Mi análisis inicial ANTES de consultar la IA Escenario A:

Playlist de música

Hipótesis: Usaría una lista doblemente enlazada, porque permite moverse hacia adelante y atrás fácilmente y eliminar canciones sin recorrer toda la estructura.

Escenario B: Buffer de teclado

Hipótesis: Usaría un arreglo, ya que el tamaño es fijo y se necesita rapidez en el acceso.

Escenario C: Sistema de versiones

Hipótesis: Usaría una pila (stack), porque siempre se trabaja con la última versión y solo se regresa a las anteriores.

Análisis de la IA (Arquitecto de Software)

Escenario A: Sistema de playlist Operaciones

principales

Agregar canciones al final Insertar

en cualquier posición Saltar a una

canción por número Eliminar la

canción actual Frecuencia estimada

Operación	Frecuencia
Agregar al final	Alta
Saltar por número	Alta
Insertar	Media
Eliminar	Media

Requisitos de memoria

Moderados (50–500 canciones). La sobrecarga de punteros es aceptable.

Recomendación: Lista Doblemente Enlazada Justificación

(Complejidad)

Inserción/eliminación con referencia: O(1)

Recorrido: O(n)

Permite navegación bidireccional (ideal para playlists).

Aunque un arreglo tiene acceso O(1) por índice, insertar en medio cuesta O(n) por los desplazamientos.

Decisión: La flexibilidad supera el costo de memoria.

Escenario B: Buffer de teclado

Operaciones principales Insertar

caracteres Procesarlos en orden

Frecuencia

Operación	Frecuencia
Inserción	Muy alta
Eliminación (procesar)	Muy alta

Memoria
Muy limitada (256 caracteres).

Recomendación: Cola implementada con arreglo circular Justificación

Inserción: O(1)

Eliminación: O(1)

Sin fragmentación de memoria

Mejor uso de caché → menor latencia

Una lista enlazada tendría más overhead por nodo.

Decisión: Rendimiento > flexibilidad.

Escenario C: Sistema de versiones Operaciones

principales

Guardar nuevas versiones

Acceder a la última

Consultar versiones anteriores

Frecuencia

Operación	Frecuencia
Agregar versión	Alta
Acceder última	Muy alta
Navegar atrás	Media

Memoria

Alta, porque cada versión es grande → evitar estructuras con overhead innecesario.

Recomendación: Pila (Stack)

(puede implementarse con arreglo dinámico)

Justificación

Push: O(1) amortizado

Pop: O(1)

Acceso a la cima: O(1)

El patrón es claramente LIFO (Last In, First Out).

Decisión: La pila modela perfectamente el comportamiento.

Reflexión sobre las diferencias

Comparando mi análisis con el de la IA:

Escenario A: Coincidí en la lista doblemente enlazada. Confirmé que cuando hay muchas inserciones y eliminaciones, los arreglos no son ideales.

Escenario B: Pensé en un arreglo, pero aprendí que lo correcto es un arreglo circular usado como cola, porque evita mover datos.

Escenario C: Coincidí en la pila. Ahora entiendo mejor que elegir la estructura correcta depende del patrón de acceso.

Principal aprendizaje:

No se trata de cuál estructura es “mejor”, sino de cuál minimiza el costo de las operaciones más frecuentes.

Escenario	Mejor estructura	Razón principal	Complejidad clave
Playlist	Lista doblemente enlazada	Inserciones y eliminaciones flexibles	O(1) modificar
Buffer	Cola con arreglo circular	Latencia mínima	O(1) operaciones
Versiones	Pila	Acceso constante a la última versión	O(1) push/pop

Conclusiones — ¿Cuándo usar cada estructura?

Arreglos

Cuando necesitas acceso rápido por índice

Tamaño conocido o estable

Prioridad en cache y velocidad

Listas enlazadas

Muchas inserciones/eliminaciones

El orden cambia frecuentemente

Listas doblemente enlazadas

Navegación en ambos sentidos

Eliminación eficiente desde cualquier nodo

Pilas

Historial, deshacer, versiones

Patrón LIFO

Colas

Procesamiento en orden

Buffers

Sistemas en tiempo real