Introducción a los algoritmos

1. **¿Qué es un algoritmo?**

    - Un algoritmo es un conjunto de pasos o instrucciones bien definidas que se utilizan para resolver un problema o realizar una tarea. En programación, los algoritmos se usan para procesar datos, realizar cálculos y automatizar tareas.

2. **¿Cuáles son las características de un buen algoritmo?**

    - **Eficiencia**: Debe utilizar la menor cantidad de recursos posibles (tiempo y memoria).
    - **Corrección**: Debe producir siempre la salida correcta para cualquier entrada válida.
    - **Claridad**: Debe ser fácil de entender y modificar si es necesario.

3. **¿Por qué son importantes los algoritmos en la programación?**

    - Los algoritmos son fundamentales en la programación porque permiten resolver problemas de manera estructurada y eficiente. Son la base del diseño de software y permiten optimizar procesos.

4. E**jemplo de un algoritmo sencillo Problema**: Sumar dos números

    Pseudocódigo:

    - Inicio
    - Ingresar número A
    - Ingresar número B
    - Leer A
    - Leer B
    - Sumar A + B y almacenar en C
    - Mostrar C
    - Fin

Algoritmos de Ordenamiento

- ✓ Quicksort
- ✓ Mergesort

1. **¿Cómo funciona cada algoritmo?**

    - **Quicksort**: Divide el conjunto en subproblemas más pequeños y ordena cada parte recursivamente usando un pivote.
    - **Mergesort**: Divide la lista en mitades, ordena cada mitad y luego combina las mitades ordenadas.

2. **Complejidad Temporal (Big-O)**

    - Quicksort: Mejor caso $O(n \log n)$, peor caso $O(n^2)$
    - Mergesort: Siempre $O(n \log n)$

3. **¿Cuándo usar cada algoritmo?**

    - Quicksort es eficiente en la mayoría de los casos.
    - Mergesort es preferible cuando se requiere estabilidad en la ordenación.

Preguntas Guia

Algoritmo de Búsqueda

- ✓ Búsqueda Binaria
- ✓ Búsqueda lineal

1. **¿Cómo funciona la búsqueda binaria?**

    - Busca un elemento dividiendo la lista ordenada en mitades hasta encontrar el valor.

2. **Comparación de complejidades**

    - Búsqueda Lineal: $O(n)$
    - Búsqueda Binaria: $O(\log n)$

Preguntas Guía

¿En qué casos es preferible usar búsqueda lineal en lugar de búsqueda binaria?

- Usar búsqueda lineal cuando la lista no está ordenada o es pequeña.

¿Qué pasa si intentas usar búsqueda binaria en una lista no ordenada?

- La búsqueda binaria no funciona en listas desordenadas.

Algoritmo de Grafos

- ✓ Dijkstra
- ✓ BFS (Búsqueda en Anchura)
- ✓ DFS (Búsqueda en Profundidad)

1. **¿Cómo funciona cada algoritmo?**

- **Dijkstra**: Encuentra la ruta más corta en un grafo ponderado.
- **BFS**: Explora los nodos en niveles, útil en grafos no ponderados.
- **DFS**: Explora en profundidad, útil en exploración de caminos.

2. **Complejidad Temporal**

- Dijkstra: $O((V + E) \log V)$
- BFS/DFS: $O(V + E)$

Preguntas Guía

¿Por qué Dijkstra no funciona con pesos negativos?

- **Dijkstra no funciona con pesos negativos.**

¿En qué situaciones preferirías usar DFS en lugar de BFS?

- **DFS es preferible cuando queremos recorrer todo el grafo sin prioridad en distancias.**

Algoritmo de Compresión y Otros

- Huffman
- Kadane

1. **¿Cómo funciona Huffman?**
    - Construye un árbol de codificación basado en la frecuencia de los símbolos.
2. ¿Qué resuelve Kadane?
    - Encuentra la submatriz de suma máxima en un array.

Preguntas Guía

¿Por qué el algoritmo de Huffman es óptimo para la compresión sin pérdida?

- El algoritmo de Huffman es óptimo para la compresión sin pérdida porque garantiza la codificación más eficiente posible según la frecuencia de los símbolos, minimizando la cantidad de bits necesarios para representar los datos sin perder información.

¿En qué situaciones prácticas se utiliza el algoritmo de Kadane?

- Kadane se usa en análisis de datos y finanzas.