# Notación asintótica y análisis de algoritmos

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y T.I.

CI2612 Algoritmos y Estructuras de Datos II



## Plan



# Definiciones de Algoritmo

## Wikipedia

*An algorithm is a set of instructions, typically to solve a class of problems or perform a computation. Algorithms are unambiguous specifications for performing calculation, data processing, automated reasoning, and other tasks.*

## Webster Dictionary

*A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation*

## Donald Knuth

*An algorithm is a finite, definite, effective procedure, with some input and some output.*





# Orden asintótico de crecimiento

## *O*-notación

$$O(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^{\geq 0} | (\exists c \in \mathbb{R}^{+})(\forall n \in \mathbb{N})[f(n) \leq c * g(n)]\}$$

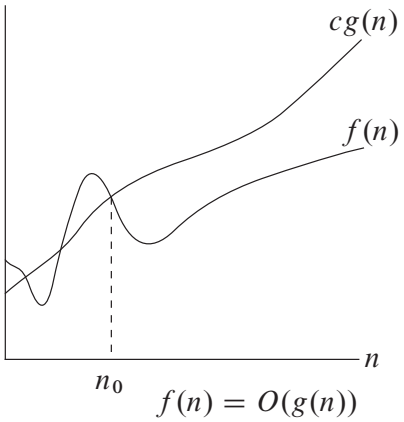



Figura: Fuente [1]

# Orden asintótico de crecimiento

## Ω-notación

$$\Omega(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^{\geq 0} | (\exists c \in \mathbb{R}^{+})(\forall n \in \mathbb{N})[f(n) \geq c * g(n)]\}$$

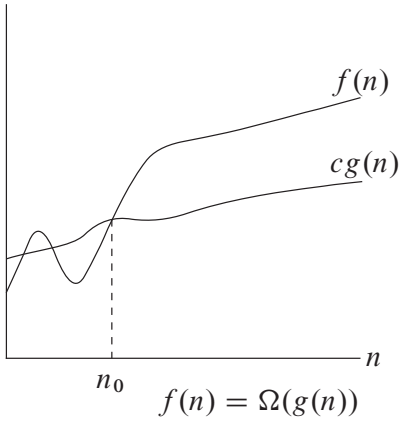



Figura: Fuente [1]





# Orden asintótico de crecimiento

## Θ-notación

Se tiene que $f \in \Theta(g(n))$ si $f \in O(g(n))$ y $f \in \Omega(g(n))$.

$$\Theta(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^{\geq 0} | (\exists c_1, c_2 \in \mathbb{R}^{+})(\forall n \in \mathbb{N})[c_1 * g(n) \leq f(n) \leq c_2 * g(n))]\}$$

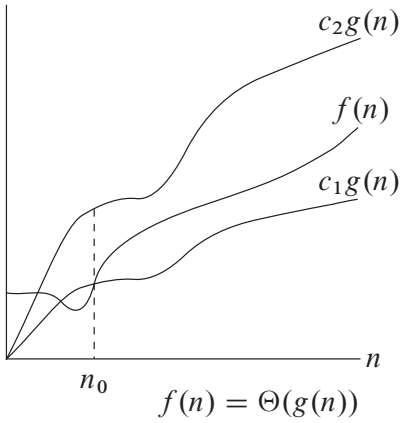



Figura: Fuente [1]

# Sumas

$$\sum_{k=1}^{n} k = 1 + 2 + , \ldots , + n \tag{1}$$

$$= \frac{1}{2} n(n+1) \tag{2}$$

$$= \Theta(n^2) \tag{3}$$

$$\sum_{k=1}^{n} (ca_k + b_k) = c \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k \tag{4}$$

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} \tag{5}$$

$$H_n = \sum_{k=1}^{n} \frac{1}{k} = \log_e n + O(1) \tag{6}$$





# Sumas

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} \quad (|x| < 1) \tag{7}$$

$$\sum_{k=0}^{\infty} k x^k = \frac{1}{(1 - x)^2} \quad (|x| < 1) \tag{8}$$

$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} \left( \frac{1}{k} - \frac{1}{k+1} \right) = 1 - \frac{1}{n} \tag{9}$$

$$\log \prod_{k=1}^{n} a_k = \sum_{k=1}^{n} \log a_k \tag{10}$$

# Sobre el análisis de algoritmos

- Se asume un computador de un procesador
- Se usa el modelo de computación random-access machine (RAM)
- Las operaciones son ejecutadas secuencialmente
- La computadora tiene operaciones matemáticas básicas de aritmética, movimiento de datos, control de datos
- Los tipos de datos de RAM son enteros y reales
- El tamaño de la entrada de un algoritmo se mide como el número de elementos de la entrada
- El tiempo de ejecución (*running time*) de un algoritmo será medido como el número de operaciones o pasos ejecutados por el algoritmo





# Tipos de análisis

- Peor caso: Cota superior del tiempo de ejecución de un algoritmo
- Mejor caso: Cota inferior del tiempo de ejecución de un algoritmo
- Caso promedio: Comportamiento esperado del tiempo de ejecución de un algoritmo, dada una entrada aleatoria

# Comparación del tiempo de ejecución de los algoritmos

- Se quiere evitar hacer una comparación que dependa del tipo de computador o lenguaje de programación usado, entre otros.
- Se expresa el tiempo de ejecución como una función del tamaño de la entrada
- La idea es poder comparar las funciones de tiempo de ejecución de dos algoritmos





# Ejemplo 1

**Procedimiento** Algoritmo1($A$ : *Arreglo*)

```
1 inicio
2     A[0] ← 1 /* costo c1 */
3     A[1] ← 5 /* costo c1 */
4     A[2] ← 8 /* costo c1 */
5     A[3] ← 0 /* costo c1 */
6     A[4] ← 12 /* costo c1 */
```

- Tiempo de ejecución $= c_1 + c_1 + c_1 + c_1 + c_1 = 5c_1$
- Tiempo de ejecución es $O(1)$

# Ejemplo 2

**Procedimiento** Algoritmo2($A$ : *Arreglo*, $n$ : *Int*)

```
1 inicio
2     para i ← 0 a n /* costo c2                    */
3      hacer
4         A[i] ← 1 /* costo c1                     */
```

- Tiempo de ejecución $= (n+1)c_2 + nc_1 = n(c_1 + c_2) + c2$
- Tiempo de ejecución es $O(n)$





# Ejemplo 3

**Función** Algoritmo3($matriz$ : *Arreglo2dim*, $n$ : *Int*)

```
1 inicio
2     acc ← 0 /* costo c1                          */
3     para i ← 0 a n /* costo c2                   */
4      hacer
5         para j ← 0 a n /* costo c2               */
6          hacer
7             acc ← matriz[i][j] /* costo c3       */
8     devolver acc /* costo c4                     */
```

- Tiempo de ejecución $= c_1 + (n+1)c_2 + n(n+1)c_2 + n^2c_3 + c_4$
- Tiempo de ejecución es $O(n^2)$

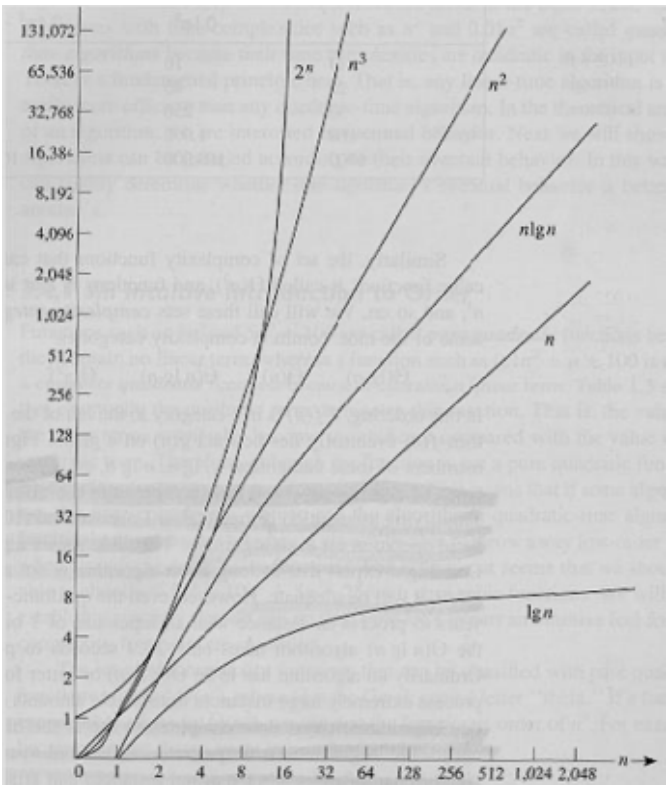
# Análisis asintótico de los algoritmos

- Se compara las funciones de tiempo de ejecución de dos algoritmos usando la tasa de crecimiento de las caracterizan
- Se compara dos funciones de tiempo de ejecución $f(n)$ y $g(n)$ asintóticamente
- Por ejemplo, el tiempo del algoritmo 3 es $O(n^2)$, es mayor que el tiempo de los algoritmos 1 y 2, que son $O(1)$ y $O(n)$ respectivamente, porque tiene una tasa de crecimiento mayor





# Ilustración de órdenes de crecimiento




# Problema de ordenamiento

## Definición

Dada una secuencia de $n$ números $a_1, a_2, a_3, \ldots a_n$, se quiere obtener una permutación $a'_1, a'_2, a'_3, \ldots a'_n$ de la entrada, tal que la secuencia resultante esté ordenada $a'_1 \leq a'_2 \leq a'_3 \leq \cdots \leq a'_n$





# Sobre Insertion Sort

- Similitud con el ordenamiento de una mano de cartas
- Se comienza con las cartas en la mesa boca abajo y la mano vacía
- Se toma una carta de la mesa y se coloca en la mano en la posición correcta
- Al remover todas las de la mesa, las cartas en la mano se encuentran ordenadas

# Ejemplo de Insertion Sort

Se quiere ordenar el arreglo $A = [5, 2, 4, 6, 1, 3]$

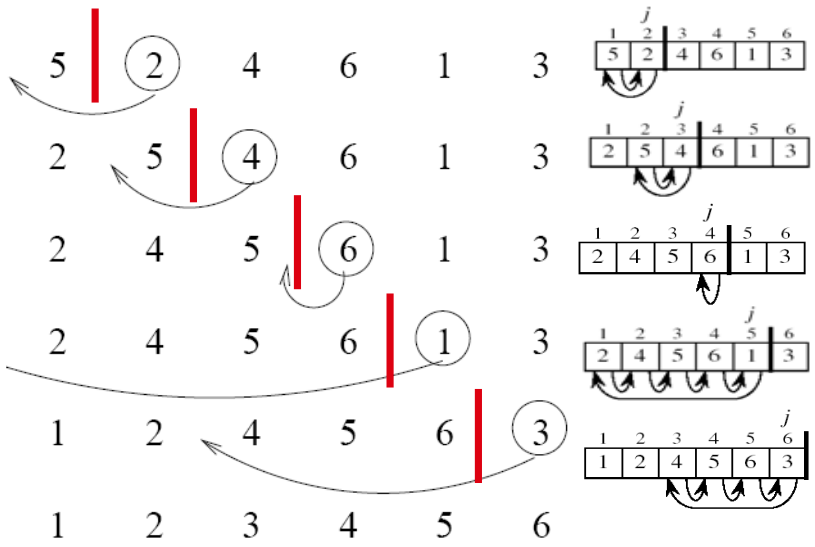



Figura: Ilustración de la ejecución de algoritmo Insertion Sort. Fuente [1]





# Insertion Sort

| INSERTION-SORT*(A)* | cost | times |
|---|---|---|
| **for** j ← 2 **to** n | $c_1$ | n |
| **do** key ← A[ j ] | $c_2$ | n-1 |
| ▷Insert A[ j ] into the sorted sequence A[1 . . j -1] | 0 | n-1 |
| i ← j - 1 | $c_4$ | n-1 |
| **while** i > 0 and A[i] > key | $c_5$ | $\sum_{j=2}^{n} t_j$ |
| **do** A[i + 1] ← A[i] | $c_6$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| i ← i – 1 | $c_7$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| A[i + 1] ← key | $c_8$ | n-1 |

Figura: Algoritmo Insertion Sort junto con el costo y las veces que se ejecuta cada instrucción. Se tiene que $t_j$ es el número de veces que ciclo WHILE es ejecutado. Fuente [1]

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

# Análisis del mejor caso de Insertion Sort

- Ocurre cuando la secuencia está ordenada
- Cada ciclo WHILE se ejecuta solo una vez, esto es $t_j = 1$
- El tiempo de ejecución de $\Theta(n)$





# Análisis del peor caso de Insertion Sort

- El peor caso es cuando la secuencia de entrada en ordenada en orden decreciente
- En el ciclo WHILE siempre se cumple que $A[i] > key$
- Se termina comparando cada nuevo elemento a insertar con todos los elementos ya insertados (en la mano)
- El tiempo de ejecución es $\Theta(n^2)$

# Referencias


T. Cormen, C. Leirserson, R. Rivest, and C. Stein.
*Introduction to Algorithms*.
McGraw Hill, 3ra edition, 2009.