

La Eficiencia tiene dos factores:

- Tamaño del problema.
- Función usada. (Búsquedas extrañas)

Vamos a usar una composición de funciones usando

$\Theta(g(u))$, la cual nos permite comprender con más facilidad las funciones de código,

Consejo

Estar funciones $u^2, u^3 \circ 2^u$ donde u es el tamaño del problema

Algoritmos vs Implementaciones

Conjunto de posos fríos que nos llevan a resolver un problema

Realización de un algoritmo en un lenguaje de programación determinado

Aquí nos centramos nosotros

Hay dos formas de evaluar la eficiencia de un código.

→ Empírica: Tomamos una talla de datos y tracemos un ajuste. Necesitamos la implementación del propio algoritmo y probarlo con muchos casos.

- Inconvenientes:
- Implementar y probar el algoritmo
 - Experimentar sobre un conjunto fruto de entradas, que pueden no ser significativas.
 - Para comparar, el entorno de comparación debe ser el mismo.

→ Algorítmico: Nos bastan de uestro análisis para conocer su eficiencia.

- Ventajas:
- No necesita implementación
 - Es teórico y global.

Familias de órdenes de ejecución

Hay varias y un problema particular a una familia $f(u)$ cuando el tiempo de ejecución t pudiese ser acotado por una constante c y $f(u) \leq t \leq c f(u)$.

- Tipos:
- lineal $\equiv u$
 - cuadrática $\equiv u^2$
 - polinómica $\equiv u^k / k \in \mathbb{N}$
 - logarítmica $\equiv \log_u(u) / k \in \mathbb{N}$
 - exponencial $\equiv c^u / c \in \mathbb{R}$

Queremos trabajar con familias de funciones. La función
○ notación $\Theta(u)$ lo único que hace es simplificar $f(u)$
para facilitar su clasificación (clase de equivalencia)

Comparación de órdenes de eficiencia

- La determinación de la eficiencia de un algoritmo depende de su orden de eficiencia cuando $n \rightarrow \infty$
- Comparamos perfiles de crecimiento, de manera que un algoritmo es más eficiente si su perfil de crecimiento es menor.
- Condiciones para la comparación:
 - El resultado no puede depender del resultado para un n finito de valores de la prueba.
 - Se refiere a que por muy amplia que sea la prueba puede fijar valores concretos que representen las correspondientes clases
 - El resultado no puede depender de las fijaciones concretas que representen las correspondientes clases

Notación O

Sirve para poder indicar el grado de eficiencia estudiando sólo el algoritmo.

$O(g(u))$ simplemente trata de quedarnos con el elemento más pesado de la función. Formalmente, acotamos $g(u)$ por otra función cuyo grado sea el mismo y sea más fácil de interpretar.

Pasos

1. Análisis del tiempo de ejecución
2. Análisis de eficiencia ⇒ clasificar.

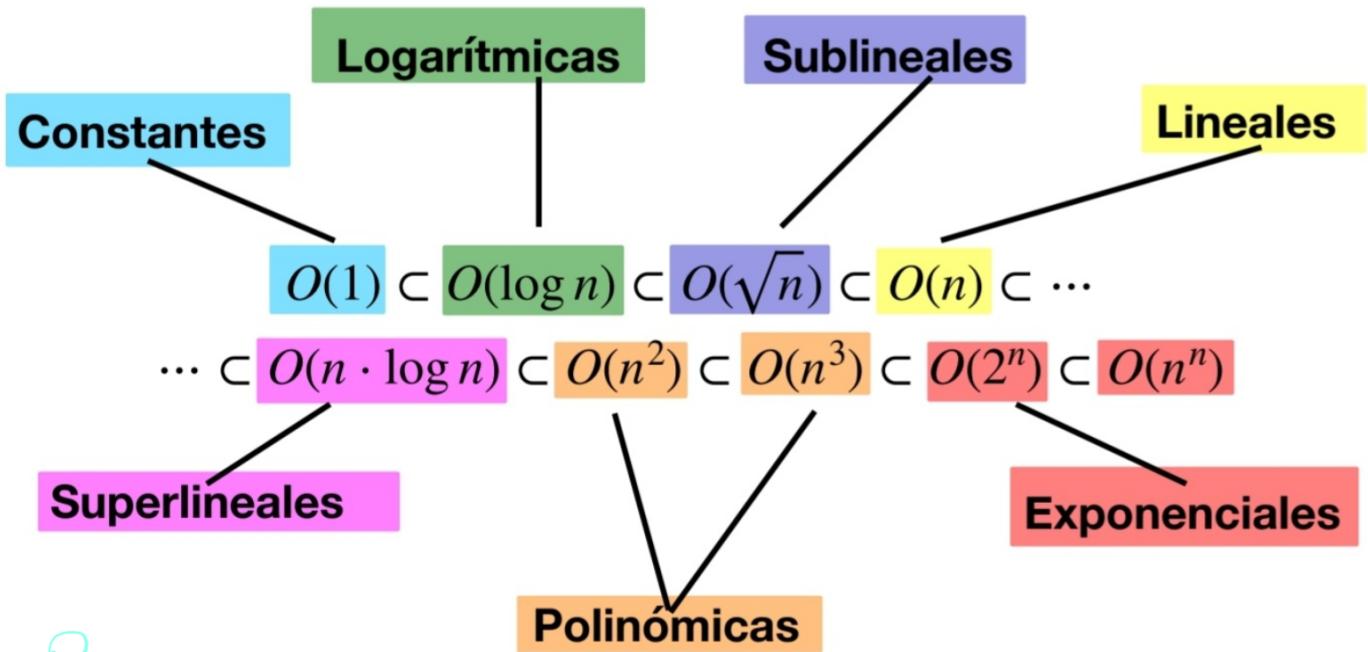
Definición rigurosa

Sean $g(u)$ y $g(u)$ dos funciones, se dice que $g(u)$ es $O(g(u))$ si: $g(u) \leq c g(u) / c \in \mathbb{R}$ $\forall u \geq u_0$ con c y u_0 constantes

Lo que buscamos es un supremo

Jerarquía de funciones

Se aprende
usando



Pensar en
cálculo

Elección del mejor algoritmo

Además de la eficiencia, debemos tener en cuenta otros factores:

- Tamaño de los problemas a resolver.
- Requerimientos de espacio y tiempo del sistema.
- Complejidad de implementación y mantenimiento de los algoritmos.

Algunas reglas simples

Transitividad si $g(u) \in O(g(u))$ y $g(u) \in O(h(u)) \Rightarrow g(u) \in O(h(u))$.

Polinomios de menor grado

Jerarquía de funciones (mirar diapositiva)

Bases y potencias de logaritmos pueden ignorarse.

Bases y potencias de exponentes no pueden ignorarse

Regla de la suma Me que lo con lo
peor si tengo suma de varias
funciones.

Regla del producto Consiste en
sumar el n^o de veces y multiplicar
por el nivel de eficiencia de
cada iteración.

Cálculo de la eficiencia

- Operación elemental: operación de un algoritmo cuya tiempo de ejecución puede ser acotado por una constante.

Interesa	No interesa
No de op. elementales	Saber la ejecución exacta que indica el tiempo de ejecución.

- Estructura secuencial: Si tengo los códigos consecutivos, me basta con hacer $\Theta(\text{suma de ambos funciones})$.
- Estructura condicional: Siempre y cuando la condición sea $\Theta(1)$, tomaremos como tiempo de ejecución el $\Theta(g(n))$ que sea más costoso entre el "if" y el "else".

En caso contrario, tomaremos como inicio, la parte de la condición, si alguna parte la supera, esta última será la eficiencia.

○ Estructura iterativa: Debemos tener en cuenta el tiempo usado en la inicialización, la evaluación de la condición y la actualización si procede

- Bucle for: $O(Ini(n)) + O(Con(n)) + O(Ite(n)) \cdot [O(Cu(n)) + O(Inc(n)) + O(Con(n))]$
- Bucle while: $O(Con(n)) + O(Ite(n)) \cdot [O(Cu(n)) + O(Con(n))]$
- Bucle do-while: $O(Ite(n)) \cdot [O(Cu(n)) + O(Con(n))]$

donde:

$O(Ini(n))$: Inicialización

$O(Con(n))$: Condición

$O(Ite(n))$: Iteración

$O(Cu(n))$: Cuerpo

$O(Inc(n))$: Incremento

Aquí ocurre igual que en la estructura condicional con la condición.

Además, en caso de que haya los condicionales, tomaremos el peor camino posible para así encontrar ese supuesto $O(\text{plus})$.

○ Funciones: Consiste en evaluar el código de la función y obtener su eficiencia. La llamada y el paso por referencia son $O(1)$. Si es por copia y los

clases no son primitivas, no es $O(1)$ y debemos calcular su coste.

- En particular:

- Las asignaciones con llamadas a función deben sumar el tiempo de ejecución de cada llamada
- En las condiciones e incrementos de bucles con llamada se deberá multiplicar el tiempo de la llamada por las iteraciones del bucle
- La inicialización de bucles o la condición de sentencias condicionales con llamadas deben sumar el tiempo de ejecución de la llamada al tiempo total del bucle o del condicional

Progressión Geométrica

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1}$$

Progressión Aritmética

$$\sum_{i=0}^n q_i = a + \frac{an + q_0}{2}$$

