veces repiten el trabajo interno.

2. ¿En qué momento sentí certeza sobre mi respuesta? ¿Qué me dio esa certeza?

La certeza apareció cuando puse números reales sobre la mesa.
Al calcular cuántas operaciones hacía cada función para valores grandes de n (100, 10,000, 1,000,000), quedó evidente que la función $O(n^2)$ crece de forma desproporcionada.

La certeza no vino solo del patrón teórico, sino de ver el crecimiento numérico (de millones a billones de operaciones), lo que reforzó que mi análisis no era solo conceptual, sino práctico.

3. Si me equivoqué o dudé, ¿cuál fue el supuesto que no verifiqué?

El supuesto que inicialmente no verifiqué fue pensar que:

"Si el código funciona bien con entradas pequeñas, entonces es aceptable".

En el Reto 3, este supuesto estaba implícito: no había todavía una comparación clara de escalabilidad.
Fue hasta el Reto 4 que confirmé que el comportamiento en grande es lo que define la calidad real del algoritmo, no el funcionamiento correcto en casos pequeños.

4. ¿Qué regla o patrón apliqué casi automáticamente? ¿De dónde viene ese automatismo?

Apliqué casi automáticamente el patrón:

- Recorrido único → O(n)
- Recorridos repetidos sobre los mismos datos → $O(n^2)$

Este automatismo viene de haber visto repetidamente ejemplos donde el crecimiento cuadrático se vuelve problemático, pero en el Reto 4 ese patrón dejó de ser solo una "regla memorizada" y pasó a estar respaldado por datos medibles.

5. Si tuviera que enseñarle a un compañero cómo analizar este código, ¿cuáles serían mis 3 pasos?

1. Identificar qué operación es la más costosa y cuántas veces se ejecuta.
2. Relacionar ese número de ejecuciones con el tamaño de la entrada n.
3. Probar mentalmente (o con números reales) qué pasa cuando n crece mucho, no solo cuando es pequeño.

Retroalimentación metacognitiva basada en tus trabajos

Fortalezas metacognitivas claras

- Pase de **describir código** a **comparar comportamientos**.
- Use **datos concretos** (tabla de operaciones) para validar mi análisis.
- Entendi que la complejidad no es solo teoría, sino un **riesgo real en producción**.
- Propuse **mecanismos reales de detección** (profiling, revisiones, alertas), lo cual indica pensamiento más cercano a ingeniería que a ejercicio escolar.

Esto es un salto importante entre el Reto 3 y el Reto 4.

Debilidad identificada

mi punto débil no es conceptual, sino **temporal**:

- El análisis de complejidad aparece **después** de pensar en si el código funciona, no **antes** de validarlo como solución.

Es decir, primero acepto el código como correcto y luego analizas si es eficiente.

Plan concreto para abordarla

A partir de ahora, aplicare el análisis de los códigos antes de solo verlos como buenos de cuanto a “vista”.

Mi “algoritmo personal” real para analizar complejidad

1. Identificar la operación dominante.
2. Contar cuántas veces se ejecuta según n.
3. Comparar ese crecimiento contra alternativas conocidas.
4. Probar mentalmente con valores grandes.
5. Validar que el peor caso sea aceptable.