Capítulo 3: Notación Asintótica

Objetivo general:

Este capítulo establece el marco matemático necesario para analizar la eficiencia de los algoritmos, presentando **la notación asintótica** como la herramienta clave para describir cómo crece el tiempo de ejecución o el uso de recursos en función del tamaño de la entrada.

¿Por qué es importante?

A medida que el tamaño de la entrada crece, las constantes y los factores de menor orden se vuelven irrelevantes. Lo que más importa es el **comportamiento asintótico**, es decir, cómo se comporta un algoritmo cuando la entrada se hace muy grande.

Sección 3.1: O-Notación, Θ-Notación y Ω-Notación

Estas tres notaciones básicas permiten expresar el **tiempo de ejecución** o el **uso de recursos** de forma abstracta:

- **$\Theta(g(n))$ – Cota ajustada (crecimiento exacto):**
  Significa que una función $f(n)$ crece a la misma tasa que $g(n)$ *en orden de magnitud*. Es decir, está acotada tanto superior como inferiormente por $g(n)$, hasta constantes multiplicativas.

  > Ejemplo: si $f(n) = 3n^2 + 2n + 5$, entonces $f(n) = \Theta(n^2)$

- **$O(g(n))$ – Cota superior (peor caso):**
  $f(n)$ crece como máximo tan rápido como $g(n)$. No necesariamente es un límite exacto.

  > Ejemplo: Si $f(n) = 2n + 10$, entonces $f(n) = O(n)$

- **$\Omega(g(n))$ – Cota inferior (mejor caso):**
  $f(n)$ crece al menos tan rápido como $g(n)$.

  > Ejemplo: $f(n) = \Omega(n \log n)$ significa que su crecimiento no puede ser más lento que eso.

Sección 3.2: Definiciones formales

Se dan las definiciones matemáticas rigurosas usando cuantificadores ($\exists$ y $\forall$) para las notaciones anteriores. Por ejemplo:

- $f(n) = O(g(n)) \Leftrightarrow \exists$ constantes $c > 0$ y $n_0 \geq 0$ tal que
  $f(n) \leq c \cdot g(n)$ para todo $n \geq n_0$.

Estas definiciones permiten **demostrar** formalmente si un algoritmo tiene cierta complejidad.

Sección 3.3: Notaciones estándar y funciones comunes

Se introduce un conjunto de funciones que aparecen frecuentemente en el análisis de algoritmos:

- Constantes: 1, 2, 100, etc.
- Logaritmos: $\log n$, $\log_2 n$, $\log_{10} n$
- Polinomios: n, n^2, n^3
- Exponenciales: 2^n, e^n
- Factorial: $n!$
- Funciones combinatorias y sumatorias

También se enfatiza que para comparar el rendimiento de algoritmos es clave saber **ordenar funciones por su tasa de crecimiento**:

Por ejemplo, $n \log n < n^2 < 2^n < n!$