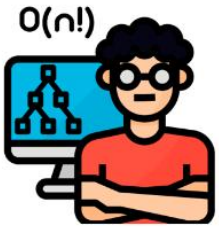
Instituto Politécnico Nacional

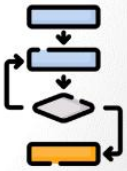
Escuela Superior de Cómputo



Análisis de algoritmos

Tema 04: Notación asintótica

Prof. Edgardo Adrián Franco Martínez
http://eafranco.com
edfrancom@ipn.mx
@edfrancom edfrancom



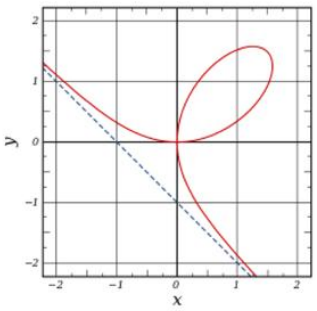
Introducción

- El **costo para obtener una solución** de un problema concreto **aumenta con el tamaño n** del problema.
- Para valores suficientemente pequeños de n, el coste de ejecución de cualquier algoritmo es pequeño, incluyendo los algoritmos ineficientes; i.e. la elección del algoritmo no es un problema crítico para problemas de pequeño tamaño.
- **El análisis de algoritmos se realiza para valores grandes de n**. Para lo cual se considera el comportamiento de sus funciones de complejidad para valores grandes de n, es decir se estudia el **comportamiento asintótico** de $f(n)$, lo cuál permite conocer el comportamiento en el límite del coste cuando n crece.

Asíntota

- Se le llama **asíntota** a una línea recta que se aproxima continuamente a otra función o curva; es decir que la distancia entre las dos tiende a cero, a medida que se extienden indefinidamente.

- También se puede decir que es la curva la que se aproxima continuamente a la recta; o que ambas presentan un **comportamiento asintótico**.



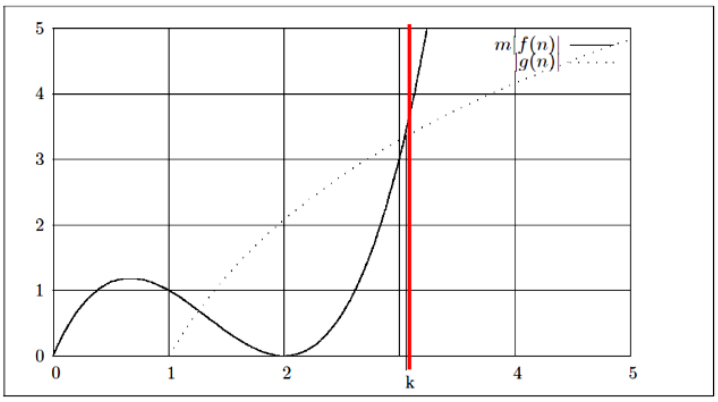
Dominio asintótico

- Sean f y g funciones de $\mathbb{N}$ a $\mathbb{R}$. Se dice que f domina asintóticamente a g o que g es dominada asintóticamente por f; si $\exists k \geq 0$ y $m \geq 0$ tales que:

$$|g(n)| \leq m|f(n)|\,, \forall n \geq k$$

- En otros términos, podemos decir que si una función domina a otra, su velocidad de crecimiento es mayor o igual.

- Puesto que las **funciones complejidad** son funciones con dominio $\mathbb{N}$ $(números\ naturales)$, y contra dominio $\mathbb{R}$ $(números\ reales)$; los conceptos y las propiedades de **dominio asintótico** proporcionan una manera conveniente de expresarlas y manipularlas.



- Gráficas de las funciones $m|f(n)|$ y $|g(n)|$,donde k es el valor a partir del cual $m|f(n)|$ es mayor que $|g(n)|$ y esta relación de magnitud se conserva conforme n crece.

Dominio asintótico (Ejemplo 1)

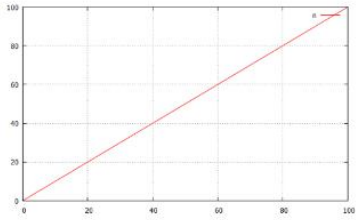
- **Ejemplo 1:** Sean $f(n) = n$ y $g(n) = n^3$ funciones de $\mathbb{N}$ a $\mathbb{R}$.

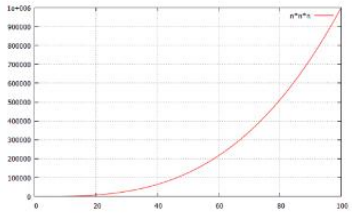
i. Demostrar que g domina asintóticamente a f

$$\exists m \geq 0, k \geq 0 \; tales \; que \; |f(n)| \leq m|g(n)| \, , \forall n \geq k$$

ii. Demostrar que f no domina asintóticamente a g

$$\neg(\exists m \geq 0, k \geq 0 \; tales \; que \; |g(n)| \leq m|f(n)| \, , \forall n \geq k)$$





- **Ejemplo 1:** Sean $f(n) = n$ y $g(n) = n^3$ funciones de $\mathbb{N}$ a $\mathbb{R}$.

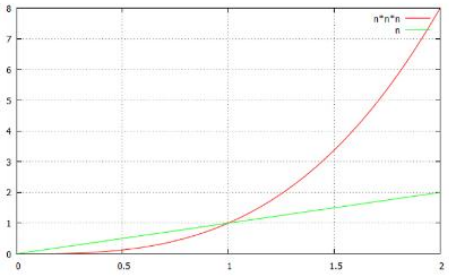
i. Demostrar que g domina asintóticamente a f

$$\exists m \geq 0, k \geq 0 \; tales \; que \; |f(n)| \leq m|g(n)| \, , \forall n \geq k$$

- Substituyendo $f(n)$ y $g(n)$

$$|n| \leq \mathrm{m}|n^3|, \forall n \geq k$$

- Si se toman $m = 1$ y $k = 1$, las desigualdades anteriores se cumplen, por lo tanto, m y k existen, y en consecuencia g domina asintóticamente a f.



- **Ejemplo 1:** Sean $f(n) = n$ y $g(n) = n^3$ funciones de $\mathbb{N}$ a $\mathbb{R}$.

ii. Demostrar que f no domina asintóticamente a g

$$\neg(\exists m \geq 0, k \geq 0 \; tales \; que \; |g(n)| \leq m|f(n)| \, , \forall n \geq k)$$

- Aplicando la negación se tiene

$$\forall m \geq 0, k \geq 0, \exists n \; \geq k \; tal \; que \; |g(n)| > m|f(n)|$$

- Sustituyendo g y f en cada lado de la desigualdad

$$|n^3| > m|n|$$

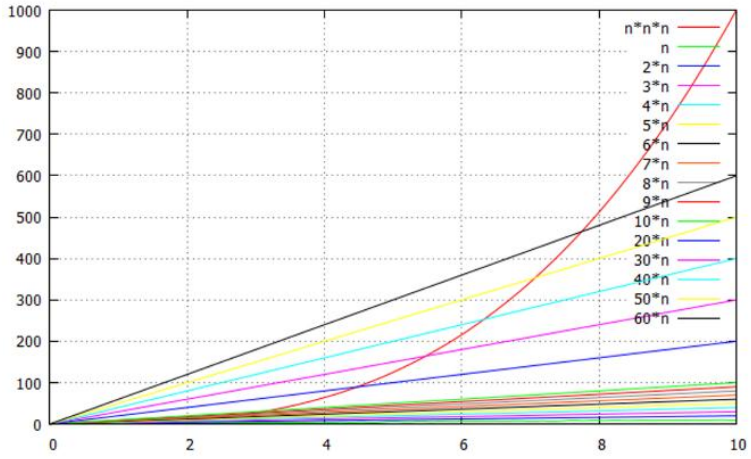
- Simplificando

$$|n^2| > m$$
$$n^2 > m$$
$$\boldsymbol{n > \sqrt{m}} \quad \text{y} \quad \mathbf{n \geq k}$$

Si se toma $\boldsymbol{n > \max(\sqrt{m}, k)}$ ambas desigualdades se cumplen para toda $m \geq 0 \; y \; k \geq 0, \therefore$

$f \; no \; domina \; asitóticamente \; a \; g$

- **Ejemplo 1:** Sean $f(n) = n$ y $g(n) = n^3$ funciones de $\mathbb{N}$ a $\mathbb{R}$.



Dominio asintótico (Ejemplo 2)

- **Ejemplo 2:** Sea $g(n)$ una función de $\mathbb{N}$ a $\mathbb{R}$ y $f(n) = cg(n)$ $con\ c > 0\ y\ c \in \mathbb{R}$.

i. Demostrar que f d. a. g

$$\exists m \geq 0, k \geq 0\ tales\ que\ |g(n)| \leq m|cg(n)|\ , \forall n \geq k$$

ii. Demostrar que g d. a. f

$$\exists m \geq 0, k \geq 0\ tales\ que\ |cg(n)| \leq m|g(n)|\ , \forall n \geq k$$

- **Ejemplo 2:** Sea g una función de $\mathbb{N}$ a $\mathbb{R}$ y$f(n) = cg(n)\ con\ c > 0\ y\ c \in \mathbb{R}$.

i. Demostrar que f d. a. g

$$\exists m \geq 0, k \geq 0\ tales\ que\ |g(n)| \leq m|cg(n)|\ , \forall n \geq k$$

- Pero $|ab| = |a||b|$ por lo tanto

$$|g(n)| \leq |m||c||g(n)|\ , \forall n \geq k$$

- Como $c > 0$; entonces

$$|g(n)| \leq mc|g(n)|\ , \forall n \geq k$$

- Tomando $m = \frac{1}{c}$ y $k = 0$ se tiene

$$|g(n)| \leq |g(n)|, \forall n \geq 0 \therefore f\ d.a.g$$

- **Ejemplo 2:** Sea g una función de $\mathbb{N}$ a $\mathbb{R}$ y$f(n) = cg(n)\ con\ c > 0\ y\ c \in \mathbb{R}$.

i. Demostrar que g d. a. f

$$\exists m \geq 0, k \geq 0\ tales\ que\ |cg(n)| \leq m|g(n)|\ , \forall n \geq k$$

- Esto es

$$c|g(n)| \leq m|g(n)|\ , \forall n \geq k$$

- Tomando $\boldsymbol{m = c\ y\ k = 0}$ se tiene

$$c|g(n)| \leq c|g(n)|\ , \forall n \geq 0 \therefore g\ d.a.f$$

Dominio asintótico a la función complejidad

- Cuando se hace el **análisis teórico** para obtener la **función complejidad** $f(n)$ que caracterice a un algoritmo, se está obteniendo un **modelo de comportamiento** para la demanda de recursos en función del parámetro n; de tal forma que si $t(n)$ es la cantidad real del recurso que se consume para una implantación específica del algoritmo se tiene que:

$$t(n)\alpha\, f(n)$$
$$t(n) = cf(n)$$
$$|t(n)| \leq c|f(n)|$$

- i.e. **$f(n)$ domina asintóticamente a cualquier $t(n)$;** dicho de otra manera la **demanda de recursos** se va a regir por el **modelo de crecimiento** que observe $f(n)$.

Notación asintótica

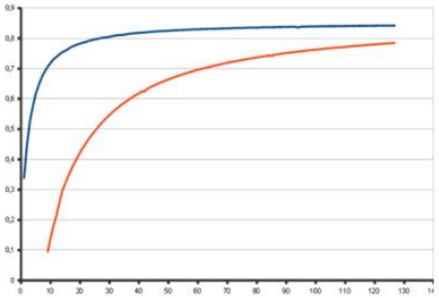
- El interés principal del análisis de algoritmos radica en saber cómo crece la demanda de recursos, cuando el tamaño del problema crece. Esto es la ***eficiencia asintótica*** del algoritmo. Se denomina "asintótica" porque analiza el comportamiento de las funciones en el *límite*, es decir, su **tasa de crecimiento**.

- La **notación asintótica** captura el **comportamiento** de la función para **valores grandes de n. Se consideran las funciones asintóticamente no negativas.**

- Las notaciones no son dependientes de los tres casos anteriormente vistos, es por eso que **una notación que determine el peor caso** puede estar presente en una o en todas las situaciones.

Notación de orden

- Cuando describimos cómo es que el número de operaciones $f(n)$ depende del tamaño n; lo que nos interesa es encontrar el patrón de crecimiento o cota para la función complejidad y así caracterizar al algoritmo; una vez hecha esta caracterización podremos agrupar las funciones de acuerdo al número de operaciones que realizan.



Cota Superior: Notación O mayúscula

- La notación O (Omicron mayúscula) se utiliza para comparar funciones. Dada una función f, se desea estudiar funciones g que a lo sumo crezcan tan deprisa como f.

- **Definición:** Sean f y g funciones de $\mathbb{N}$ a $\mathbb{R}$. Si existen constantes c y x_0 tales que:

$$\forall x > x_0, |f(x)| \leq c\,|g(x)|$$

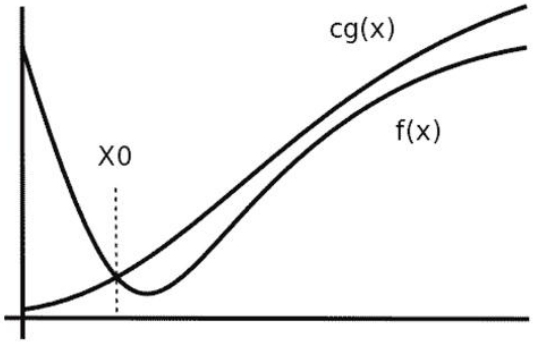
- i.e. que para $x > x_0$, f es menor o igual a un múltiplo c de g, decimos que:

$$f(x) = O(g(x))$$

- La definición formal es:

$$\boldsymbol{f(x) = O(g(x))} \Leftrightarrow \exists c > 0, x_0 > 0 \mid \forall x > x_0, |f(x)| \leq c|g(x)|$$

- $\boldsymbol{f(x) = O(g(x)) \Leftrightarrow \exists c > 0, x_0 > 0 \mid \forall x > x_0, \mid f(x) \mid \leq c \mid g(x) \mid}$



$$O(g(x)) = \left\{ \begin{array}{l} f(x) : \text{existen } c, x_0 > 0 \text{ tales que} \\ \forall x \geq x_0 : 0 \leq |f(x)| \leq c|g(x)| \end{array} \right\}$$

- **Ejemplo 3:** Muestre que $f_t(n) = 3n^3 + 5n^2 + 9 = O(n^3)$
  - Partiendo de:

$$\boldsymbol{f(x) = O(g(x)) \Leftrightarrow \exists c > 0, x_0 > 0 \mid \forall x > x_0, \mid f(x) \mid \leq c \mid g(x) \mid}$$

  - Sustituyendo

$$|3n^3 + 5n^2 + 9| \leq c|n^3|$$

  - Como ambas funciones van de $\mathbb{N}$ a x y x_0>0, y desde x_0=0 no negativa

$$3n^3 + 5n^2 + 9 \leq cn^3$$

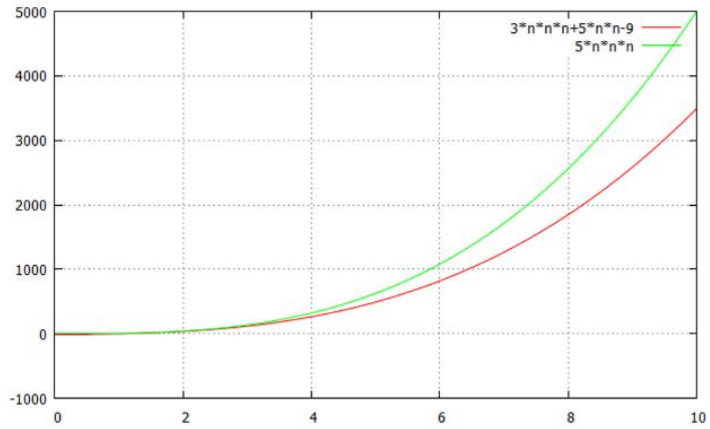
  - Agrupando

$$5n^2 \leq (c-3)n^3 - 9$$

  - **Si $\boldsymbol{c = 5}$ y $\boldsymbol{x_0 = 3}$, $\boldsymbol{\forall n > x_0}$**

$$5n^2 \leq 2n^3 - 9 \ \therefore \ 3n^3 + 5n^2 + 9 = O(n^3)$$

- **Ejemplo 3:** $5n^2 \leq 2n^3 - 9 \ \therefore \ 3n^3 + 5n^2 + 9 = O(n^3)$



Cota Superior no ajustada: Notación *o* minuscula

- La cota superior asintótica dada por la notación **O** puede o no ser ajustada asintóticamente. La cota $2n^2 = O(n^2)$ es ajustada asintóticamente, pero la cota $2n^2 \neq o(n^2)$ **no lo es**. Se emplea la notación ***o*** para denotar una cota superior que no es ajustada asintóticamente.

- **Definición:** Sean f y g funciones de $\mathbb{N}$ a $\mathbb{R}$. Si para toda constante $c > 0$ y una constante x_0 se cumple que:

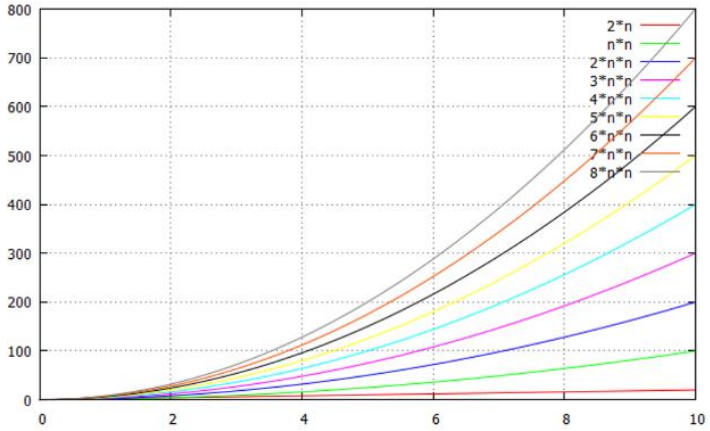
$$\forall x > x_0, c > 0, |f(x)| \leq c\,|g(x)|$$

- i.e. que para $x > x_0$, f es menor o igual a todos los múltiplos $c > 0$ de g, decimos que:

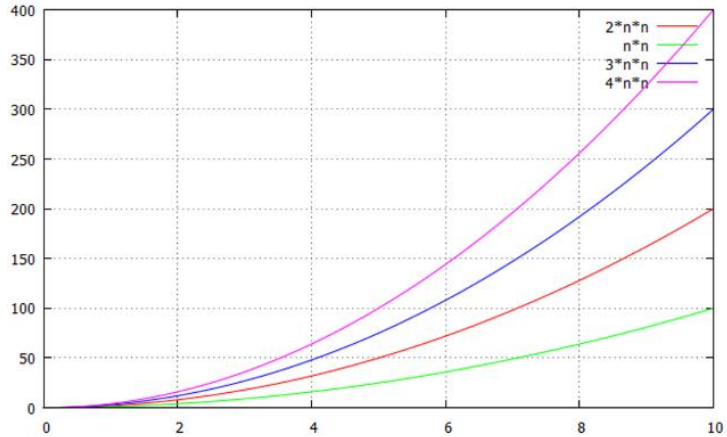
$$f(x) = o(g(x))$$

- La definición formal es:
- $\boldsymbol{f(x) = o(g(x))} \Leftrightarrow \exists x_0 > 0 \mid \forall x > x_0, c > 0, \mid f(x) \mid \leq c \mid g(x) \mid$

$$2n = o(n^2)$$



$$2n^2 \neq o(n^2)$$



Diferencia entre O y o

- Las notaciones de $\boldsymbol{O}$ y $\boldsymbol{o}$ son similares. La diferencia principal es, que en $f(n) = \boldsymbol{O}(g(n))$, la cota $0 \leq f(n) \leq cg(n)$ se cumple para *alguna* constante $c > 0$, pero en $f(n) = \boldsymbol{o}(g(n))$, la cota $0 \leq f(n) \leq cg(n)$ se cumple para ***todas*** las constantes $c > 0$.

- Intuitivamente en la notación $\boldsymbol{o}$, la función $f(n)$ se vuelve insignificante con respecto a $g(n)$ a medida que n se acerca a infinito

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0$$

- Para o la desigualdad se mantiene para todas las constantes positivas, mientras que para O la desigualdad se mantiene sólo para algunas constantes positivas.

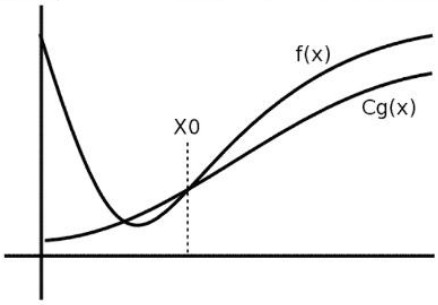
$$f(x) = O(g(x)) \Leftrightarrow \exists c > 0, x_0 > 0 \mid \forall x > x_0, |f(x)| \leq c|g(x)|$$

$$f(x) = o(g(x)) \Leftrightarrow \exists x_0 > 0 \mid \forall x > x_0, c > 0, |f(x)| \leq c|g(x)|$$

Cota Inferior: Notación Ω

- La notación Ω Es el reverso de O, i.e es una función que sirve de **cota inferior** de otra función cuando el argumento tiende a infinito

$$f(x) = \Omega(g(x)) \Leftrightarrow \exists c > 0, x_0 > 0 \mid \forall x > x_0, c|g(x)| \leq |f(x)|$$



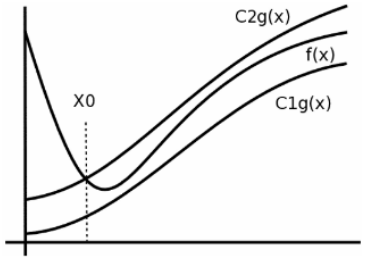
$$\Omega(g(x)) = \left\{ \begin{array}{c} f(x) : \text{existen } c, x_0 \text{ constantes positivas tales que} \\ \forall x : x_0 \leq x : 0 \leq cg(x) \leq f(x) \end{array} \right\}$$

Cota ajustada asintótica: Notación Θ

- La **cota ajustada asintótica o de orden exacto** es una función que sirve de cota tanto superior como inferior de otra función cuando el argumento tiende a infinito.

$$f(x) = \Theta(g(x)) \Leftrightarrow \exists c_1 > 0, c_2 > 0, x_0 > 0 \mid \forall x > x_0, c_1 |g(x)| \leq |f(x)| \leq c_2 |g(x)|$$

Θ Grande dice que ambas funciones se dominan mutuamente, en otras palabras, son asintóticamente equivalentes.



$$f(x) = \Theta(g(x)) \text{ si y solo si } f(x) = O(g(x)) \text{ y } f(x) = \Omega(g(x))$$

$$\Theta(g(x)) = \left\{ \begin{array}{c} f(x) : \text{existen } c_1, c_2, x_0 \text{ constantes positivas tales que} \\ \forall x : x_0 \leq x : 0 \leq c_1 g(x) \leq f(x) \leq c_2 g(x) \end{array} \right\}$$

Observaciones sobre las cotas asintóticas

1. La utilización de las cotas asintóticas para comparar funciones de tiempo de ejecución se basa en la hipótesis de que son suficientes para decidir el mejor algoritmo, prescindiendo de las constantes de proporcionalidad. Sin embargo, esta hipótesis puede no ser cierta cuando el tamaño de la entrada es pequeño.

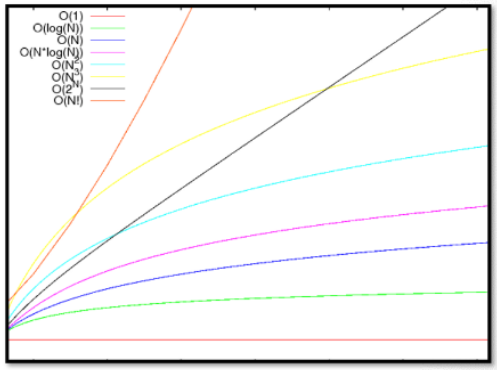
2. Para un algoritmo dado se pueden obtener tres funciones que miden su tiempo de ejecución, que corresponden a sus casos mejor, medio y peor, y que denominaremos respectivamente $T_m(n)$, $T_{1/2}(n)$ y $T_p(n)$, para cada una de ellas podemos dar hasta 4 cotas asintóticas (O, o, Ω, Θ) de crecimiento, por lo que se obtiene un total de 12 cotas para el algoritmo.

3. Para simplificar, dado un algoritmo diremos que su orden de complejidad es $O(f)$ si su tiempo de ejecución para el peor caso es de orden O de f, es decir, $T_p(n)$ es de orden $O(f)$. De forma análoga diremos que su orden de complejidad para el mejor caso es $\Omega(g)$ si su tiempo de ejecución para el mejor caso es de orden Ω de g, es decir, $T_m(n)$, es de orden $\Omega(g)$.

4. Por último, diremos que un algoritmo es de orden exacto $\Theta(f)$ si su tiempo de ejecución en el caso medio $T_{1/2}(n)$ es de este orden.

Ordenes de complejidad (Cota superior)

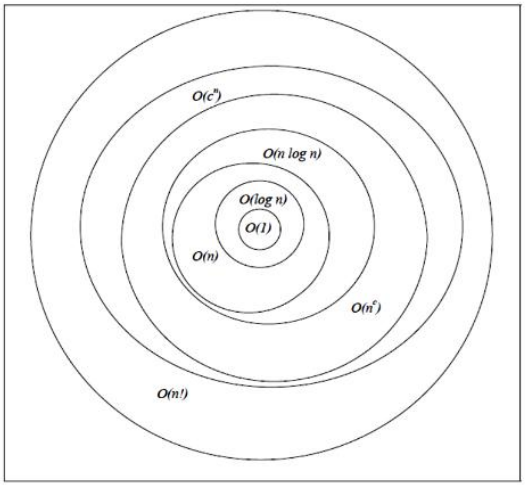
- Dado que las funciones complejidad están en el conjunto de funciones que van de de $\mathbb{N}$ a $\mathbb{R}$; es posible clasificar los algoritmos según el orden de su función complejidad. Gran parte de los algoritmos tienen complejidad que cae en uno de los siguientes casos:



- $O(1)$ **Complejidad constante**
- $O(\log n)$ **Complejidad logarítmica**
- $O(n)$ **Complejidad lineal**
- $O(n \log n)$ **Complejidad "n log n"**
- $O(n^2)$ **Complejidad cuadrática**
- $O(n^3)$ **Complejidad cubica**
- $O(c^n)$; $c > 1$ **Complejidad exponencial**
- $O(n!)$ **Complejidad factorial**

$$O(1) \subset O(log\, n) \subset O(n) \subset O(n\, log\, n) \subset O(n^2) \subset O(c^n) \subset O(n!)$$

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(c^n) \subset O(n!)$$



$f(n) = O(1)$ Complejidad constante

- Los algoritmos de complejidad constante ejecutan siempre el mismo número de pasos sin importar cuan grande es n.

$f(n) = O(\log n)$ Complejidad logarítmica

- Los algoritmos de complejidad logarítmica, habitualmente son algoritmos que resuelven un problema transformándolo en problemas menores.

$f(n) = O(n)$ Complejidad lineal

- Los algoritmos de complejidad lineal generalmente tratan de manera constante cada n del problema por lo que si n dobla su tamaño el algoritmo también dobla el número de pasos.

$f(n) = O(n \log n)$ Complejidad "n log n"

- Los algoritmos de complejidad "n log n" generalmente dividen un problema en problemas más sencillos de resolver para finalmente combinar las soluciones obtenidas.

$f(n) = O(n^2)$ Complejidad cuadrática

- Los algoritmos de complejidad cuadrática aparecen cuando los datos se procesan por parejas, en la mayoría de los casos en bucles anidados.

$f(n) = O(n^3)$ Complejidad cubica

- Los algoritmos de complejidad cubica son útiles para resolver problemas pequeños p.g. si n=100 el número de operaciones es de 1,000,000.

$O(c^n);\ c > 1$ Complejidad exponencial

- Los algoritmos de complejidad exponencial no son útiles desde el punto de vista práctico, aparecen cuando un problema se soluciona empleando fuerza bruta.

$O(n!)$ Complejidad factorial

- Un algoritmo de complejidad factorial generalmente aparece cuando el problema también es resuelto por fuerza bruta y es un problema complejo por definición; o cuando se maneje de mala manera un algoritmo recursivo.

Función de coste	Tamaño n 10	20	30	40	50	60
n	0,00001 seg.	0,00002 seg.	0,00003 seg.	0,00004 seg.	0,00005 seg.	0,00006 seg.
n^2	0,0001 seg.	0,0004 seg.	0,0009 seg.	0,0016 seg.	0,0035 seg.	0,0036 seg.
n^3	0,001 seg.	0,008 seg.	0,027 seg.	0,064 seg.	0,125 seg.	0,316 seg.
n^5	0,1 seg.	3,2 seg.	24,3 seg.	1,7 min.	5,2 min.	13 min.
2^n	0,001 seg.	1 seg.	17,9 min.	12,7 días	35,7 años	366 siglos
3^n	0,059 seg.	58 min.	6,5 años	3855 siglos	10^8 siglos	10^{13} siglos