Semana 7 Control de lectura

Métodos de análisis

Este capítulo introduce las bases teóricas y prácticas para **analizar algoritmos** y determinar su eficiencia. El objetivo es entender **cuánto tiempo** o **cuánto espacio** necesita un algoritmo en función del tamaño de la entrada.

Objetivo del análisis

- Establecer una **medida objetiva** para comparar algoritmos más allá de la implementación o el hardware.
- Centrarse en el **comportamiento asintótico** de un algoritmo: cómo crece su costo con respecto al tamaño de entrada nnn.

Medidas de eficiencia

Tiempo de ejecución

- Se mide típicamente por el número de **operaciones elementales** (comparaciones, asignaciones, etc.).
- Se distingue entre:
    - **Peor caso** (*worst case*): el mayor tiempo que puede tomar el algoritmo.
    - **Mejor caso** (*best case*): el menor tiempo.
    - **Caso promedio** (*average case*): valor esperado sobre todas las entradas posibles.

Uso de memoria

- Se considera el espacio adicional necesario además de los datos de entrada (por ejemplo, arreglos auxiliares, pilas).

Análisis asintótico

Para evitar depender de constantes y detalles de implementación, se introducen **notaciones matemáticas** que capturan la tendencia del crecimiento de una función:

Notación O grande – O(f(n))O(f(n))O(f(n))

- Representa una **cota superior asintótica**: el algoritmo no toma más tiempo que una función proporcional a f(n)f(n)f(n), para nnn suficientemente grande.
- Ejemplo: O(n2)O(n^2)O(n2), O(nlog⁡n)O(n \log n)O(nlogn), O(1)O(1)O(1)

Notación Ω (omega) – Ω(f(n))\Omega(f(n))Ω(f(n))

- Representa una **cota inferior**: el algoritmo siempre tomará **al menos** ese tiempo.

Notación Θ (theta) – Θ(f(n))\Theta(f(n))Θ(f(n))

- Representa una **cota ajustada**: el algoritmo toma un tiempo proporcional a f(n)f(n)f(n) en el caso típico.

Importancia:

- Permite comparar algoritmos independientemente del lenguaje, compilador o máquina.
- Se enfoca en lo que más importa en práctica: el **comportamiento a gran escala**.

Ejemplos de análisis de algoritmos

Se muestran varios ejemplos concretos de cómo analizar algoritmos simples:

Búsqueda secuencial (lineal)

- Recorre un arreglo hasta encontrar un elemento.
- **Peor caso:** O(n)O(n)O(n)
- **Mejor caso:** O(1)O(1)O(1)
- **Promedio:** O(n)O(n)O(n)

Búsqueda binaria

- Divide el arreglo en mitades sucesivas.
- **Peor y promedio caso:** O(log⁡n)O(\log n)O(logn)

Algoritmos de ordenamiento

- **Burbuja (bubble sort):** O(n2)O(n^2)O(n2)
- **Inserción (insertion sort):** O(n2)O(n^2)O(n2), pero puede ser O(n)O(n)O(n) en el mejor caso.
- **Fusión (merge sort):** O(nlog⁡n)O(n \log n)O(nlogn), independientemente del caso.

Estos ejemplos ilustran cómo la elección del algoritmo puede **afectar drásticamente el rendimiento** en la práctica.

Técnicas básicas de análisis

Semana 7 – Control de Lectura

Métodos de Análisis

Introducción

Este capítulo presenta los fundamentos teóricos y prácticos para **analizar algoritmos** y determinar su eficiencia. El objetivo principal es comprender **cuánto tiempo** o **cuánto espacio** requiere un algoritmo en función del tamaño de su entrada.

Objetivo del análisis

- Proporcionar una medida **objetiva y general** para comparar algoritmos, sin depender del lenguaje de programación, compilador o hardware utilizado.
- Estudiar el **comportamiento asintótico**, es decir, cómo crece el costo computacional del algoritmo a medida que aumenta el tamaño de entrada (n).

Medidas de eficiencia

Tiempo de ejecución

- Se mide por el número de **operaciones elementales** (como comparaciones, asignaciones, etc.).
- Se distinguen tres escenarios:
    - **Peor caso (worst case):** tiempo máximo requerido.
    - **Mejor caso (best case):** tiempo mínimo.
    - **Caso promedio (average case):** valor esperado considerando todas las entradas posibles.

Uso de memoria

- Se analiza el **espacio adicional** requerido por el algoritmo, además de los datos de entrada.
  Por ejemplo: estructuras auxiliares como arreglos temporales o pilas.

Análisis asintótico

Para evitar depender de factores externos como constantes o detalles de implementación, se utilizan **notaciones matemáticas** que expresan la **tendencia de crecimiento** de una función de eficiencia.

Notación O (grande) – O(f(n))

- Representa una **cota superior**: el algoritmo no tomará más tiempo que una función proporcional a *f(n)* cuando *n* sea suficientemente grande.
- Ejemplos: $O(n^2)$, O(n log n), O(1)

Notación Ω (omega) – Ω(f(n))

- Representa una **cota inferior**: el algoritmo siempre tomará al menos ese tiempo, sin importar el caso.

Notación Θ (theta) – Θ(f(n))

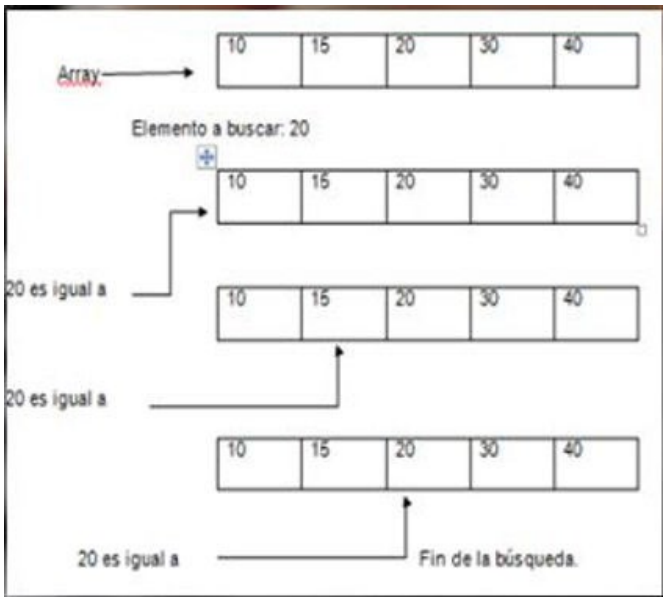
- Representa una **cota ajustada**: el algoritmo toma un tiempo **exactamente proporcional** a *f(n)* en el caso típico.

Importancia:
Estas notaciones permiten **comparar algoritmos de forma estandarizada** y enfocarse en el **comportamiento a gran escala**, que es lo más relevante en la práctica.

Ejemplos de análisis de algoritmos

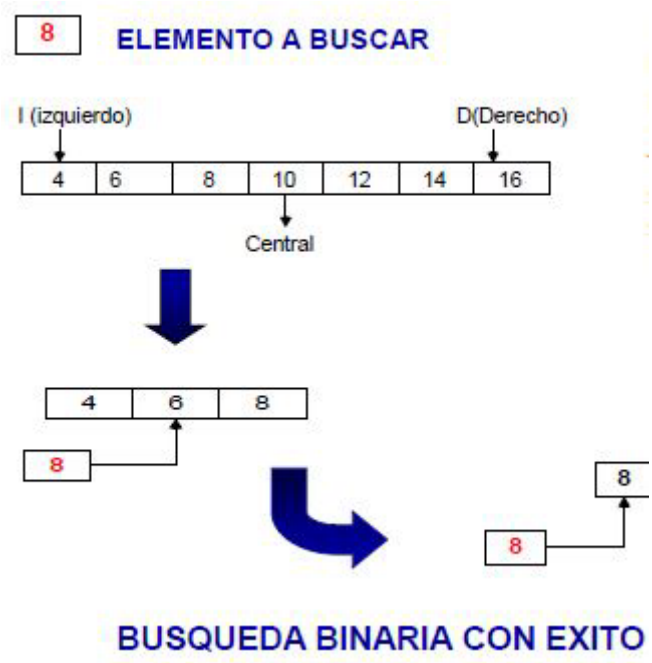
Se presentan ejemplos concretos de cómo aplicar estas ideas al analizar algoritmos simples:

Búsqueda secuencial (lineal)



- Recorre un arreglo hasta encontrar el elemento deseado.
  - Peor caso: O(n)
  - Mejor caso: O(1)
  - Promedio: O(n)

Búsqueda binaria



- Divide el arreglo en mitades sucesivas.
    - Peor y promedio caso: O(log n)

Algoritmos de ordenamiento

- **Bubble sort (burbuja):** $O(n^2)$
- **Insertion sort (inserción):** $O(n^2)$ en general, pero O(n) en el mejor caso (si el arreglo ya está ordenado).
- **Merge sort (fusión):** O(n log n) en todos los casos.

Estos ejemplos muestran cómo la **elección del algoritmo impacta el rendimiento** y puede marcar la diferencia entre una solución eficiente y una ineficiente.

Técnicas básicas de análisis

El capítulo también enseña **cómo contar operaciones paso a paso** utilizando estructuras comunes de programación.

Bucles

- Se analiza cuántas veces se ejecutan los bucles.
- Se estudia la **anidación de bucles** y cómo esta afecta la complejidad.
    - Por ejemplo: un bucle dentro de otro suele implicar una complejidad cuadrática.