**SEMANA 1 CONTROL DE LECTURA**

**Capítulo 2: Algoritmia Elemental**

**1. Introducción**

Este capítulo establece los fundamentos para analizar algoritmos, centrándose en su **eficiencia** y **correctitud**.  
Se introduce la idea de que no basta con que un algoritmo sea correcto; también debe ser eficiente en tiempo y espacio.

**2. Problemas y Ejemplares**

* **Problema computacional**: Descripción general de una tarea a resolver.  
  *Ejemplo:* ordenar una lista de números.
* **Ejemplar (instancia)**: Caso concreto del problema.  
  *Ejemplo:* la lista [5, 2, 9] es un ejemplar del problema de ordenación.

Un algoritmo debe funcionar correctamente para **todos los ejemplares válidos** del problema.

**3. Eficiencia de los Algoritmos**

La eficiencia se mide en términos de:

* **Tiempo de ejecución** (número de operaciones básicas).
* **Espacio en memoria** (cantidad de almacenamiento requerido).

La eficiencia depende del **tamaño del ejemplar**:  
*Ejemplo:* ordenar 10 números vs. 10,000 números.

**4. Análisis de "Caso Medio" y "Caso Peor"**

* **Caso peor**: El peor escenario posible para un algoritmo.  
  *Ejemplo:* en una búsqueda lineal, cuando el elemento no está en la lista.
* **Caso medio**: Rendimiento promedio bajo ciertas suposiciones sobre las entradas.

El **caso peor** es el más utilizado en análisis teórico por ofrecer un **límite superior garantizado**.

**5. ¿Qué es una Operación Elemental?**

Son operaciones básicas con tiempo constante, como:

* Asignaciones
* Comparaciones
* Sumas

*Ejemplo:* en un algoritmo de búsqueda, cada comparación cuenta como operación elemental.  
El análisis mide la **cantidad total de estas operaciones**.

**6. ¿Por qué Buscar la Eficiencia?**

Razones clave:

* Problemas grandes requieren eficiencia para ser resueltos en tiempo razonable.
* Aplicaciones críticas no toleran algoritmos lentos (ej. medicina, aviación).
* Los recursos (CPU, RAM) son **limitados**.**7. Ejemplos de Algoritmos Clásicos**

**a) Cálculo de Determinantes**

* Usa un método recursivo basado en cofactores.
* Complejidad factorial en el peor caso → **ineficiente para matrices grandes**.

**b) Algoritmos de Ordenación**

* **Selección**: compara todos los elementos → O(n2)O(n^2)O(n2).
* **Inserción**: eficiente en listas casi ordenadas.

**c) Multiplicación de Enteros Grandes**

* Método clásico: O(n2)O(n^2)O(n2).
* **Karatsuba**: más eficiente, O(n1.585)O(n^{1.585})O(n1.585).

**d) Algoritmo de Euclides (MCD)**

* Usa divisiones sucesivas.
* Complejidad: O(log⁡n)O(\log n)O(logn).

**e) Sucesión de Fibonacci**

* **Recursivo:** O(2n)O(2^n)O(2n), muy ineficiente.
* **Iterativo:** O(n)O(n)O(n).
* Se introduce **memoización** para mejorar.

**f) Transformada de Fourier**

* Se menciona la **FFT (Fast Fourier Transform)**, clave en procesamiento de señales.

**8. ¿Cuándo está Especificado un Algoritmo?**

Un algoritmo está bien definido si:

* Las **entradas y salidas** están claramente descritas.
* Cada paso es **no ambiguo y ejecutable**.
* Termina tras un **número finito de pasos**.

*Ejemplo:* el pseudocódigo de Euclides cumple con estos requisitos.

**9. Problemas y Referencias**

El capítulo concluye con ejercicios como:

* Implementar y comparar algoritmos de ordenación.
* Analizar variantes de Fibonacci y su complejidad.

Incluye referencias a textos avanzados sobre teoría de algoritmos y complejidad.

Este capítulo es clave porque:

* Introduce el **análisis riguroso de algoritmos**.
* Presenta ejemplos fundamentales para técnicas futuras (como **divide y vencerás**, **programación dinámica**).
* Enfatiza que **la elección del algoritmo depende del problema y los recursos disponibles**.