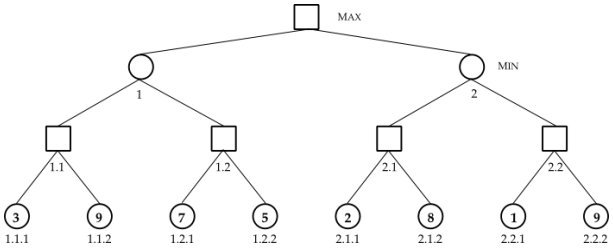
# Algoritmo Parte #2

## Introducción

En esta segunda parte profundizaremos en conceptos más avanzados: veremos cómo optimizar algoritmos, cómo reconocer y aplicar patrones estructurados de diseño en algoritmos —como divisiones y conquistas— y cómo evitar errores comunes en la construcción de algoritmos.

## Contenido




Cuando hablamos de optimización de algoritmos, nos referimos a la forma de hacer que un algoritmo funcione más rápido, use menos memoria o sea más eficiente en general. Por ejemplo, si tenemos que ordenar una lista de números, existen diversos métodos (como el ordenamiento por inserción, burbuja o por mezcla); algunos serán suficientes para listas pequeñas, pero para listas grandes necesitamos escoger un algoritmo más eficiente. Aquí entra el patrón de diseño llamado *divide y vencerás* (divide & conquer): se divide el problema en partes más pequeñas, se resuelven esas partes y luego se combinan los resultados. Este patrón aparece, por ejemplo, en el algoritmo de búsqueda binaria o en el "merge sort".

El patrón de *recursión* es otro que con frecuencia aparece: un algoritmo que se llama a sí mismo para resolver una parte del problema. Al usar recursión, debemos asegurar una condición de terminación para que el algoritmo no entre en bucle infinito. Otra consideración importante es la complejidad de un algoritmo, medida en términos de tiempo (por ejemplo "orden n" o "orden n log n") y memoria, lo que nos ayuda a comparar distintas versiones de solución antes de programar.

Los patrones de diseño de algoritmos también ayudan a reconocer soluciones estándar a problemas comunes. Por ejemplo, el patrón de *memorización* (memoization) se usa cuando un algoritmo recursivo recalcula muchas veces los mismos valores: guardamos los resultados y los reutilizamos para evitar repetir trabajo. Otro patrón es el de *iteración versus recursión*: en algunos casos es preferible transformar una solución recursiva en una iterativa para ahorrar espacio de pila y evitar límites.

En el diseño de algoritmos es muy importante identificar y evitar errores frecuentes, como bucles infinitos, condiciones de parada incorrectas, no considerar casos especiales (como listas vacías), o modificar la estructura de datos mientras la recorres sin planearlo. Además, al aplicar patrones de repetición y selección, se hace necesario estructurar bien el pseudocódigo o diagrama de flujo, para que luego la traducción al lenguaje de programación sea más clara.

Finalmente, es relevante contemplar cómo los algoritmos se aplican en la vida real: desde la forma en que un buscador trabaja para encontrar resultados, hasta cómo una aplicación de mensajería organiza mensajes o cómo se realiza un ordenamiento de datos en una base de datos. Reconocer el patrón lógico detrás del problema ayuda a escoger la solución adecuada y evita escribir código desde cero para cada nuevo reto.

**Pregunta 1.** El patrón **"divide y vencerás"** se usa cuando un problema es muy grande y difícil, y queremos hacerlo más fácil.

Por ejemplo, si tienes una lista muy larga con los puntajes de tus amigos y necesitas ordenarlos del menor al mayor, sería complicado hacerlo todo de una vez. Pero si **divides la lista en partes más pequeñas**, ordenas cada parte y luego **las unes otra vez**, el trabajo se vuelve mucho más rápido y sencillo.

Así funciona este patrón: **dividir para resolver**. Primero separas, luego solucionas cada pedazo, y al final unes todo. De esa forma, el problema grande se convierte en varios problemas pequeños y fáciles de resolver.

## Práctica

1. Describe un escenario en el que el patrón "divide y vencerás" podría aplicarse para resolver un problema de clasificación o búsqueda, indicando por qué dividir el problema ayuda.
2. Piensa en un algoritmo recursivo que hayas conocido (o que te imagines) y explica cómo definirías su condición de terminación y lo que sucedería si no existiera.
3. Analiza cómo elegirías entre una versión iterativa o recursiva de un algoritmo, teniendo en cuenta memoria y velocidad.
4. Identifica un caso real en el que la memorización (o almacenamiento de resultados intermedios) redujera el trabajo repetido, y explica por qué ese patrón mejora la eficiencia.
5. Considera qué errores podrían aparecer al modificar una estructura de datos mientras se la recorre, describe uno y sugiere cómo evitarlo.
6. Relaciona cómo se aplica en la práctica un algoritmo eficiente (por ejemplo en una búsqueda o en el procesamiento de datos grandes) y qué diferencia haría frente a un algoritmo poco optimizado.
7. Reflexiona sobre la ventaja de reconocer patrones estructurados al diseñar algoritmos: ¿cómo esto te puede ayudar a ahorrar tiempo y esfuerzo cuando programes o estudies?