

Hoja de Trucos para Entrevistas

Extraído de **Master The Coding Interview: Data Structures + Algorithms** de Andrei Neagoie

Los 3 pilares de un buen código:

1. Legible
2. Complejidad de tiempo
3. Complejidad de espacio

Habilidades que busca el entrevistador:

- Habilidades analíticas: ¿Puedes analizar y resolver problemas?
- Habilidades de codificación: ¿Escribes código limpio, simple, organizado y legible?
- Conocimientos técnicos: ¿Sabes los fundamentos del puesto para el que aplicas?
- Habilidades de comunicación: ¿Tu personalidad encaja con la cultura de la empresa?

Paso a paso para resolver un problema:

1. Cuando el entrevistador dice la pregunta, anota los puntos clave (ej., "array ordenado") en la parte superior. Asegúrate de tener todos los detalles. Muestra lo organizado que eres.
2. Asegúrate de verificar dos veces: ¿Cuáles son las entradas? ¿Cuáles son las salidas?
3. ¿Cuál es el valor más importante del problema? ¿Tiempo, espacio, memoria, etc.? ¿Cuál es el objetivo principal?
4. No seas molesto preguntando demasiadas cosas.
5. Comienza con el enfoque ingenuo/fuerza bruta, lo primero que te venga a la mente. Esto demuestra que puedes pensar bien y críticamente (no necesitas escribir este código, solo hablar de él).
6. Explica por qué este enfoque no es el mejor (ej., $O(n^2)$ o peor, no es legible, etc.).
7. Recorre tu enfoque, comenta cosas y revisa dónde podrías mejorar. ¿Alguna repetición, cuellos de botella como $O(n^2)$ o trabajo innecesario? ¿Usaste toda la información que te dio el entrevistador? El cuello de botella es la parte del código con el mayor Big O. Concéntrate en eso. A veces esto ocurre con trabajos repetidos también.
8. Antes de comenzar a codificar, recorre tu código y anota los pasos que seguirás.
9. Modulariza tu código desde el principio. Divide tu código en pequeñas piezas hermosas y agrega comentarios si es necesario.

10. Ahora empieza a escribir tu código. Ten en cuenta que cuanto más te prepares y entiendas lo que necesitas codificar, mejor te irá en el pizarrón. Nunca comiences una entrevista en pizarra sin estar seguro de cómo funcionarán las cosas, eso es una receta para el desastre.

11. Piensa en las comprobaciones de errores y en cómo podrías romper este código. Nunca hagas suposiciones sobre la entrada. Supón que la gente intentará romper tu código y que Darth Vader está usando tu función. ¿Cómo la protegerás? Siempre revisa entradas falsas que no desees. Aquí hay un truco: comenta en el código las verificaciones que quieres hacer... escribe la función y luego dile al entrevistador que escribirías pruebas ahora para hacer que tu función falle (pero no necesitas escribir las pruebas).







12. No uses nombres malos o confusos como i y j. Escribe código que se lea bien.

13. Prueba tu código: verifica sin parámetros, 0, undefined, null, arrays masivos, código asíncronico, etc. Pregunta al entrevistador si puedes hacer suposiciones sobre el código. ¿Puedes hacer que la respuesta devuelva un error? Encuentra fallos en tu solución. ¿Te estás repitiendo?

14. Finalmente, habla con el entrevistador sobre cómo mejorarías el código. ¿Funciona? ¿Hay diferentes enfoques? ¿Es legible? ¿Qué buscarías en Google para mejorar? ¿Cómo se puede mejorar el rendimiento? Posiblemente: Pregunta al entrevistador cuál ha sido la solución más interesante que ha visto para este problema.

15. Si el entrevistador está contento con la solución, generalmente la entrevista termina aquí. También es común que el entrevistador haga preguntas adicionales, como cómo manejarías el problema si toda la entrada es demasiado grande para caber en memoria o si la entrada llega como un flujo de datos. Esta es una pregunta común en Google, donde se preocupan mucho por la escalabilidad. La respuesta suele ser un enfoque de divide y vencerás: realiza el procesamiento distribuido de los datos y lee solo ciertos fragmentos de la entrada del disco en memoria, escribe la salida de vuelta al disco y combínalos después.

Lista de verificación para buen código:

- [] Funciona
- [] Buen uso de estructuras de datos
- [] Reutilización de código / No te repitas
- [] Modular: hace que el código sea más legible, mantenible y fácil de probar
- [] Menor a $O(N^2)$. Queremos evitar bucles anidados si es posible, ya que son costosos. Dos bucles separados son mejores que dos anidados.
- [] Baja complejidad espacial --> La recursión puede causar desbordamiento de pila, la copia de arrays grandes puede exceder la memoria de la máquina.

Heurísticas para sobresalir en la pregunta:

- [✓] Los hash maps suelen ser la respuesta para mejorar la complejidad de tiempo.
- [✓] Si es un array ordenado, usa un árbol binario para lograr $O(\log N)$. Divide y vencerás: divide un conjunto de datos en fragmentos más pequeños y luego repite un proceso con un subconjunto de datos. La búsqueda binaria es un gran ejemplo de esto.
- [✓] Intenta ordenar tu entrada.
- [✓] Las tablas hash y la información precomputada (ej., ordenada) son algunas de las mejores formas de optimizar tu código.
- [✓] Considera el intercambio de tiempo vs espacio. A veces almacenar un estado adicional en la memoria puede ayudar en el tiempo de ejecución.
- [✓] Si el entrevistador te da consejos o sugerencias, síguelos.
- [✓] Intercambios de espacio y tiempo: las tablas hash suelen resolver esto. Usas más espacio, pero puedes obtener una optimización de tiempo en el proceso. En programación, a menudo puedes usar un poco más de espacio para obtener un tiempo más rápido.

Y recuerda siempre: Comunica tu proceso de pensamiento lo más posible. No te preocupes por terminar rápido. Cada parte de la entrevista cuenta.