Análisis de Algoritmos
Complejidad temporal y espacial

Luis Alfredo Alvarado Rodríguez

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

17 de julio de 2025

Sumario

Complejidad de tiempo

La complejidad temporal mide el *número de operaciones* (u otras unidades de costo) que un algoritmo ejecuta en función del tamaño de su entrada n.

Objetivo

Estimar el tiempo de ejecución antes de implementar, comparar alternativas y garantizar escalabilidad.

Complejidad de tiempo

La complejidad temporal mide el *número de operaciones* (u otras unidades de costo) que un algoritmo ejecuta en función del tamaño de su entrada n.

Objetivo

Estimar el tiempo de ejecución antes de implementar, comparar alternativas y garantizar escalabilidad.

Por ejemplo, la multiplicación clásica de matrices $n \times n$ requiere $\Theta(n^3)$ operaciones aritméticas, mientras que Strassen (1969) reduce el exponente a $\approx 2{,}81$.

Complejidad de espacio

La complejidad espacial cuantifica la *memoria adicional* que necesita un algoritmo para procesar la entrada.

- Un algoritmo de búsqueda lineal en un arreglo de n elementos usa $O(1)$ espacio extra.
- La ordenación *Merge Sort* emplea $O(n)$ espacio temporal para los arreglos auxiliares.

Estimar costes de memoria es esencial en dispositivos con recursos limitados — p.ej. microcontroladores o teléfonos móviles.

Definición formal de la o pequeña

Definición 1.1. Decimos que $f(x) = o\big(g(x)\big)$ cuando $x \to \infty$ si

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0.$$

Definición formal de la o pequeña

Definición 1.1. Decimos que $f(x) = o\big(g(x)\big)$ cuando $x \to \infty$ si

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0.$$

Ejemplos:

(a) $x^2 = o(x^5)$

(b) $\operatorname{sen} x = o(x)$

(c) $14{,}709\sqrt{x} = o\big(x/2 + 7\cos x\big)$

(d) $\dfrac{1}{x} = o(1)$

(e) $23 \log x = o\big(x^{0{,}02}\big)$

Observaciones sobre la o pequeña

- Determinar una relación de tipo o puede ser trivial (ej. (a)) o requerir herramientas avanzadas — el ejemplo (e) se demuestra con la regla de *L'Hôpital*.
- En análisis de algoritmos, un límite superior más ajustado que cualquier potencia fija (ej. $o\big(n^{2,8}\big)$) asegura un mejor rendimiento asintótico que $635n^3$ para n suficientemente grande.
- No obstante, un algoritmo con complejidad mayor podría ejecutarse más rápido en tamaños pequeños: la teoría aplica a "n grandes".

Definición formal de la O grande

Definición 1.2. Decimos que $f(x) = O\big(g(x)\big)$ cuando $x \to \infty$ si existen constantes $C > 0$ y x_0 tales que

$$|f(x)| < C\, g(x) \quad \text{para todo } x > x_0.$$

Definición formal de la O grande

Definición 1.2. Decimos que $f(x) = O\big(g(x)\big)$ cuando $x \to \infty$ si existen constantes $C > 0$ y x_0 tales que

$$|f(x)| < C\, g(x) \quad \text{para todo } x > x_0.$$

Ejemplos:

- $\operatorname{sen} x = O(x)$ y, de hecho, $\operatorname{sen} x = O(1)$.
- $x^3 + 5x^2 + 77 \cos x = O\big(x^5\big)$.
- $\dfrac{1}{1+x^2} = O(1)$ pero también $= o(1)$ (más preciso).

Comparación entre O y o

Precisión

La relación o es más fuerte que O: $f = o(g)$ implica $f = O(g)$, pero no viceversa.

Cuando basta O

- Cotas superiores amplias.
- Análisis de complejidad "nivel ingeniería".
- Diseño de estructuras de datos.

Cuando se prefiere o

- Mejora teórica fina (algoritmos sub-cuadráticos, etc.).
- Demostraciones de límites inferiores.
- Criptografía y estimaciones de probabilidad de colisión.

Scalene: perfilador rápido y preciso

- **Scalene** es un *profiler* de muestreo para Python que mide tiempo de CPU, uso de memoria y actividad de GPU *línea por línea*.
- Distingue entre tiempo ejecutado en Python puro y en extensiones en C/C++/Fortran; soporta programas multiproceso y multihilo.
- Genera informes detallados en texto o HTML con mapas de calor que resaltan cuellos de botella y fugas de memoria.
- No requiere instrumentación manual: `$ pip install scalene`
  `$ scalene mi_script.py`
- Complementa el análisis asintótico al ofrecer evidencias empíricas del rendimiento real de un algoritmo.