PUCP

Algoritmia y Estructura de Datos

2024

Profesores:

Cueva, R. | **Allasi, D.** | **Roncal, A.** | **Huamán, F.**

0581 | 0582 | 0583 | 0584

Capítulo 3

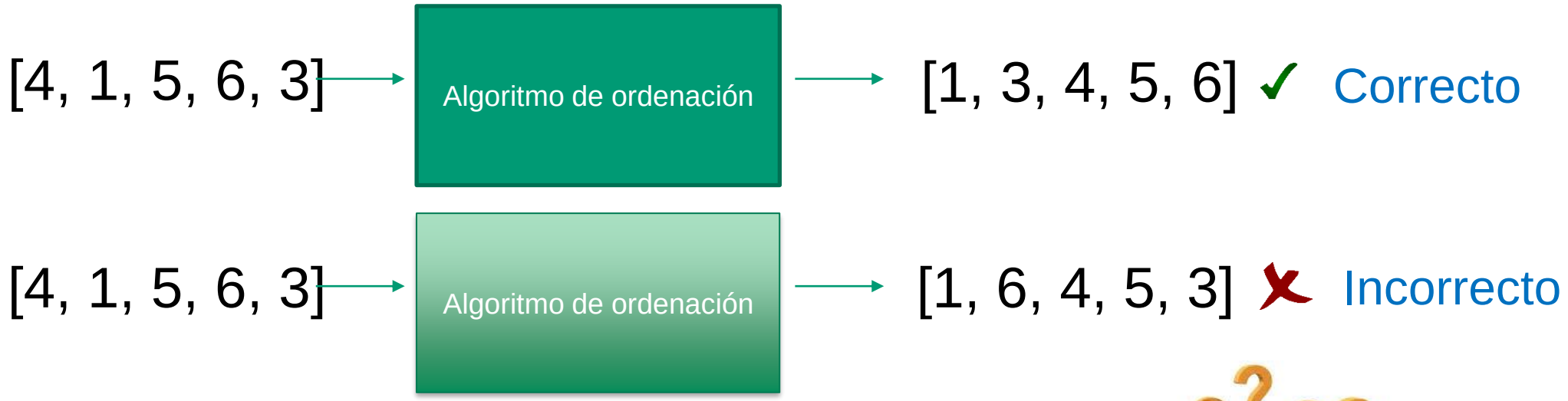
Complejidad Computacional

PUCP

Solución de problemas

Ejemplo de Problema: Ordenación

Primer objetivo: Diseñar algoritmos correctos



Al diseñar algoritmos, buscamos que sean correctos.

Eficiencia de los algoritmos

- ¿Y si hay varios algoritmos correctos para un mismo problema, cuál escogemos?
- Analizamos los recursos que consumen.

[4, 1, 5, 6, 3]

Algoritmo de ordenación 1

Num. Operaciones: 100

Cantidad de memoria: 1MB

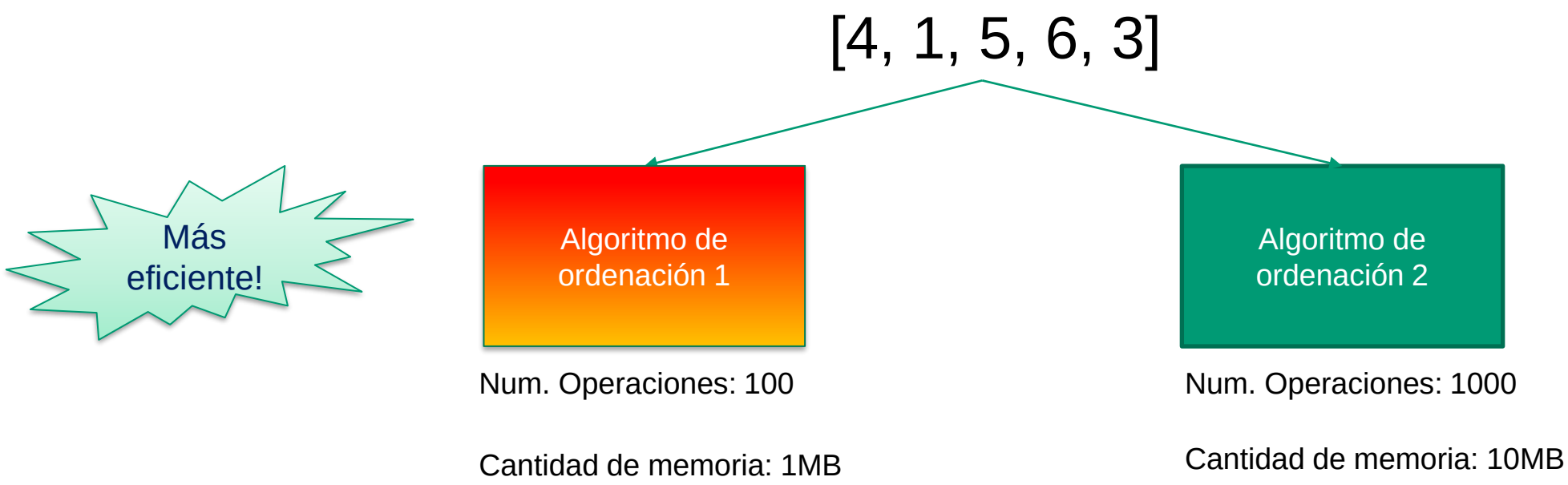
Algoritmo de ordenación 2

Num. Operaciones: 1000

Cantidad de memoria: 10MB

Eficiencia de los algoritmos

- ¿Y si hay varios algoritmos correctos para un mismo problema, cuál escogemos?
- Analizamos los recursos que consumen.



Recursos y eficiencia

Tiempo: La eficiencia de tiempo indica qué tan rápido se ejecuta un algoritmo para resolver un problema.

Espacio: La eficiencia de espacio nos indica la cantidad de memoria requerida (suma total del espacio que ocupan las variables del algoritmo) que necesita un algoritmo para su ejecución.

Medición del tiempo de ejecución

Idea: comparar dos algoritmos por el tiempo de ejecución que demoran para retornar un resultado

Hacemos un experimento: Corremos el mismo algoritmo de ordenación en estos dos equipos

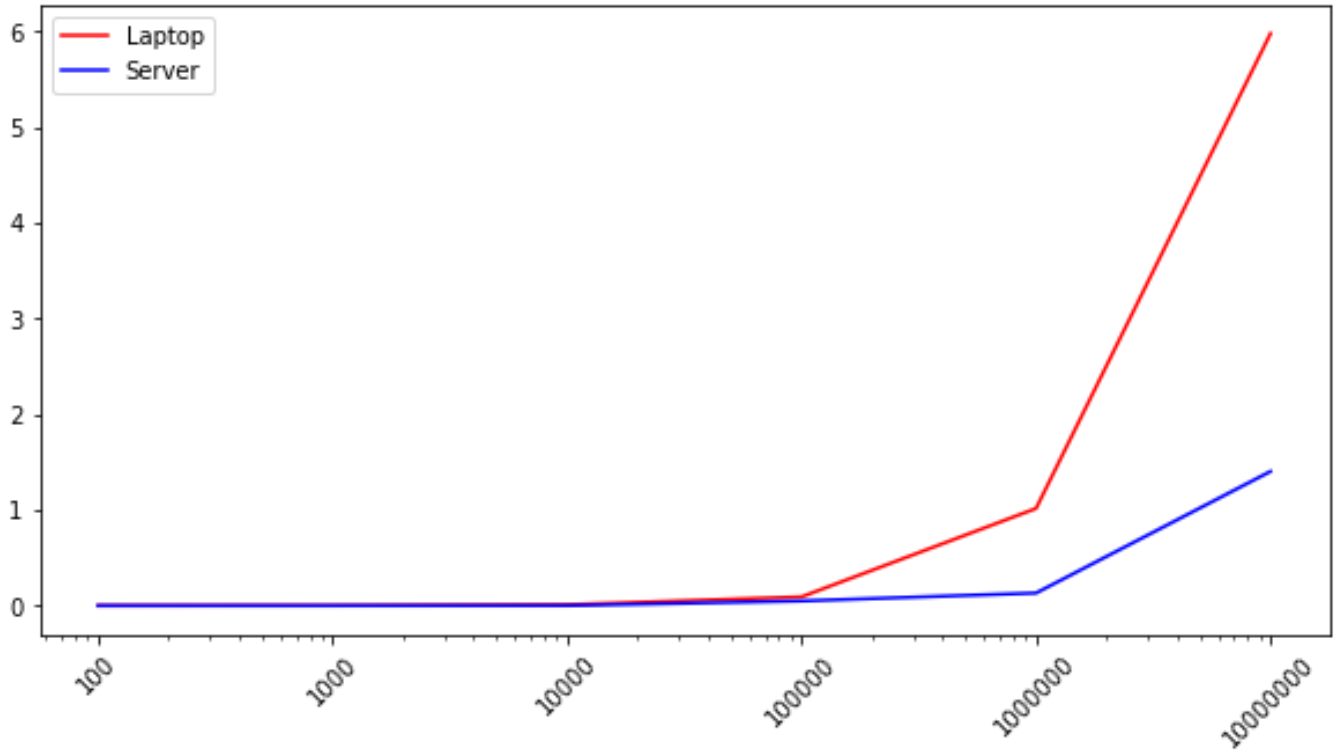
Equipo 1: Laptop
- CPU: Intel Core i7 ~ 2.5Ghz
- RAM: 12GB

Equipo 2: Server
- CPU: Intel Core i9 ~ 3.5Ghz
- RAM: 128GB

Calculamos el tiempo de ordenación para arreglos aleatorios de tamaños: $10^2, 10^3, 10^4, 10^5, 10^6, 10^7$

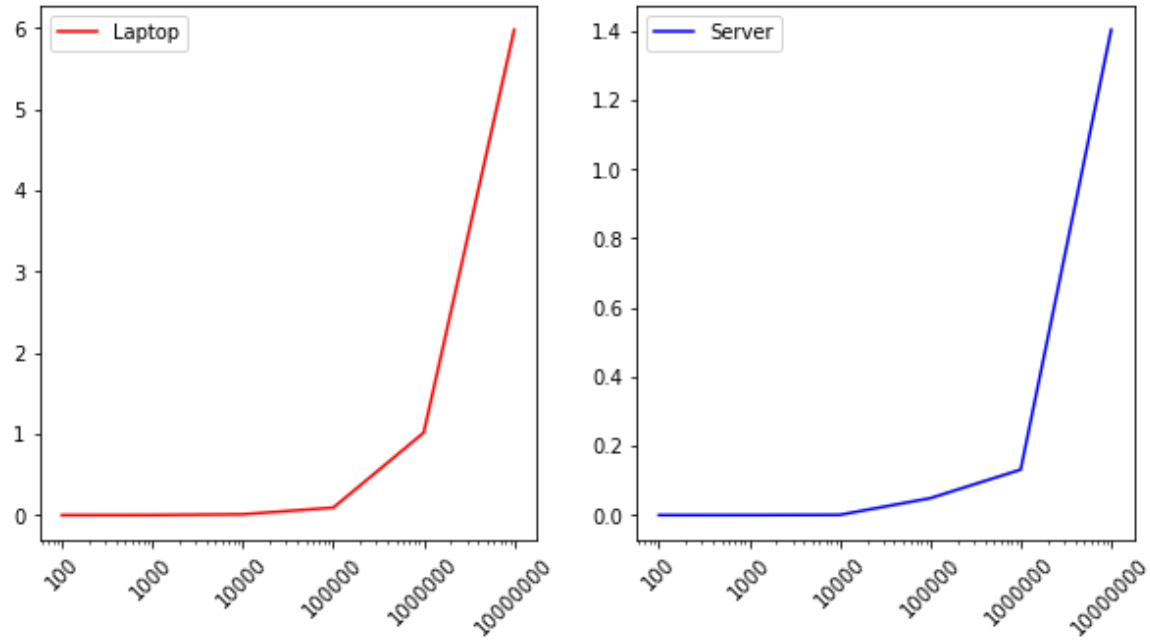
Medición del tiempo de ejecución

Resultado



Medición del tiempo de ejecución

Resultado



El comportamiento es el mismo, sólo que un computador es más rápido que el otro.

Es necesario analizar la eficiencia independientemente del equipo en donde se corre el algoritmo.

Medición del tiempo de ejecución

Medir la eficiencia como una función de la cantidad de operaciones que hace un algoritmo

Para entradas de tamaño n:

c_{op} : tiempo de ejecución de la operación básica de un algoritmo en una computadora en particular.

C (n) : cantidad de veces que se debe ejecutar esta operación para el algoritmo.

El tiempo de ejecución T (n):

$$T(n) \approx c_{op} C(n)$$

Medición del tiempo de ejecución

Si $$C(n) = \frac{1}{2} n(n-1),$$

¿Cuánto tiempo más se ejecutará el algoritmo si duplicamos su tamaño de entrada?

La respuesta es cuatro veces más larga:

$$C(n) = \frac{1}{2} n(n-1) = \frac{1}{2} n^2 - \frac{1}{2} n \approx \frac{1}{2} n^2$$

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op} C(2n)}{c_{op} C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2} n^2} = 4.$$

(Nota: no interesa el valor de c_{op}, ni el factor ½ .)

Tiempo de ejecución de un programa

¿Como se debe elegir entre varios algoritmos?

El algoritmo debe cumplir dos objetivos:

1. Que el algoritmo sea fácil de entender, codificar y depurar.

2. Que el algoritmo use eficientemente los recursos del computador y, en especial, que se ejecute con la mayor rapidez posible.

Tiempo de ejecución de un programa

Medición del tiempo de ejecución de un programa

El tiempo de ejecución depende de factores como:

1. Los **datos de entrada** al programa
2. La **calidad del código** generado por el compilador utilizado para crear el programa objeto
3. La naturaleza y rapidez de las **instrucciones de máquina empleadas** en la ejecución del programa
4. La **complejidad temporal** del algoritmo base del programa.

Tiempo de ejecución de un programa

¿Cómo medimos el tiempo de ejecución de un algoritmo?

- Mejor caso: En condiciones óptimas (no se usa por ser demasiado optimista).
- Peor caso: En el peor escenario posible (nos permite acotar el tiempo de ejecución).
- Caso promedio: Caso difícil de caracterizar en la práctica.

- En este caso se define **T(n)** como el tiempo de ejecución del peor caso, es decir, el máximo valor del tiempo de ejecución para entradas de tamaño **n**.

Soporte matemático para Complejidad Computacional

Soporte Matemático

El objetivo es establecer un orden relativo entre funciones.

Dadas dos funciones, por lo general hay puntos donde una función es menor que la otra, por lo que no tiene sentido afirmar que:

$$f(n) < g(n)$$

Así, se compararan sus *tasas de crecimiento relativas*.

Compare, por ejemplo, $\boldsymbol{f(n)} = 1000n$ y $\boldsymbol{g(n)} = n^2$.

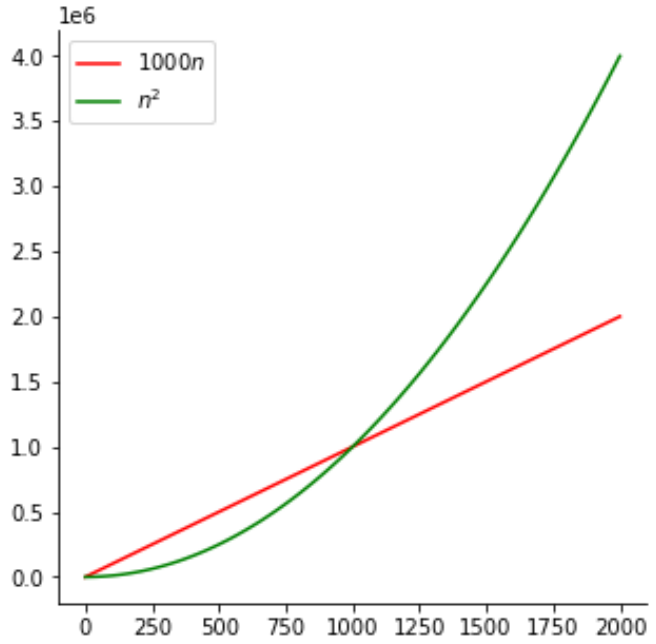
$1000n > n^2$?

$1000n < n^2$?

Soporte Matemático

$1000n > n^2$ para valores pequenos de n.

$1000n < n^2$ para $n > 1000$. El punto de cambio es $n = 1000$.



n^2 crece con una tasa mayor, y así n^2 finalmente será la función mayor.

Notación O grande

Definición: $T(n) = O(f(n))$ si existen una *constante* positiva c y un entero no negativo n_0 tales que $T(n) \leq cf(n)$ cuando $n \geq n_0$.

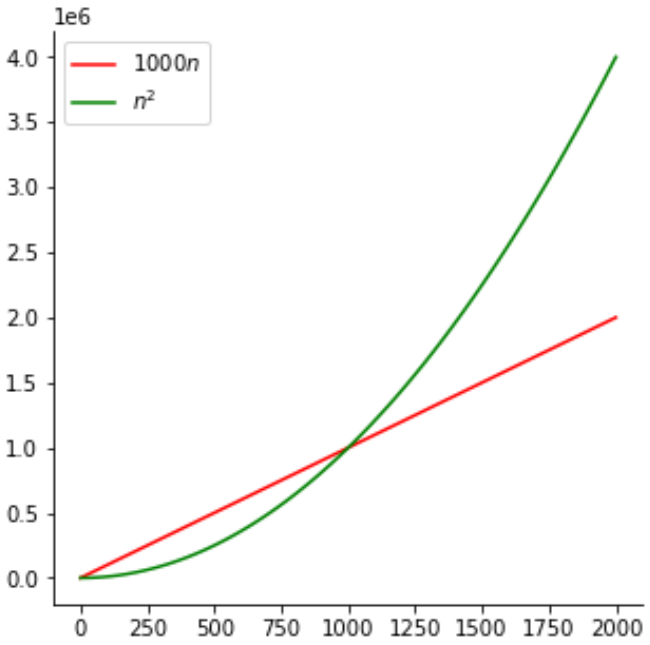
- $T(n) = 1000n$, $f(n) = n^2$, para $n_0 = 1000$, $c = 1$, o $n_0 = 10$, $c = 100$
- $1000n = O(n^2)$ (orden n cuadrada, también O grande n cuadrada)

$T(n) \leq cn^2$ cuando $n \geq n_0$.

La tasa de crecimiento de $T(n)$ es menor o igual que la de $f(n)$.

Mark Allen Weiss

Notación O grande



$T(n) = 1000n$

$f(n) = n^2$

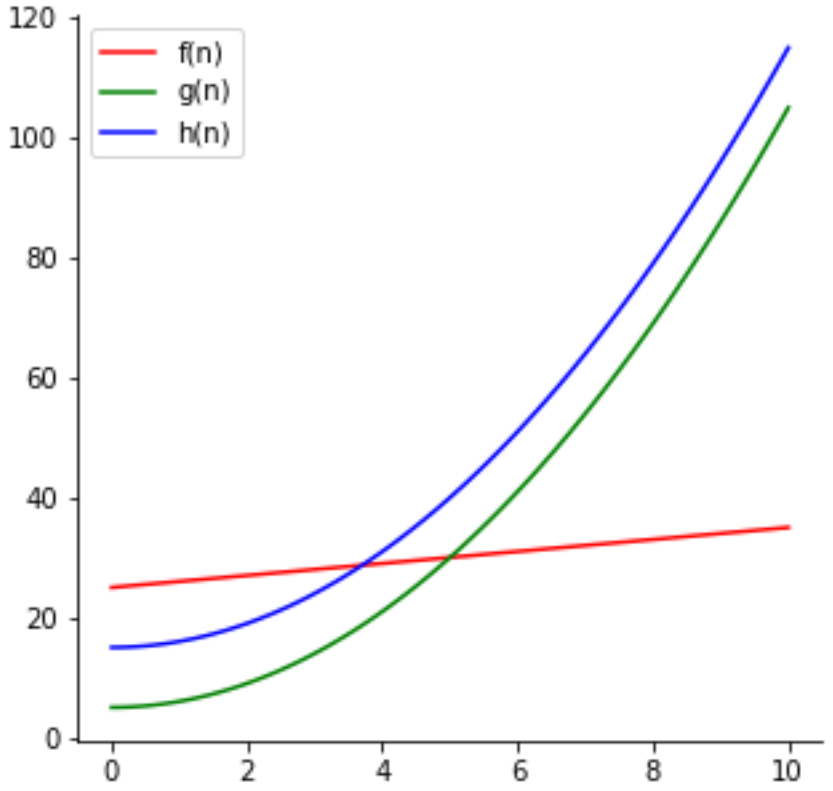
Decimos $T(n) = O(f(n))$ porque, dado un cierto punto, $T(n)$ crece más lento que $f(n)$

Se puede decir $T(n) = O(f(n))$ ó $T(n) \in O(f(n))$

Mark Allen Weiss

Soporte Matemático

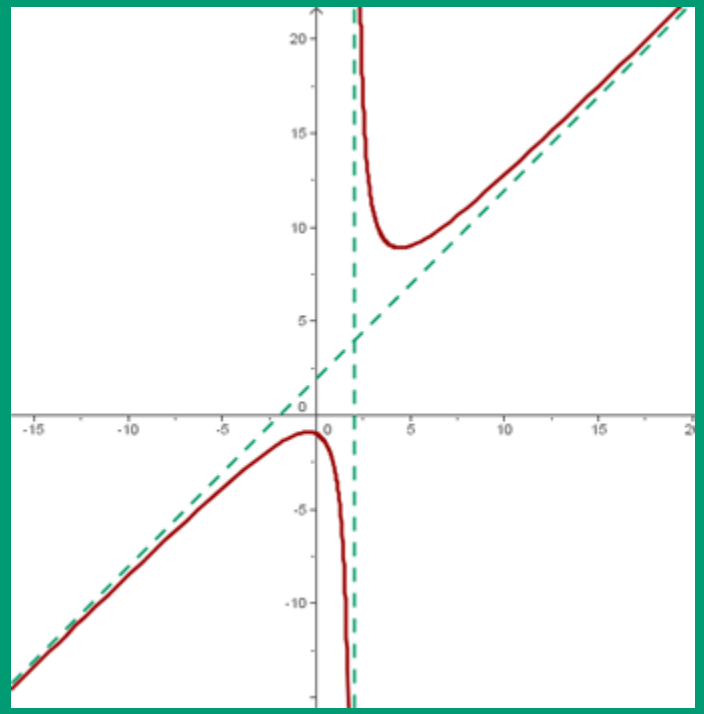
- $f(n) = n + 25$
- $g(n) = n^2 + 5$
- $h(n) = n^2 + 15$



Entonces podemos decir:

$$f(n) = O(g(n))$$

$$f(n) = O(h(n))$$



Asíntotas

- Para conocer que tan bueno es un algoritmo, necesitamos analizar su potencia, independientemente de las capacidades de la máquina que los ejecute e incluso de la habilidad del programador que los codifique.
- Este análisis nos interesa especialmente cuando el algoritmo se aplica a problemas grandes. Casi siempre los problemas pequeños se pueden resolver de cualquier forma, apareciendo las limitaciones al atacar problemas grandes. No debe olvidarse que cualquier técnica de ingeniería, si funciona, acaba aplicándose al problema más grande que sea posible: las tecnologías de éxito, antes o después, acaban llevándose al límite de sus posibilidades.
- Las consideraciones anteriores nos llevan a estudiar el comportamiento de un algoritmo cuando se fuerza el tamaño del problema al que se aplica. Matemáticamente hablando, cuando N tiende a infinito. Es decir, su comportamiento asintótico.

Análisis Asintótico

Estudia el comportamiento del algoritmo cuando el tamaño de las entradas es lo suficientemente grande, sin tener en cuenta lo que ocurre para entradas pequeñas y obviando factores constantes.

Analizar $T(n),\ n \to \infty$

Notación Asintótica

Informalmente, $O(f(n))$ es el conjunto de todas las funciones de crecimiento con un orden menor o igual a $f(n)$.

Por lo tanto, las siguientes afirmaciones son todas ciertas:

$$n \in O(n^2), \qquad 100n + 5 \in O(n^2), \qquad \tfrac{1}{2}n(n-1) \in O(n^2).$$

Por otra parte:

$$n^3 \notin O(n^2), \qquad 0.00001n^3 \notin O(n^2), \qquad n^4 + n + 1 \notin O(n^2).$$

Si un algoritmo se puede demostrar de un cierto orden O(...), es cierto que también pertenece a todos los órdenes superiores, pero en la práctica lo útil es encontrar la menor orden de complejidad que lo cubra.

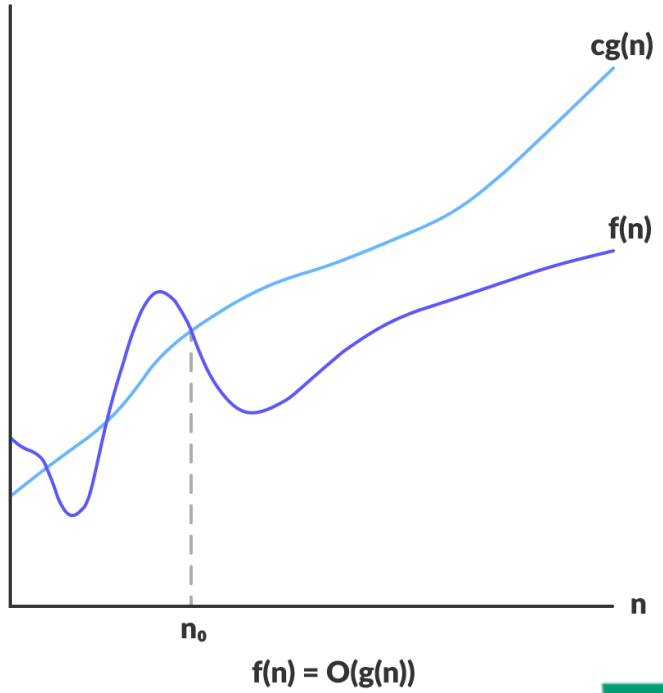
Notación Asintótica *O*

Podemos definirlo matemáticamente como:

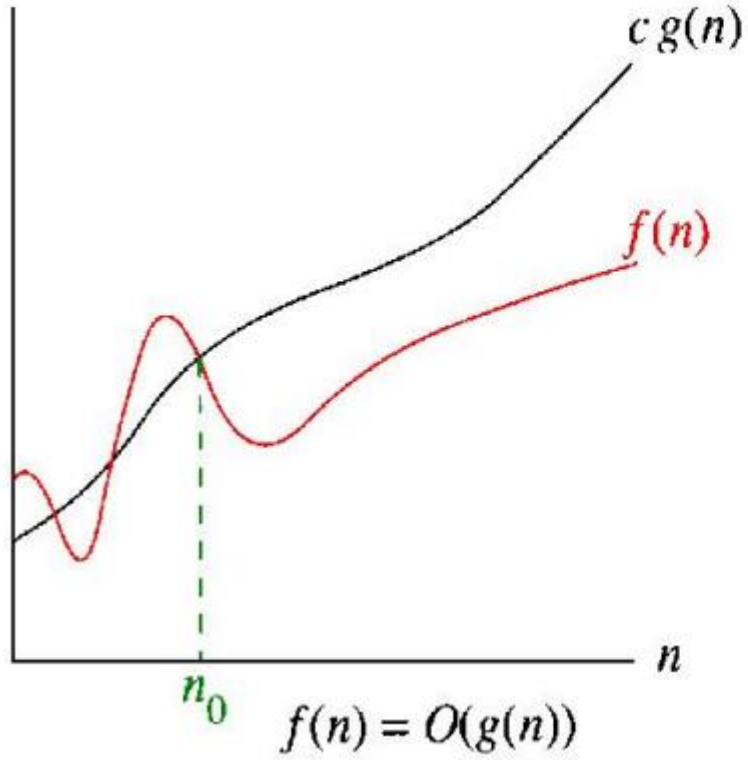
$$O(g(n)) = \{f(n) : \exists c \in \mathbf{R}^+, \exists n_0 \in \mathbf{N}, \text{tal que } f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$$

- Determina una cota superior en la tasa de crecimiento de una función, dentro de un factor constante
- Ejemplos: $6n^3 \in O(n^3)$ ya que se cumple la definición con c = 6

Crédito: Programiz



Notación Asintótica *O*



Ejemplos:

- $300n^2 \in O(n^2)$
- $5n^4 - 4n^3 + 10n^2 + 39 \in O(n^4)$
- $\log_b n \in O(\log_a n), \forall a, b$
- $2^n \in O(n!)$
- $500000n \in O(0{,}00001n^2)$
- $0{,}000001n^2 \notin O(500000n)$
- $n! \notin O(2^n)$

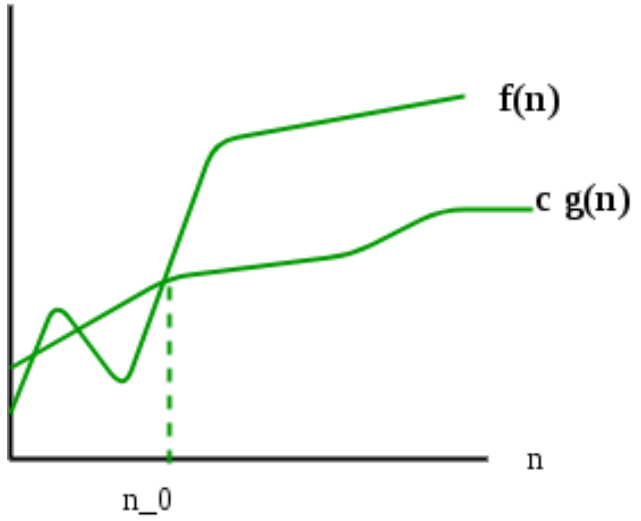
Anany Levitin

Notación Asintótica Ω

Podemos definirlo matemáticamente como:

$$\Omega(g(n)) = \{f(n) : \exists c \in \mathbf{R}^+, \exists n_0 \in \mathbf{N}, \text{tal que } f(n) \geq cg(n) \text{ para todo } n \geq n_0\}$$

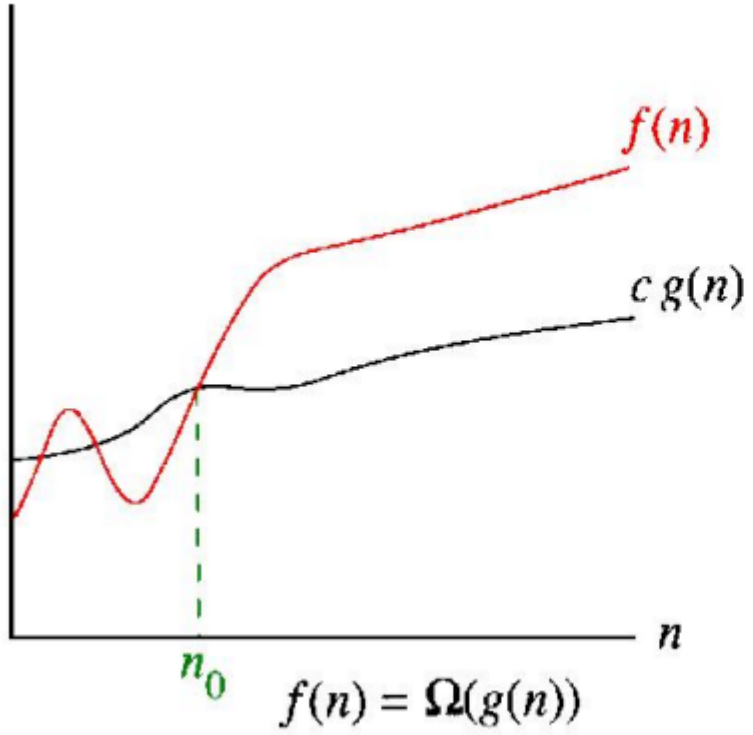
Crédito: GeeksforGeeks



$f(n) = \Omega(g(n))$

- Determina una cota inferior en la tasa de crecimiento de una función, dentro de un factor constante
- Ejemplos: $6n^3 \in \Omega(n^3)$ ya que se cumple la definición con c = 1

Notación Asintótica Ω



Ejemplos:

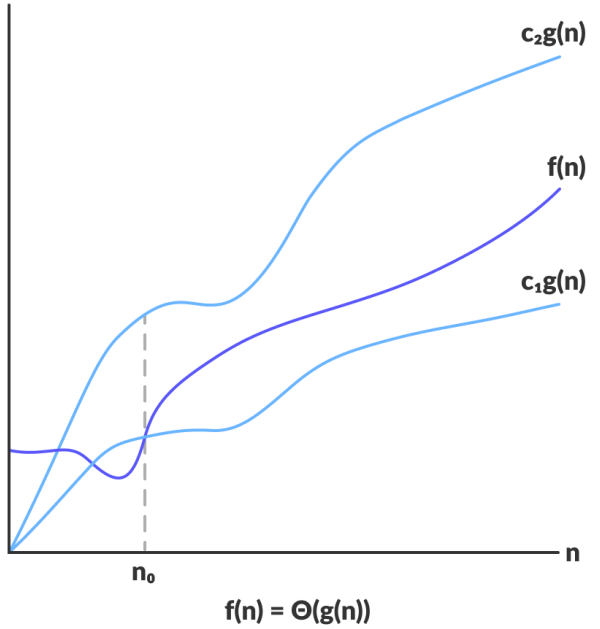
- $3n^5 + 4n^3 - 8n^2 + 10n \in \Omega(n^4)$
- $\log_b n \in \Omega(\log_a n), \forall a, b$
- $n! \in \Omega(2^n)$
- $0{,}00001n^2 \in \Omega(50000n)$
- $50000n \notin \Omega(0{,}00001n^2)$
- $2^n \notin \Omega(n!)$

Anany Levitin

Notación Asintótica Θ

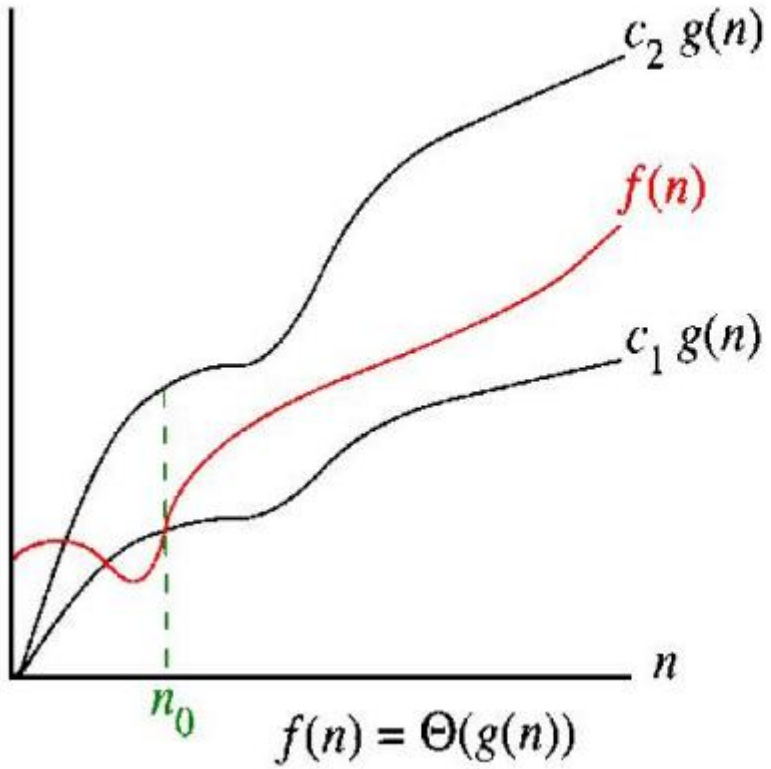
Podemos definirlo matemáticamente como:

$$\Theta(g(n)) = \{f(n) : \exists c, d \in \mathbf{R}^+, \exists n_0 \in \mathbf{N}, \text{ tal que } cg(n) \leq f(n) \leq dg(n) \text{ para todo } n \geq n_0\}$$



- Determina una cota superior e inferior en la tasa de crecimiento de una función, dentro de un factor constante
- Ejemplos: $6n^3 \in \Theta(n^3)$ ya que se cumple la definición con c = 6, d = 6

Notación Asintótica Θ



Ejemplos:

- $3n^2 \in \Theta(n^2)$
- $\log n \notin \Theta(n)$
- $2^{n+1} \in \Theta(2^n)$
- $500000n^2 \in \Theta(0{,}00001n^2)$
- $\log_b n \in \Theta(\log_a n)$ para todo $a, b > 0$

Anany Levitin

Notación Asintótica *O*

Por ejemplo, ¿es cierto que?

(i) $n^2 \in O(n^3)$

(ii) $n^3 \in O(n^2)$

La forma general de proceder es la siguiente:

- Supongamos que la afirmación es verdadera.
- Trabajar a partir de la definición de '*O*', e intentar encontrar valores adecuados de c y n_0
- Si puede encontrar un par de valores, la afirmación es cierta. Si hay alguna razón fundamental por la que no se puede encontrar un par c y n_0, entonces la hipótesis original era **errónea** y la afirmación es **falsa**.

Notación Asintótica *O*

¿Es $n^2 \in O(n^3)$?

Supongamos que es. Entonces:

$n^2 - cn^3 \leq 0$, para todos $n \geq n_0$

$\Rightarrow n^2 (1 - cn) \leq 0$, para todos $n \geq n_0$

$\Rightarrow cn \geq 1$, para todos $n \geq n_0$

$\Rightarrow n \geq 1 / c$, para todos $n \geq n_0$

Al elegir (por ejemplo) $c = 2$, $n_0 = 1$ es satisfactorio y por lo que es **VERDADERO** que $n^2 \in O(n^3)$.

Notación Asintótica *O*

¿Es $n^3 \in O(n^2)$?

Una vez más, supongamos que es. Entonces:

$n^3 - cn^2 \leq 0$, para todos $n \geq n_0$

$\Rightarrow n^2 (n - c) \leq 0$, para todos $n \geq n_0$

$\Rightarrow n - c \leq 0$, para todos $n \geq n_0$

Pero c tiene que tener un valor fijo. No hay forma de satisfacer $n \leq c$, para todos $n \geq n_0$ para una c fija.

De ahí que el supuesto original es **FALSO**, y $n^3 \notin O(n^2)$.

Notación Asintótica *O*

Cuando se dice que T(n) = O(f(n)), se esta garantizando que la función T(n) crece a una velocidad no mayor que f(n); así f(n) es una cota superior de T(n).

Como esto implica que f(n) = Ω(T(n)), se dice que T(n) es una cota inferior de f(n).

Por ejemplo:

n^3 crece mas rápido que n^2, así se puede decir que $n^2 = O(n^3)$ o $n^3 = \Omega(n^2)$, $f(n) = n^2$ y $g(n) = 2n^2$ crecen a la misma velocidad, así que ambas, $f(n) = O(g(n))$ y $f(n) = \Omega(g(n))$, se cumplen.

Notación Asintótica *O*

Cuando dos funciones crecen a la misma velocidad, la decisión de representar esto o no con $\Theta()$ puede depender del contexto particular.

Intuitivamente, si $g(n) = 2n^2$, entonces $g(n) = O(n^4)$, $g(n) = O(n^3)$ y $g(n) = O(n^2)$, son técnicamente correctas, pero es obvio que la ultima opción es la mejor respuesta.

Escribir $g(n) = \Theta(n^2)$ dice no solo que $g(n) = O(n^2)$, sino también que el resultado es tan bueno (exacto) como es posible.

Mark Allen Weiss

Notación Asintótica *O*

Las cosas importantes a saber son:

Regla 1: Si $T_1(n) = O(f(n))$ y $T_2(n) = O(g(n))$, entonces
(a) $T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$,
(b) $T_1(n) * T_2(n) = O(f(n)*g(n))$.

Regla 2: Si $T(x)$ es un polinomio de grado n, entonces $T(x) = \Theta(x^n)$.

Regla 3: $\log^k n = O(n)$ para cualquier k constante. Esto indica que los logaritmos crecen muy lentamente.

Las demostraciones de estas reglas se dejan como ejercicio.

Mark Allen Weiss

Soporte matemático

Tasas de crecimiento características

Función	Nombre	Ejemplo
c	constante	sumando 2 números
$\log n$	logarítmica	búsqueda en árbol binario balanceado
$\log^2 n$	logarítmica cuadrada	
n	lineal	búsqueda en lista
$n \log n$	linearítmica	*mergesort, quicksort*
n^2	cuadrática	Insertion/Selection/Bubble Sort
n^3	cúbica	multiplying 2 n x n matrices
2^n	exponencial	

Soporte matemático

Tasas de crecimiento características

n	1	2	4	16	256	4096	65536
$\ln n$	0	1	2	4	8	12	16
$n \ln n$	0	2	8	64	2048	49152	1048576
n^2	1	4	16	256	65536	16777216	$4.295*10^9$
n^3	1	8	64	4096	16777216	$6.872*10^{10}$	$2.815*10^{14}$
2^n	2	4	16	65536	$1.16*10^{77}$	$> 10^{1232}$	Inmenso

Mark Allen Weiss

Soporte matemático

$O(k * f(n)) = O(f(n))$, para cualquier constante k.

Esto se debe a que la multiplicación por una constante solo corresponde a un reajuste del valor de la constante arbitraria k en la definición de "*O*".

Esto significa que bajo notación O, podemos olvidar factores constantes. Aunque estas "constantes ocultas" podrían ser importantes en la práctica, No cambie el orden del resultado.

Tenga en cuenta que, como consecuencia de esto, desde $\log_a n = \log_a b * \log_b n$ existe no hay diferencia efectiva entre las bases logarítmicas bajo notación *O*; convencionalmente solo usamos *O* (log n), olvidando la base (irrelevante).

Mark Allen Weiss

Soporte matemático

Hay varios puntos que destacar.

Primero, es muy mal estilo incluir constantes o términos de orden menor en una O grande. No se debe decir $T(n) = O(2n^2)$ o $T(n) = O(n^2+n)$. En ambos casos, la forma correcta es $T(n) = O(n^2)$. Esto significa que en cualquier análisis que requiera una repuesta O grande, todos los tipos de simplificaciones son posibles. Por lo regular pueden ignorarse los términos de orden menor y desecharse las constantes. La precisión que se requiere en estos casos es considerablemente menor.

En segundo lugar, siempre se pueden determinar las tasas de crecimiento relativo de dos funciones $f(n)$ y $g(n)$ mediante el cálculo $\lim_{n\to\infty} f(n)/g(n)$, usando la regla de L'Hôpital si es necesario.

Mark Allen Weiss

Resumen

- Crecimiento de funciones
- Notaciones O, Ω y Θ
- Propiedades y operaciones

Bibliografía

- LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición. USA: Pearson, 2012. ISBN 0-13-231681-1.
- Mark Allen Weiss. **Estructuras de Datos y Algoritmos, Addison-Wesley Iberoamericana,** 1995, pp. 17-44.
- Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, **Estructuras de Datos y Algoritmos, Addison-Wesley Iberoamericana, 1988,** pp. 16-27

PUCP