**1. ¿Cuál es la importancia de la recursividad en la estrategia de diseño de algoritmos?**

La recursividad es como una herramienta que usamos para resolver problemas en pasos más pequeños. Imagina que tienes un rompecabezas grande y complicado. En lugar de intentar resolverlo todo de una vez, puedes dividirlo en partes más pequeñas y fáciles de manejar. Luego, resuelves esas partes pequeñas y, al juntarlas, resuelves todo el rompecabezas. La recursividad nos ayuda a hacer esto en la programación.

La recursividad es una herramienta esencial en el diseño de algoritmos, ya que permite abordar problemas complejos al dividirlos en instancias más pequeñas y similares. Esto reduce la complejidad del problema al tratar subproblemas, lo que a menudo simplifica la implementación y comprensión del algoritmo.

Es una herramienta poderosa que sirve para resolver cierto tipo de problemas reduciendo la complejidad y ocultando los detalles del problema.

**2. ¿Cuáles son los tipos de recursividad?**

Existen dos tipos de recursividad:

**Recursividad directa o simple:** Es como cuando te das cuenta de que necesitas resolver el mismo problema pero con datos más pequeños. Entonces, llamas a la misma solución para esos datos más pequeños.

**Recursividad indirecta o mutua:** Es como si tú le pidieras ayuda a tu amigo para resolver un problema, y tu amigo te dice que también necesita tu ayuda. Ambos trabajan juntos para resolver el problema.

Existen dos tipos de recursividad:

**Recursividad directa o simple:** Un subprograma se llama a sí mismo directamente una o más veces.

**Recursividad indirecta o mutua:** Un subprograma A llama a otro subprograma B, y B a su vez llama al subprograma A.

**Directa o simple:** un subprograma se llama a sí mismo una o más veces directamente.

**Indirecta o mutua:** un subprograma A llama a otro subprograma B y éste a su vez llama al subprograma A.

**3. Explique la técnica de dividir y conquistar.**

Dividir y conquistar es como resolver una tarea grande y difícil, dividiéndola en partes más pequeñas y manejables. Después de resolver esas partes más pequeñas, unes sus soluciones para obtener la respuesta final. Es como cuando cocinas. Divides los ingredientes y pasos, los cocinas por separado, ¡y luego tienes una deliciosa comida completa!

La técnica de dividir y conquistar implica tres pasos fundamentales:

**Dividir:** El problema se divide en subproblemas más pequeños, similares al problema original.

**Conquistar:** Los subproblemas se resuelven recursivamente si son lo suficientemente pequeños; de lo contrario, se resuelven directamente.

**Combinar:** Las soluciones de los subproblemas se combinan para crear una solución para el problema original.

**Dividir.** Dividen el problema en varios sub-problemas similares al problema original pero de menor tamaño;

**Conquistar.** Resuelven recursivamente los sub-problemas si los tamaños de los sub-problemas son suficientemente pequeños, entonces resuelven los sub-problemas de manera directa; y luego,

**Combinar.** Combinan estas soluciones para crear una solución al problema original.

**4. ¿Qué tiene más influencia en el proceso de comparar la eficacia temporal de dos algoritmos?**

Cuando comparamos la rapidez de dos algoritmos, el tipo de función que describe su velocidad es más importante que un número específico. Es un poco como comparar dos coches. No importa si uno arranca más rápido desde cero, lo que importa es cómo se comportan cuando van a velocidades altas. Al comparar algoritmos, vemos cómo crecen en velocidad a medida que el problema se vuelve más grande.

Al comparar la eficacia temporal de dos algoritmos, el tipo de función que define su complejidad asintótica tiene más influencia que la constante "c". Las medidas de eficiencia, como la complejidad temporal y espacial, no siempre son comparables directamente. La elección del algoritmo más eficiente depende de la medida de eficiencia que se priorice, como la velocidad de salida o el uso de memoria.

Según el principio de invarianza, al comparar la eficiencia temporal de dos algoritmos, tiene mayor influencia el tipo de función que la constante c.

Sin embargo, varias medidas (ejemplo, complejidad temporal, complejidad espacial) no pueden ser comparadas directamente, luego, cuál de dos algoritmos es considerado más eficiente, depende de cuál medida de eficiencia se está considerando como prioridad, ejemplo, la prioridad podría ser obtener la salida del algoritmo lo más rápido posible, o que minimice el uso de la memoria, o alguna otra medida particular.

**5. ¿Qué se usa para medir el tiempo de ejecución de un algoritmo?**

Para medir cuán rápido funciona un algoritmo, contamos las operaciones sencillas que realiza, como sumar, restar o comparar. Es como contar los pasos que das para llegar a algún lugar. No medimos el tiempo real en segundos, sino cuántos pasos toma para que el algoritmo haga su trabajo.

Para medir el tiempo de ejecución T(n) de un algoritmo, se cuenta el número de operaciones elementales, como operaciones aritméticas, asignaciones, llamadas a funciones, comparaciones lógicas y acceso a estructuras. El término "tiempo de ejecución" se refiere al número de operaciones elementales realizadas en el algoritmo, no al tiempo físico real.

Para medir T(n) usamos el número de operaciones elementales, las mismas pueden ser:

Operación aritmética.

Asignación a una variable.

Llamada a una función.

Retorno de una función.

Comparaciones lógicas (con salto).

Acceso a una estructura (arreglo, matriz, lista ligada…).

Se le llama tiempo de ejecución, no al tiempo físico, sino al número de operaciones elementales que se llevan a cabo en el algoritmo.

**6. Explique que es una medida asintomática.**

Imagina que tienes una planta en una maceta que crece a lo largo del tiempo. En lugar de mirarla día a día, observas cómo crece a medida que pasan los años. Las medidas asintomáticas son como observar cómo crece un algoritmo a medida que el problema se hace más grande. Nos ayuda a comparar algoritmos sin preocuparnos por detalles pequeños.

Una medida asintótica, también conocida como cota de complejidad, clasifica funciones para comparar su comportamiento a medida que el tamaño de los datos de entrada crece. Las medidas asintóticas permiten analizar cómo crece el tiempo de ejecución de un algoritmo en función del tamaño de entrada, independientemente del lenguaje de programación o la máquina en la que se ejecuta.

Las cotas de complejidad, también llamadas medidas asintóticas sirven para clasificar funciones de tal forma que podamos compararlas. Las medidas asintóticas permiten analizar qué tan rápido crece el tiempo de ejecución de un algoritmo cuando crece el tamaño de los datos de entrada, sin importar el lenguaje en el que esté implementado ni el tipo de máquina en la que se ejecute.

**7. ¿Cuál es la diferencia entre programación paralela y secuencial?**

Programación secuencial es como cocinar siguiendo una receta paso a paso. No empiezas con el siguiente paso hasta que hayas terminado el anterior. En la programación paralela, es como si varios cocineros trabajaran juntos para preparar la comida. Cada uno maneja un paso diferente al mismo tiempo, ¡y el trabajo se hace más rápido!

La programación secuencial implica que las tareas se ejecutan una detrás de otra en un orden secuencial. Aunque fácil de entender e implementar, puede resultar lenta si una tarea se retrasa. En contraste, la programación paralela utiliza múltiples recursos computacionales para resolver un problema simultáneamente. Esto mejora la velocidad, pero también puede ser más complejo de implementar.

La programación secuencial es cuando una tarea va después de otra. Es un proceso lento en el que, si una tarea se retrasa, el sistema completo debe esperar. La ventaja es que es fácil de entender y de implementar.

Mientras que programación paralela es el uso de múltiples recursos computacionales para resolver un problema. En el sentido más simple, la programación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional.

**8. ¿Dónde se utiliza un algoritmo distribuido?**

Un algoritmo distribuido es como un plan para que muchos amigos trabajen juntos en una tarea. Imagina que deben construir un castillo de arena gigante en diferentes lugares de la playa. Cada amigo sigue el plan y construye su parte, ¡y al final tendrán un gran castillo de arena!

Un algoritmo distribuido se emplea en sistemas distribuidos, diseñados para aprovechar las características de este tipo de sistemas, como redes de computadoras y procesamiento distribuido. Estos algoritmos permiten la colaboración de múltiples componentes para lograr tareas complejas y coordinadas en un entorno distribuido.

Un algoritmo distribuido es un tipo específico de algoritmo que se utiliza en sistemas distribuidos y que ha sido diseñado para aprovechar las características de este tipo de sistemas.

**9. ¿Cuáles son las fases de la resolución de problemas?**

Resolución de problemas es como hacer una receta de cocina. Primero, miras los ingredientes (analizas el problema). Luego, decides cómo cocinar la comida (diseñas un plan o algoritmo). Después, cocinas la comida (codificación). Luego, pruebas y ajustas los sabores (verificación y depuración). Finalmente, si encuentras una manera mejor de hacer la receta, la cambias un poco (mantenimiento).

Las fases de la resolución de problemas son:

**Análisis del problema:** Comprender los requisitos y especificaciones del problema por parte del cliente.

**Diseño o desarrollo de un algoritmo:** Crear una solución que se convierta en un algoritmo para resolver el problema.

**Codificación (implementación):** Traducir el algoritmo en un lenguaje de programación con sintaxis y sentencias adecuadas.

**Ejecución, verificación y depuración:** Ejecutar y verificar el programa, corrigiendo errores ("bugs") que puedan surgir.

**Mantenimiento:** Actualizar y modificar el programa según las necesidades cambiantes de los usuarios.

**Análisis del problema:** El problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por otra persona que encarga el programa.

**Diseño o desarrollo de un algoritmo:** Una vez analizado el problema, se diseña una solución que conducirá a un algoritmo que resuelva el problema.

**Codificación (implementación):** Esta etapa consiste en transcribir o adaptar el algoritmo a un lenguaje de programación, se tendrá que adaptar todos los pasos diseñados en el algoritmo con sentencias y sintaxis propias del lenguaje.

**Ejecución, verificación y depuración:** El programa se ejecuta, se comprueba rigurosamente y se elimina todos los errores (denominados “bugs”, en inglés) que puedan aparecer.

**Mantenimiento:** El programa se actualiza y modifica, cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de su usuarios.

**10. ¿Cómo se mide la eficiencia de un algoritmo?**

Para medir cuán eficiente es un algoritmo, contamos cuántas operaciones hace en total mientras resuelve el problema. Es como comparar dos coches para ver cuál usa menos gasolina para viajar una distancia. La eficiencia se trata de encontrar la manera más rápida y económica de resolver un problema.

Una forma de medir la eficiencia de un algoritmo es contando el número de operaciones realizadas para encontrar una solución, en función del tamaño de entrada.

Una forma de medir la eficiencia de un algoritmo es contar cuántas operaciones necesita para encontrar la respuesta con diferentes tamaños de la entrada.