# Análisis de Algoritmos

**Alan Reyes-Figueroa**

**Teoría de la Computación** **(Aula 21) 01.octubre.2025**

Definición

Inputs

Ejemplos

Notación Asintótica

# Análisis de Algoritmos

- Estimar los recursos (tiempo y memoria) que un algoritmos requiere para funcionar.
  - Estructura
  - Operaciones
- Algoritmos:  argumentos de entrada
- El consumo de recursos del algoritmo se escribe en función del "tamaño" de estos *inputs*.

# Inputs: Ejemplos

- Input: Arreglo a.
- Tamaño: número de elementos del arreglo a.

- Input: Un número entero n.
- Tamaño: Número de bits que requiere la representación binaria de n.

# Inputs: Ejemplos

- Input: Grafo G.
- Tamaño: Número de nodos de G.
  Número de nodos + aristas.

- Input: Base de datos.
- Tamaño: Número de variables x Número de registros.

# Tiempo de Ejecución

Buscamos determinar el tiempo de ejecución (*running time*) de un algoritmo, esto es, el número de pasos u operaciones primitivas realizadas.

Ejemplo: (Algoritmo para contar coincidencias en un arreglo):

Input: Array a;  int b.

```
n = len(a)
count = 0
For i in range(0, n):
      if (a[i] == b):
            count = count + 1
```

Asignación t = $c_1$

Asignación t = $c_1$

Ciclo t = n *

Comparación t = $c_2$

Asignación t = $c_1$

Suma t = $c_3$

# Ejemplo 1

Ejemplo: sumar los valores en un arreglo.
Inputs: array **a**.

| Operación | Tiempo |
|---|---|
| **n** = len(**a**) | |
| **suma** = 0 | |
| | |
| for **i** in range(0, **n**): | |
| **suma** = **suma** + **a**[**i**] | |
| return **suma** | |

# Ejemplo 1

Ejemplo: sumar los valores en un arreglo.
Inputs: array **a**.

| Operación | Tiempo | |
|---|---|---|
| **n** = len(**a**) | $t = c_0 + c_1$ | (lectura + asignación) |
| **suma** = 0 | $t = c_1$ | (asignación) |
| | Ciclo: 1 asignación i al inicio | |
| for **i** in range(0, **n**): | $t = c_0+c_1+c_2+c_3$ | (comp + lec + suma + asig) |
| **suma** = **suma** + **a**[**i**] | $t = 2c_0 + c_3 + c_1$ | (2 lectura + suma + asign.) |
| return **suma** | $t = c_0 + c_1$ | (lectura + asignación) |

¿Cuántas operaciones hace en total?

$$T = 1 + 2c_0 + 3c_1 + n(3c_0 + 2c_1 + c_2 + 2c_3)$$

# Sumas útiles

$$\sum_{i=1}^{n} 1 = n,$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2},$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{2},$$

$$\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$$

# Ejemplo 2

Ejemplo: Hacer el ordenamiento de un array mediante *bubblesort*. Inputs: array **a**.

| Operación | Tiempo |
|---|---|
| **n** = len(**a**) | |
| for **i** in range(0, **n**): | |
| for **j** in range(**0**, **n-i-1**): | |
| if (**a**[**j**] > **a**[**j+1**]): | |
| **temp** = **a**[**j**] | |
| **a**[**j**] = **a**[**j+1**] | |
| **a**[**j+1**] = **temp** | |
| return **a** | |

# Ejemplo 2

Ejemplo: Hacer el ordenamiento de un array mediante *bubblesort*. Inputs: array **a**.

| Operación | Tiempo |
|---|---|
| **n** = len(**a**) | t = $c_0 + c_1$ (lectura + asignación) |
| for **i** in range(0, **n**): | Ciclo: 1 asignación i al inicio<br>t = $c_0+c_1+c_2+c_3$ (comp + lec + suma + asig) |
| for **j** in range(**0**, **n-i-1**): | Ciclo: 1 asignación j al inicio<br>t = $c_0+c_1+c_2+c_3$ (comp + lec + suma + asig) |
| if (**a**[**j**] > **a**[**j+1**]): | t = $2c_0 + c_2$ (2 lecturas + comparación) |
| **temp** = **a**[**j**] | t = $c_0 + c_1$ (lectura + asignación) |
| **a**[**j**] = **a**[**j+1**] | t = $c_0 + c_1$ (lectura + asignación) |
| **a**[**j+1**] = **temp** | t = $c_0 + c_1$ (lectura + asignación) |
| return **a** | t = $c_0 + c_1$ (lectura + asignación) |

# Ejemplo 3

Ejemplo: Contar ocurrencias de **b** en un arreglo.
Inputs: array **a**, int **b**.

| Operación | Tiempo | |
|---|---|---|
| **n** = len(**a**) | $t = c_0 + c_1$ | (lectura + asignación) |
| **count** = 0 | $t = c_1$ | (asignación) |
| for **i** in range(0, **n**): | Ciclo: 1 asignación al inicio para i<br>$t = c_0 + c_3 + c_1$ | (lectura + suma + asign.) |
| if (**a**[**i**] == **b**): | $t = 2c_0 + c_2$ | (lectura + lectura + comp.) |
| **count** = **count** + 1 | $t = c_0 + c_3 + c_1$ | (lectura + suma + asign.) |
| return **count** | $t = c_0 + c_1$ | (lectura + asignación) |

¿Cuántas operaciones hace en total?

$$T = 1 + 2c_0 + 3c_1 + n(3c_0+c_1+c_2+c_3) + k(c_0+c_1+c_3)$$

# Ejemplo 4

Ejemplo: Hallar el máximo de un arreglo.
Inputs: array **a**.

| Operación | Tiempo | |
|---|---|---|
| **n** = len(**a**) | $t = c_0 + c_1$ | (lectura + asignación) |
| **max** = a[0] | $t = c_0 + c_1$ | (lectura + asignación) |
| for **i** in range(1, **n**): | Ciclo: 1 asignación al inicio para i<br>$t = c_0 + c_3 + c_1$ | (lectura + suma + asign.) |
| if (**a**[**i**] > **max**): | $t = 2c_0 + c_2$ | (lectura + lectura + comp.) |
| **max** = **a**[**i**] | $t = c_0 + c_1$ | (lectura + asignación) |
| return **max** | $t = c_0 + c_1$ | (lectura + asignación) |

¿Cuántas operaciones hace en total?

$$T = 1 + 3c_0 + 3c_1 + n(3c_0+c_1+c_2+c_3) + k(c_0+c_1)$$

# Tiempo de Ejecución

- No calculamos directamente el tiempo de ejecución (en ns, μs) por varias razones:
  - no se comporta igual en cada máquina
  - variabilidad
  - dificultad en los cálculos.

- Es mucho más simple calcular el número de operaciones ejecutadas dentro del algoritmos en función de tamaño del input.

# Escenarios

- Para un mismo algoritmo (y mismos *inputs*) podemos tener variaciones en el tiempo de ejecución de un algoritmo.

- Consideramos tres escenarios:
  - worst-case (peor caso),
  - average-case (caso promedio),
  - best-case (mejor caso).

# Ejemplo

Ejemplo: (Algoritmo para contar coincidencias en un arreglo, versión simplificada):

Input: Array a;  int b.

| Operación | Tiempo |
|---|---|
| `n = len(a)` | Asignación t = $c_1$ |
| `count = 0` | Asignación t = $c_1$ |
| `For i in range(0,n):` | Ciclo t = n * |
| `    if (a[i] == b):` | Comparación t = $c_2$ |
| `        count = count + 1` | Asignación t = $c_1$<br>Suma t = $c_3$ |

¿Cuántas operaciones hace el algoritmo?

$$T = c_1 + c_1 + n(c_2 + k(c_1 + c_3))$$

# Ejemplo

Analizamos tres posibles casos:

Mejor Caso:

$T = 2c_0 + 2c_1 + c_2 + n(2c_0 + 2c_2)$

Peor Caso:

$T = 2c_0 + 2c_1 + c_2 + n(2c_0 + 2c_2) + n(c_0 + c_1 + c_3)$

Caso Promedio:

$T = 2c_0 + 2c_1 + c_2 + n(2c_0 + 2c_2) + n/2(c_0 + c_1 + c_3)$

# Ejemplo

- Si construimos una fórmula para contar las operaciones del algoritmo, a los coeficientes en el mejor caso los podemos resumir en constantes a y b, así como en el peor caso en a, b y c.
- Para el mejor caso tendremos una función lineal como tiempo de ejecución, mientras que para el peor caso tendremos una cuadrática.
- Nos interesa: comparar dos algoritmos en cuanto a su tiempo de ejecución (**tasa de crecimiento**).

# Notación Asintótica

- Notación **big-Oh**: $O(g(x))$

Decimos que f es **O-grande** respecto de g, $f(\mathbf{x}) = O(g(\mathbf{x}))$, cuando $\mathbf{x} \rightarrow \mathbf{a}$, si existe una constante $C > 0$ tal que

$$|f(\mathbf{x})| \leq C|g(\mathbf{x})|, \text{ para todo } |\mathbf{x}-\mathbf{a}| \leq r.$$

- Equivalentemente, $f(\mathbf{x}) = O(g(\mathbf{x}))$ cuando $\mathbf{x} \rightarrow \mathbf{a}$ si existe $C > 0$ tal que

$$\lim_{\mathbf{x} \rightarrow a} |f(x)/g(x)| \leq C.$$

# Notación Asintótica

- Notación **big-Oh**: $O(g(x))$

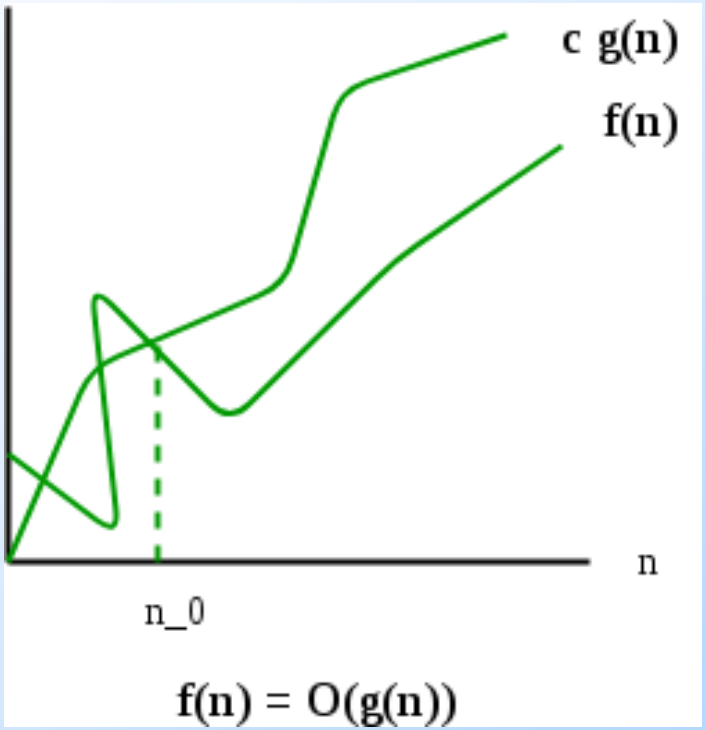
Decimos que f es **O-grande** respecto de g, $f(\mathbf{x}) = O(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si existen constantes positivas r y C con

$$|f(\mathbf{x})| \leq C|g(\mathbf{x})|, \text{ para todo } |\mathbf{x}| \geq r.$$

- Equivalentemente, $f(\mathbf{x}) = O(g(\mathbf{x}))$ cuando $\mathbf{x} \to \infty$ si existe $C > 0$ tal que

$$\lim_{\mathbf{x}\to\infty} |f(x)/g(x)| \leq C.$$

# Notación Asintótica




- f(n) = O(g(n)) quiere decir: asintóticamente (para valores muy grandes de n), g crece mucho rápido que f.

# Notación Asintótica

- Notación **big-Omega**: $\Omega(g(x))$

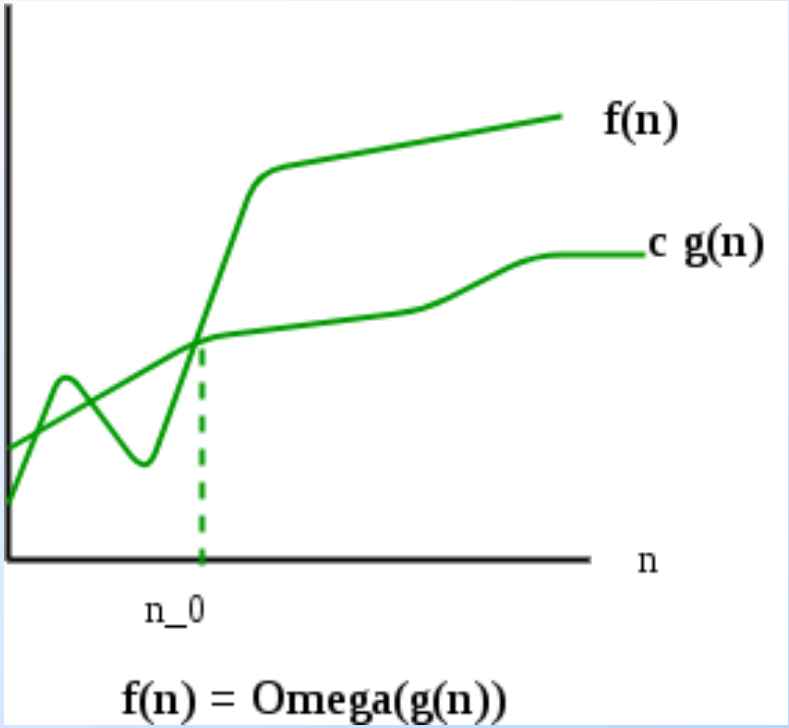
Decimos que f es **Ω-grande** respecto de g, $f(\mathbf{x}) = \Omega(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si existen constantes positivas r y C con

$$|f(\mathbf{x})| \geq C|g(\mathbf{x})|, \text{ para todo } |\mathbf{x}| \geq r.$$

- Equivalentemente, $f(\mathbf{x}) = \Omega(g(\mathbf{x}))$ cuando $\mathbf{x} \to \infty$ si existe $C > 0$ tal que

$$\lim_{\mathbf{x}\to\infty} |f(x)/g(x)| \geq C.$$

# Notación Asintótica




- f(n) = Ω(g(n)) quiere decir: asintóticamente (para valores muy grandes de n), f crece mucho rápido que g.

# Notación Asintótica

- Notación **big-Theta**: $\Theta(g(x))$

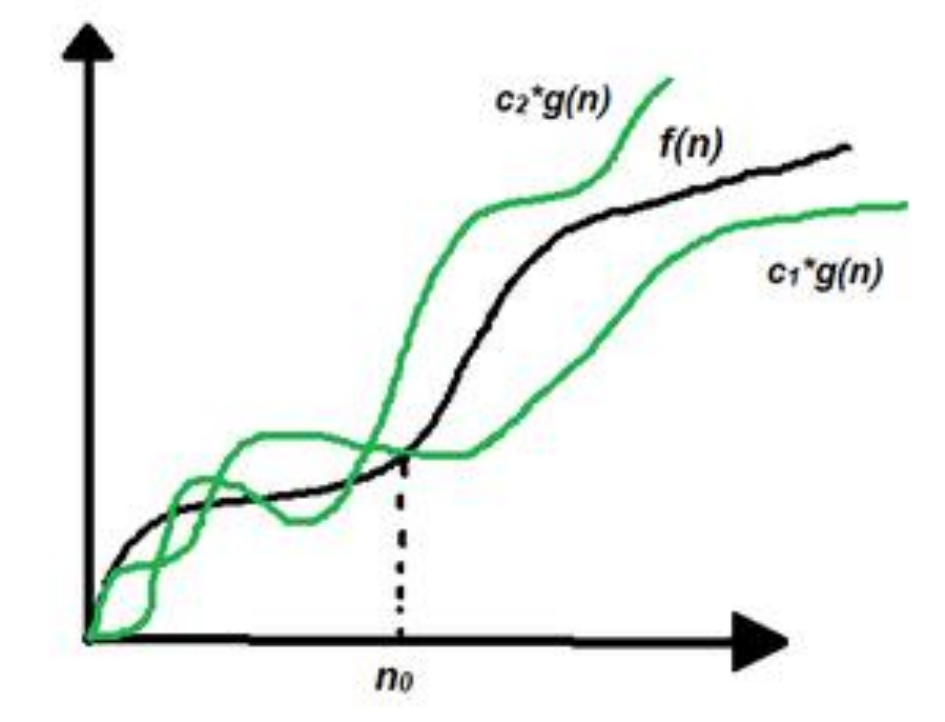
Decimos que f es **Θ-grande** respecto de g, $f(\mathbf{x}) = \Theta(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si existen constantes positivas r y $c_1$, $c_2$ con

$c_1|g(\mathbf{x})| \leq |f(\mathbf{x})| \leq c_2|g(\mathbf{x})|$, para $|\mathbf{x}| \geq r$.

- Equivalentemente, $f(\mathbf{x}) = \Theta(g(\mathbf{x}))$ cuando $\mathbf{x} \to \infty$ si existe $C > 0$ tal que

$$c_1 \leq \lim_{\mathbf{x}\to\infty} |f(x)/g(x)| \leq c_2.$$

# Notación Asintótica




- f(n) = O(g(n)) quiere decir: asintóticamente (para valores muy grandes de n), f y g crecen de forma similar.

# Notación Asintótica

- Notación **little-oh**: $o(g(x))$

Decimos que f es **o-pequeña** respecto g, $f(\mathbf{x}) = o(g(\mathbf{x}))$, cuando $\mathbf{x} \rightarrow \infty$ si

$$\lim_{\mathbf{x}\rightarrow\infty} |f(x)/g(x)| = 0.$$

# Notación Asintótica

Típicamente vamos a tener:

$f(x) = O(g(x)) \Rightarrow g(x) = \Omega(f(x))$

$f(x) = \Omega(g(x)) \Rightarrow g(x) = O(f(x))$

$f(x) = o(g(x)) \Rightarrow g(x) = \Omega(f(x))$ y

$\lim_{x \to \infty} |g(x)/f(x)| = \infty$

$f(x) = \Theta(g(x)) \Leftrightarrow g(x) = \Theta(f(x))$

Si $f(x) = \Theta(g(x))$ y $\lim_{x \to \infty} |g(x)/f(x)| = 1$,
f y g son asintóticamente equivalentes.

# Ejemplos

- Ejemplo 1: Estudiar la relación asintótica entre las funciones
  $f(n) = n^3 - n + 1$ $g(n) = n^3$

- Ejemplo 2: ¿Qué es $f(n) = O(\log n)$?

- Ejemplo 3: ¿Qué significa $f(n) = O(1)$?

# Ejemplos

- Ejemplo 4: ¿Cuál función es mayor?
  $f(n) = \log n$ $g(n) = \text{sqrt}(n)$

- Ejemplo 5: ¿Cuál es mayor?
  $f(n) = 0.5n^{1.5}$ $g(n) = 25n \log_{10} n$

- Ejemplo 6: ¿Cuál es mayor?
  $f(n) = n^3 + 5$ $g(n) = n^3 - 1$

# Ejemplos

- Ejemplo 7: ¿Cuál es mayor?
  $f(n) = n^{1000}$ $g(n) = 5^n$

- Ejemplo 8: ¿Cuál es mayor?
  $f(n) = 10^n$ $g(n) = n^n$

- Ejemplo 9: ¿Qué es mayor?
  $f(n) = n^n$ $g(n) = n!$

# Ejemplos

- Ejemplo 10: Hay dos algoritmos A y B, con tiempos de ejecución
  $T_A(n) = 5n \log_{10} n$ ms
  $T_B(n) = 25n$ ms

- ¿Cuál es mejor asintóticamente?

- Cuál es mejor para resolver un problema de tamaño n=512?

# Growth ratio

- O(log(n))
- O(√n)
- O(n)
- O(nlog(n))
- $O(n^2)$
- $O(n^3)$
- …

- $O(2^n)$
- $O(3^n)$
- $O(10^n)$
- …
- $O(n^n)$
- O(n!)

# Ejemplos: Growth

| | |
|---|---|
| O(1) | hacer una operación arit. |
| (log(n)) | búsqueda binaria |
| O(n) | búsqueda lineal |
| O(nlog(n)) | MergeSort |
| $O(n^2)$ | suma de matrices,<br>*shortest path* entre 2 nodos<br>*Knapsack problem* |
| $O(n^3)$ | producto de matrices<br>Dijkstra en grafo completo |
| $O(k^n)$ | optimización finita exhaustiva<br>*n-queens* |
| O(n!) | determinante por cofactores<br>*traveling salesman problem* |