

① LECCION 2: EFICIENCIA. NOTACIÓN 0-GRANDE

• EFICIENCIA: Qué recursos gasta un algoritmo

→ Podemos catalogar un algoritmo

• RECURSOS: MEMORIA Y TIEMPO de EJECUCIÓN

• EJEMPLO: Asignación de

• 50 trabajadores a 50 trabajos

- ¿Mejor asignación?

- Búsqueda Masiva - Mira todas las posibilidades

• $50! \approx 10^{64}$

• Ordenador:

- Evalúa 1 billon de posibilidad/s
- Empezo 15000 millones de años
- Aún están procesando

• No basta con la mejora de hardware

• ALGORITMO: Conjunto de pasos o sentencias bien ordenadas que resuelven un problema.

• IMPLEMENTACIÓN: Traducción de las sentencias a un determinado lenguaje de programación

• PRINCIPIO DE INVARIANZA: dos implementaciones de un mismo algoritmo solo se diferencia en una constante multiplicativa.

• ¿Qué algoritmo escogemos?

- El más eficiente en recursos.

• EJEMPLO: Para un problema

• Dos algoritmos A_1 y A_2

• Eficiencia de A_1 : $T_1(n) = 10^{-4} \times 2^n$

• Eficiencia de A_2 : $T_2(n) = 10^{-2} \times n^3$

• n es el nº de datos de entrada al algoritmo.

n	10	20	30	38
$T_1(n)$	0.4s	2m	>1 día	1 año
$T_2(n)$	10s	80s ≈ 3m	270s	549s

• ANÁLISIS ASINTÓTICO

Análisis de la eficiencia de un algoritmo cuando el nº de datos de entrada tiende a valores muy grandes ($n \rightarrow \infty$)

• En la eficiencia de un algoritmo la estructura de datos que escogamos también es relevante.

FUNCIÓN DE EFICIENCIA:-

Se define sobre el nº de datos de entrada del algoritmo $f(n)$ (n : nº de datos de entrada o talla) y devuelve un valor real mayor o igual que 0 midiendo la eficiencia (tiempo o memoria)

$$f: \mathbb{N} \rightarrow \mathbb{R}_0^+$$

$n \rightarrow f(n)$

tiempo de ejecución (s)

espacio (bytes)

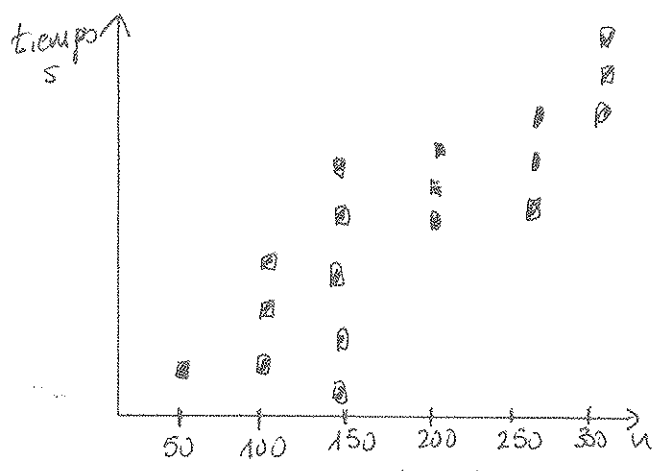
n : tamaño del problema o talla

②

LECCIÓN 2: continuación

- ¿Cómo calcular la eficiencia?
 - Teórica - Estudio Asintótico
 - Empírica - Experimentalmente

EMPIRICA



- Ejecutamos el algoritmo una vez implementado y obtenemos el tiempo:
 - Para un mismo n (nº de datos de entrada) lo ejecutamos varias veces
 - Lo ejecutamos para diferentes n (distintas tallas)

En C++ : como obtener el tiempo

```
#include <time>
;
```

```
time_t tantes, tdespues;
```

```
time (&tantes)
```

```
//sentencias
```

```
time (&tdespues)
```

```
cout << diff time (tdespues, tantes)
```

PROBLEMAS de la E. EMPÍRICA

- 1) Los datos que hayamos escogido no son representativos de las características del problema
- 2) Depende del hardware y librería usada
- 3) Para poder comparar dos algoritmos debemos ejecutarlos en las mismas condiciones.

E. TEORICA — Análisis asintótico

CARACTERÍSTICAS

- Independiente del hardware y software
- Representa a todas las entradas
- Usa una descripción a alto nivel del programa.

Orden de Eficiencia, $T(n)$

Un algoritmo tiene un orden de eficiencia $T(n)$, si \exists una implementación del algoritmo y su tiempo de ejecución $f(n)$ está acotado superiormente por $c \cdot T(n)$ siendo c una constante $c \geq 1$ y n el tamaño del problema.

$$f(n) \leq c \cdot T(n)$$

Ordenes de Eficiencia más comunes

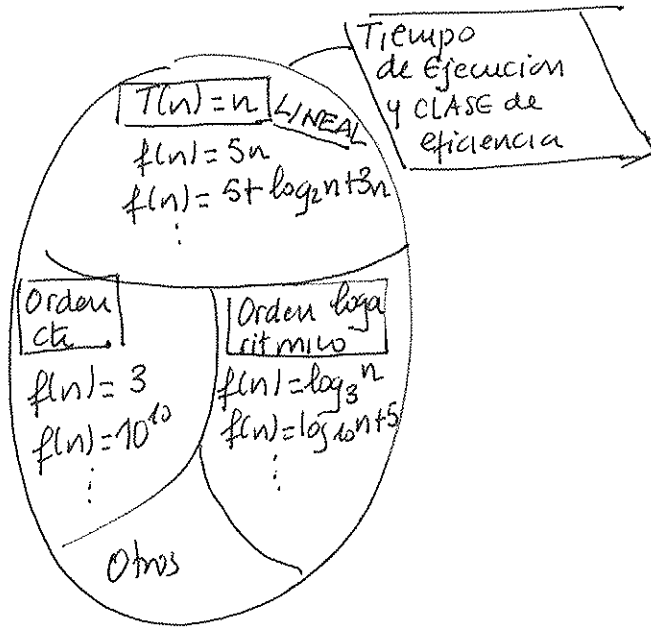
- Orden lineal — $T(n) = n$
- Orden cte — $T(n) = a$ a es una cte
- Orden logarítmico — $T(n) = \log(n)$
- Orden cuadrático — $T(n) = n^2$
- Orden exponencial — $T(n) = c^n$ c es una cte

3

LECCION 2: continuaci3n

E. TEORICA de UN ALGORITMO

- 1) Descubrir la funci3n de eficiencia $f(n)$ del algoritmo
- 2) Dada $f(n)$ deducir cual es su orden de eficiencia $T(n)$



EJEMPLO 2: Orden parcial

$$f(n) = \begin{cases} n^2 & \text{si } n \text{ es par} \\ n^4 & \text{si } n \text{ es impar} \end{cases}$$

$$g(n) = n^3$$

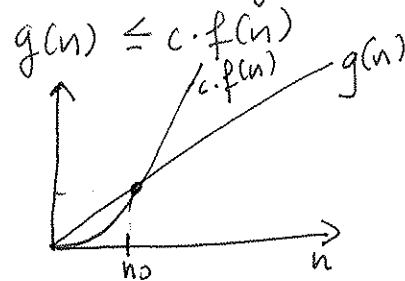
¿Qui3n es mayor $f(n)$ o $g(n)$?

NOTACION ASINTOTICA

$$5n^2 + 3n + 10 \approx n^2$$

DEFINICION: O-grande (tiempo de ejecuci3n en el peor caso) ej: Búsqueda secuencial.

Decimos que un algoritmo con tiempo de ejecuci3n $g(n) \in O(f(n))$ si $\exists c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tq $\forall n \geq n_0$



Ejemplos

p.e

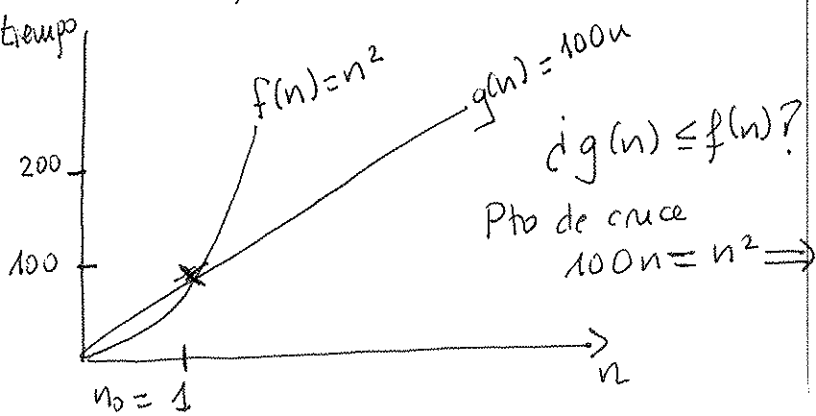
- $n^2 + 5n \in O(n^2)$ $c=2$ $n_0=5$
- $3n^3 + 2n^2 \in O(n^3)$ $c=4$ $n_0=2$
- $3^n \notin O(2^n)$

COMPARACI3N DE EFICIENCIA ENTRE DOS ALGORITMOS

CARACTERISTICAS

- Debe \exists independencia de lo que ocurra en un n3 finito de valores para n (tama3o del problema)
- Independiente de la funci3n que representa al orden de eficiencia.

COMPARAR \neq los ordenes de eficiencia
= comparar los perfiles de crecimiento



$$g(n) \leq c \cdot f(n) \Rightarrow n^2 - 100n = 0 \Rightarrow n = 100$$

④

LECCION 2 continuaci3n

Ejemplos:

- $10n^2 \log_2 n + 5n^2 + 3 \in O(n^2 \log_2 n)$
- $5 \cdot 2^n + 3n \in O(2^n)$

CLASES ESPECIALES de Ordenes de EFICIENCIA

- Constante $O(1)$
- Logaritmico $O(\log_2(n))$
- Cuadrático $O(n^2)$ LINEAL $O(n)$
- Polinomial $O(n^k)$ $k > 1$
- Exponencial $O(a^n)$ $n > 1$ $a > 1$

$$O(1) \subseteq O(\log_2(n)) \subseteq O(n) \subseteq O(n^2) \subseteq O(n^k) \subseteq O(a^n) \quad k > 2$$

MAS CLASES

$$O(1) \subset O(\log_2(n)) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log_2 n) \subset O(n^2) \subset O(n^k) \subset O(2^n) \subset O(n!)$$

REGLAS de O-GRANDE

1. $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$
entonces $f(n) \in O(h(n))$
2. $a_n n^d + a_{n-1} n^{d-1} + \dots + a_1 n + a_0 \in O(n^d)$
3. La base de los logaritmos no importa
 $\log_a(n) \in O(\log_b(n))$
4. El exponente en los logaritmos no importa
 $\log_a(n) \in O(\log_b(n))$
5. La base y exponente en los exponenciales si importa
 $3^n \notin O(2^n)$
 $a^{n^2} \notin O(a^n)$

PECULIARIDADES de O-Grande

Ejemplo

$$t_1(n) = 100n \leftarrow A_1$$

$$t_2(n) = \frac{n^2}{5} \leftarrow A_2$$

¿Cual escogeriamos?

- Donde se cruzan

$$100n = \frac{c \cdot n^2}{5}$$

$$n \left(\frac{n \cdot c}{5} - 100 \right) = 0 \Rightarrow n = \frac{100 \cdot 5}{c}$$

$$\text{con } c=1 \Rightarrow n=500$$

cruzan.

Luego el algoritmo A_1 se comporta peor hasta un tamaño de 500 (tiene peor tiempo).

• Si tus muestras no superan el tamaño de 500 puede interesarte ejecutar A_2 .

• Debemos tambien estudiar el espacio (memoria) que requiere cada algoritmo.

• Coste de la implementaci3n y mantenimiento.

5

LECCIÓN 25

EJERCICIO

Ordenar las siguientes funciones de acuerdo a su velocidad de crecimiento.

$n, \sqrt{n}, \log n, \log \log n, \log^2 n, n/\log n, \sqrt{n} \log^2 n, (1/3)^n, (3/2)^n, 17, n^2$

SOLUCIÓN

$$O\left(\left(\frac{1}{3}\right)^n\right) \subset O(17) \subset O(\log \log n) \subset O(\log n) \subset O(\log^2 n)$$

$$\subset O(\sqrt{n}) \subset O(\sqrt{n} \log^2 n) \subset O(n/\log n) \subset O(n)$$

$$\subset O(n^2) \subset O\left(\left(\frac{3}{2}\right)^n\right)$$

*Para comprobar que $O(f) \subset O(g)$ basta ver quién crece más rápido cuando $n \rightarrow \infty$. Es decir $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

EJERCICIO.-

Ordenar las siguientes funciones.

$n, \log n, 1, n \log n, n^2, n!, 2^n, n^k (k > 3)$

SOLUCIÓN

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset O(n^k)$$

$$O(2^n) \subset O(n!)$$

Jerarquía de costes computacionales: consecuencias prácticas (II)

Tiempos de ejecución en una máquina que ejecuta 10^9 pasos por segundo (~ 1 GHz), en función del coste del algoritmo y del tamaño del problema n :

Talla	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
10	3.322 ns	10 ns	33 ns	100 ns	1 μ s	1 μ s
20	4.322 ns	20 ns	86 ns	400 ns	8 μ s	1 ms
30	4.907 ns	30 ns	147 ns	900 ns	27 μ s	1 s
40	5.322 ns	40 ns	213 ns	1.6 μ s	64 μ s	18.3 min
50	5.644 ns	50 ns	282 ns	3 μ s	125 μ s	13 días
100	6.644 ns	100 ns	664 ns	10 μ s	1 ms	$40 \cdot 10^{12}$ años
1000	10 ns	1 μ s	10 μ s	1 ms	1 s	
10000	13 ns	10 μ s	133 μ s	100 ms	16.7 min	
100000	17 ns	100 μ s	2 ms	10 s	11.6 días	
1000000	20 ns	1 ms	20 ms	16.7 min	31.7 años	

Jerarquía de costes computacionales: consecuencias prácticas (II)

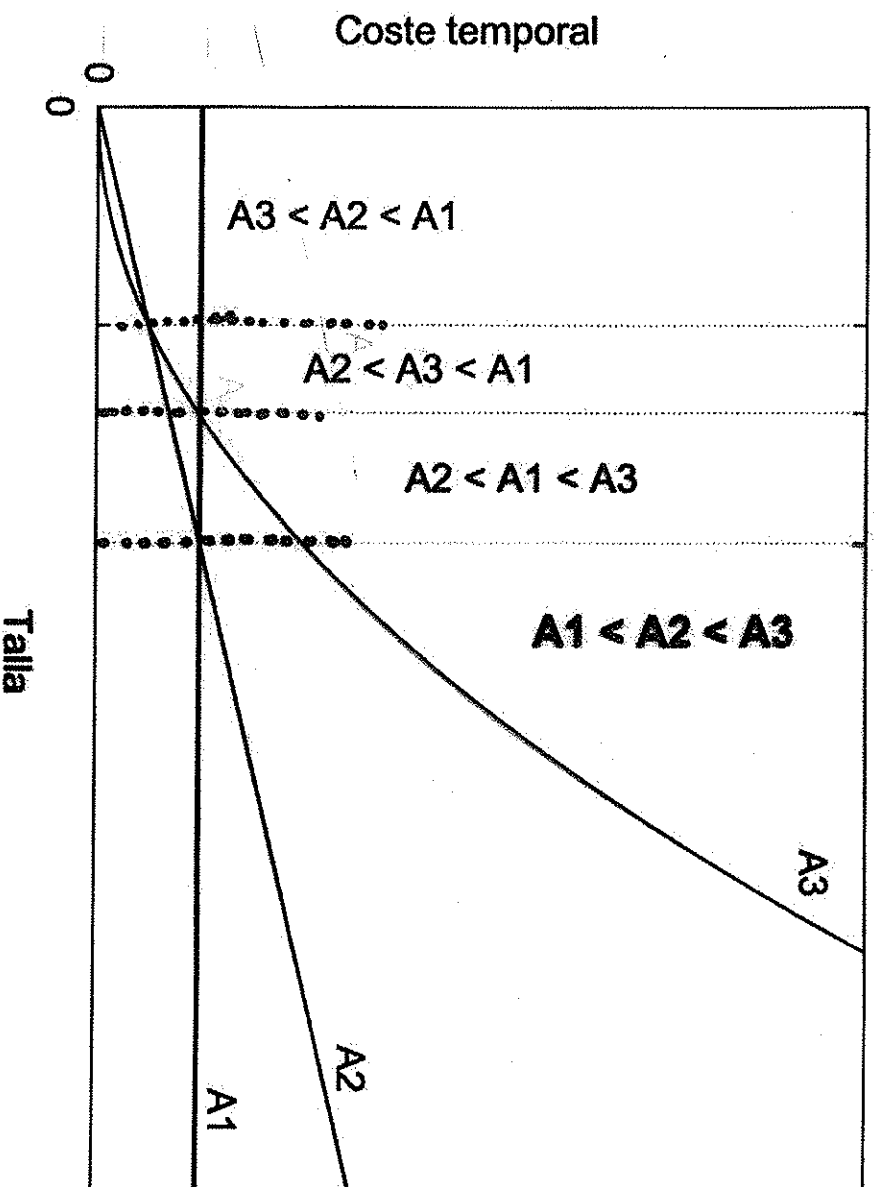
Tiempos de ejecución en una máquina que ejecuta 10^9 pasos por segundo (~ 1 GHz), en función del coste del algoritmo y del tamaño del problema n :

Talla	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
10	3.322 ns	10 ns	33 ns	100 ns	1 μs	1 μs
20	4.322 ns	20 ns	86 ns	400 ns	8 μs	1 ms
30	4.907 ns	30 ns	147 ns	900 ns	27 μs	1 s
40	5.322 ns	40 ns	213 ns	1.6 μs	64 μs	18.3 min
50	5.644 ns	50 ns	282 ns	3 μs	125 μs	13 días
100	6.644 ns	100 ns	664 ns	10 μs	1 ms	$40 \cdot 10^{12}$ años
1000	10 ns	1 μs	10 μs	1 ms	1 s	
10000	13 ns	10 μs	133 μs	100 ms	16.7 min	
100000	17 ns	100 μs	2 ms	10 s	11.6 días	
1000000	20 ns	1 ms	20 ms	16.7 min	31.7 años	

Coste asintótico

En general, tendríamos un comportamiento relativo de $A1$, $A2$, $A3$ tal como:

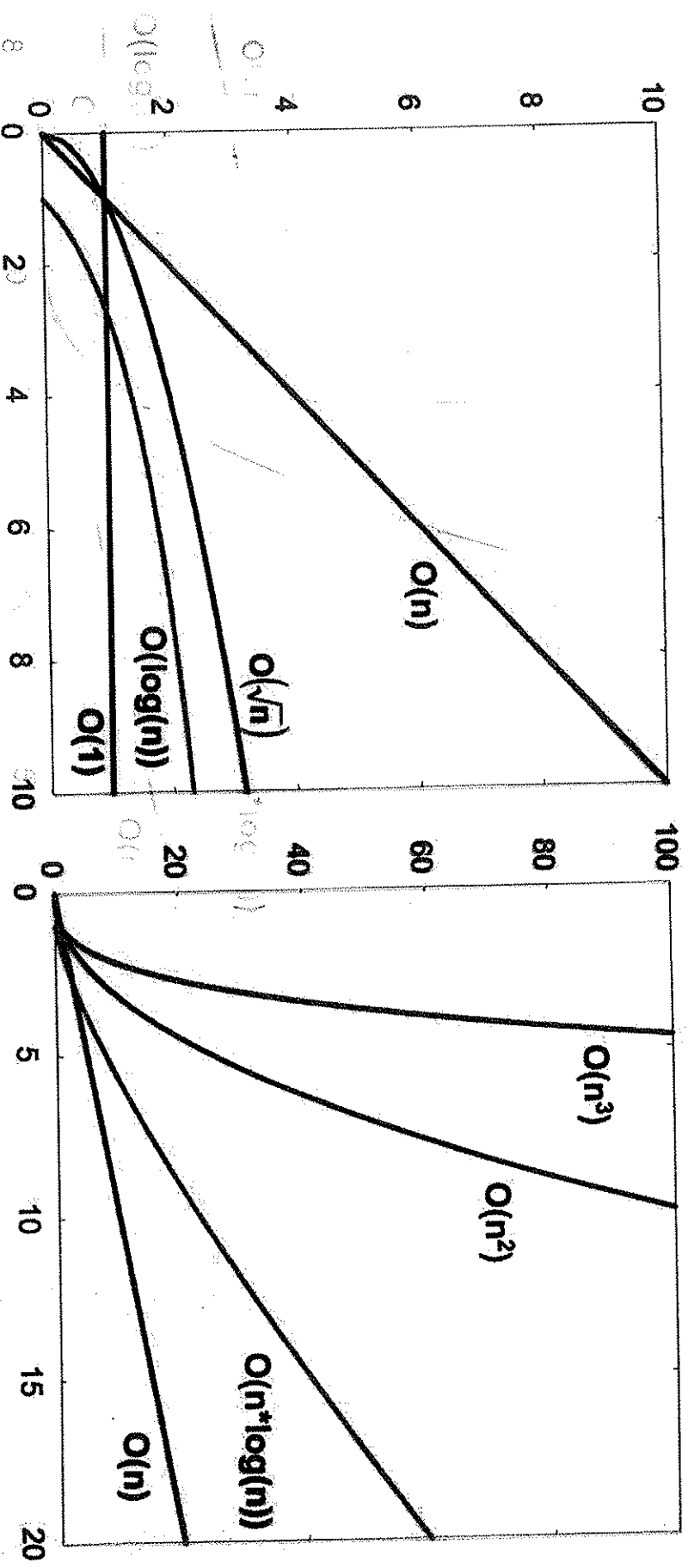
Cálculo de n^2 : costes relativos de $A1$, $A2$, $A3$



Una buena caracterización computacional de un programa:

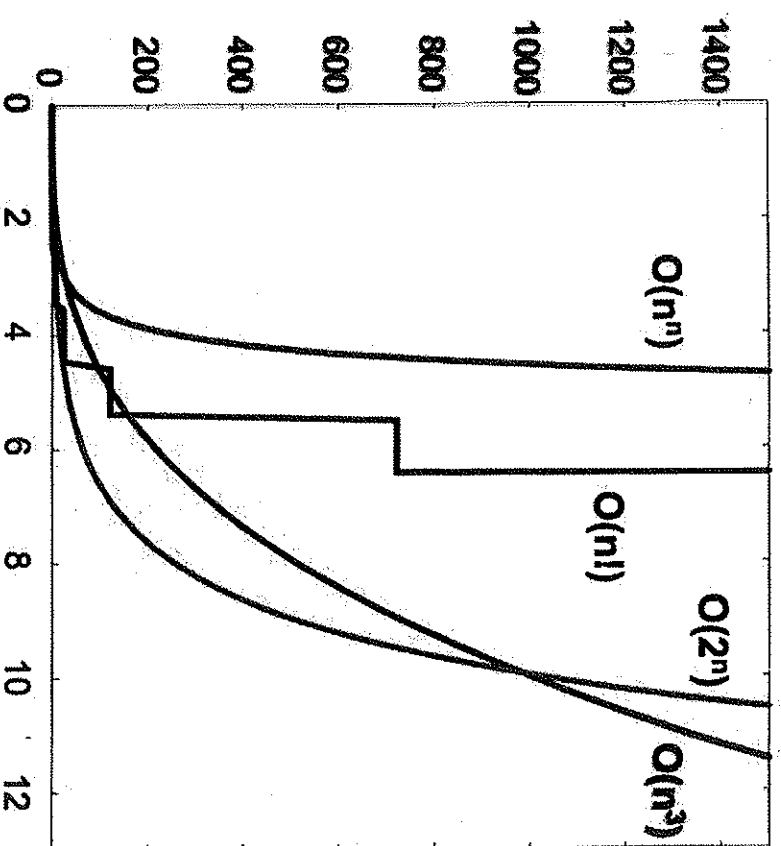
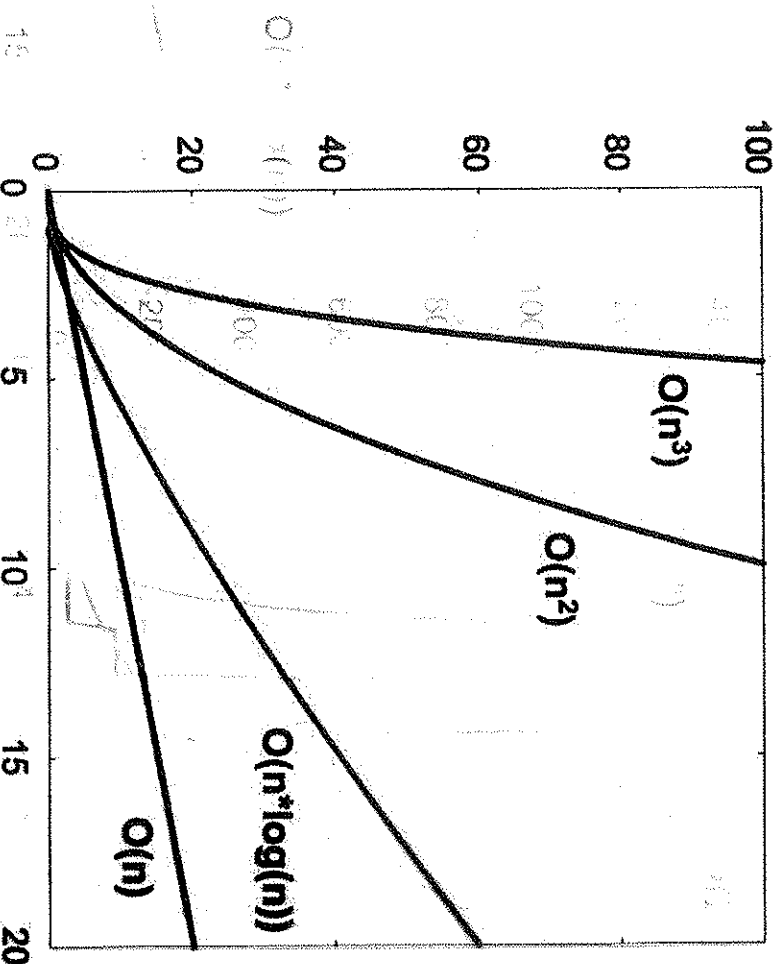
Dependencia funcional del coste con la talla – ¡para tallas grandes!

Notación asintótica: jerarquía de costes computacionales



sublineales, lineales y superlineales

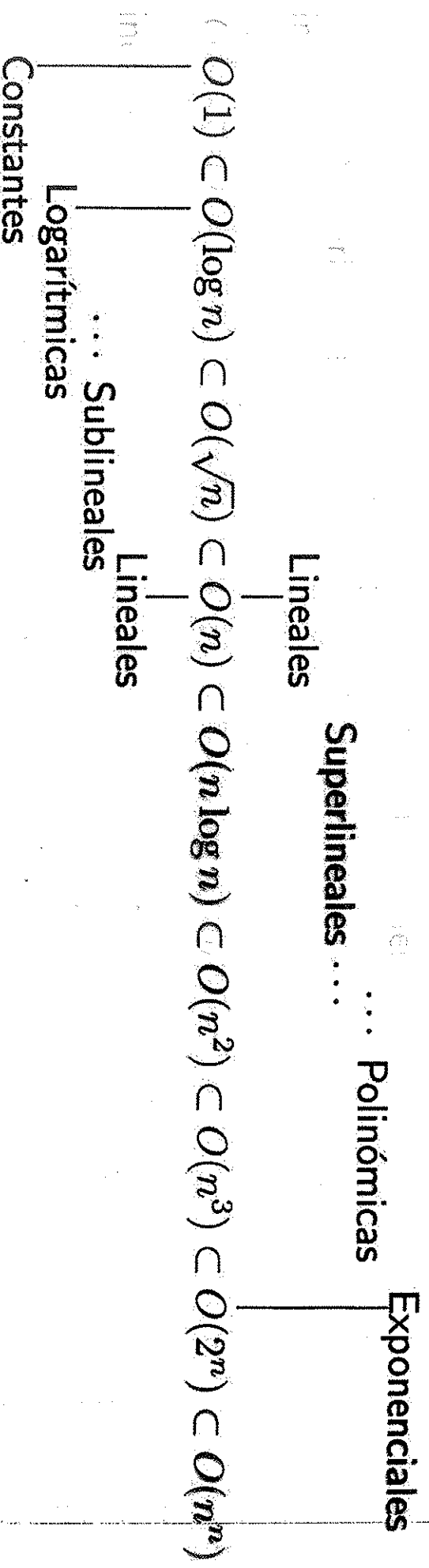
Notación asintótica: jerarquía de costes computacionales



superlineales, polinómicas y exponenciales

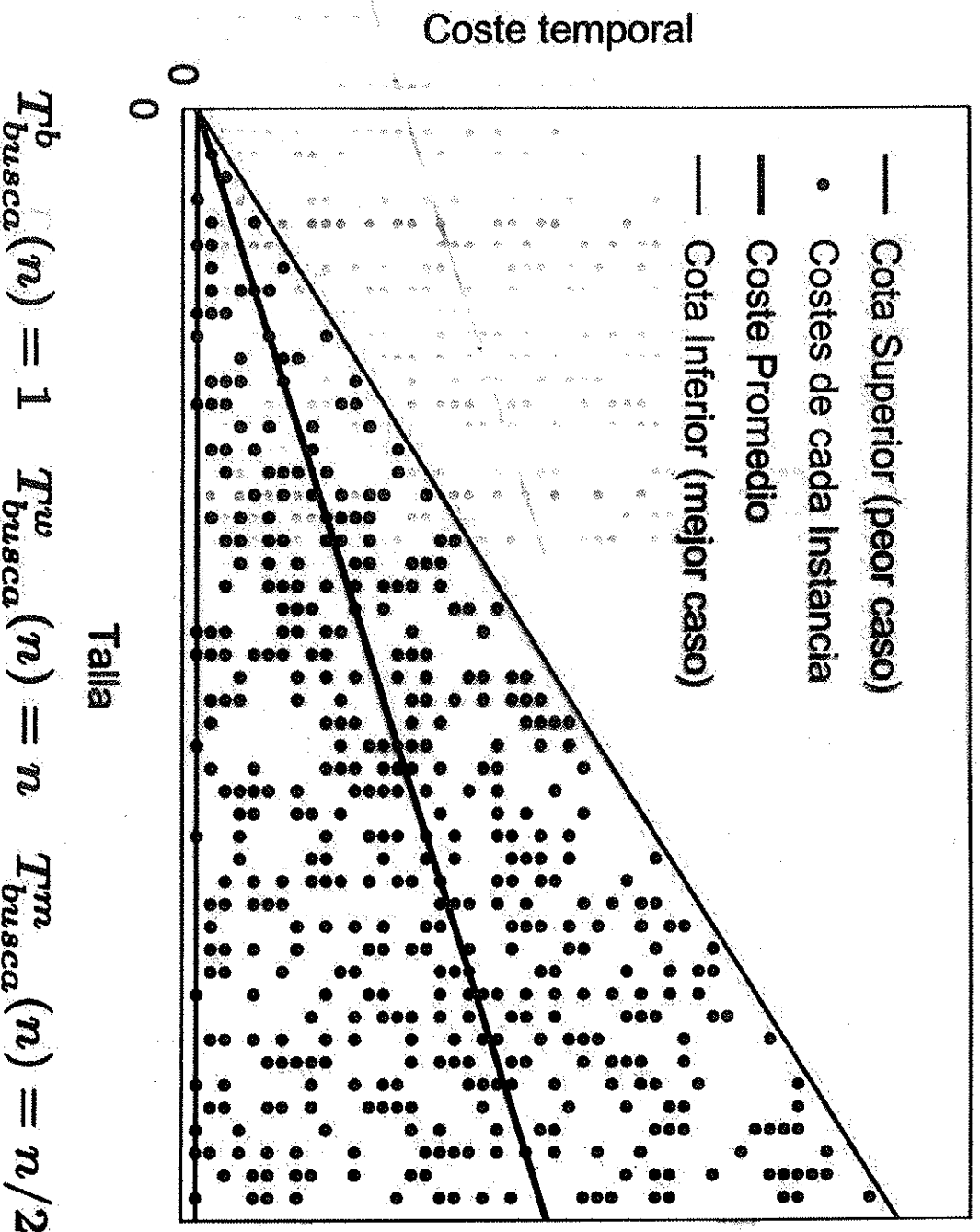
Notación asintótica: jerarquía de costes computacionales

Algunas relaciones entre órdenes usuales:



Extremos del coste: casos mejor, peor y promedio

Número de PASOS requeridos por 'busca'



①

LECCION 3.- CALCULO de la EFICIENCIA de un código

- RECURSOS MATEMÁTICOS NECESARIOS

- Propiedades de los logaritmos

$$-\log_b x \cdot y = \log_b x + \log_b y$$

$$-\log_b x/y = \log_b x - \log_b y$$

$$-\log_b x^y = y \cdot \log_b x$$

$$-\log_a x = \frac{\log_b x}{\log_b a}$$

- Propiedades de las exponenciales

$$-a^x y = (a^x)^y \quad -a^{x-y} = \frac{a^x}{a^y}$$

$$-a^{x+y} = a^x a^y \quad -b = a^{\log_a b}$$

$$-b^x = a^{x \cdot \log_a b}$$

$\lfloor x \rfloor$: representa al mayor entero menor o igual que x

$\lceil x \rceil$: representa el menor entero mayor o igual que x

SUMATORIAS

$$\sum_{i=s}^t f(i) = f(s) + f(s+1) + \dots + f(t)$$

- PROGRESION ARITMÉTICA $f(i) = i$

$$\sum_{i=0}^n i = 0 + 1 + 2 + \dots + n = n \cdot \frac{(n+1)}{2}$$

- PROGRESION GEOMÉTRICA $f(i) = a^i$

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

- SUMA de CUADRADOS

$$\sum_{i=0}^n i^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$$

- Normas para obtener la eficiencia de un código

• Operación Elemental $\rightarrow O(1)$

• Declaraciones
int a, b; $\rightarrow O(1)$

• Asignaciones
a = b $\rightarrow O(1)$

• Comparaciones simples
if (a < b) $\rightarrow O(1)$

• Operaciones Aritméticas (+, -, *, /)
a * b $\rightarrow O(1)$

• Entrada y salida de un dato básico

cin >> a; cout << b;
O(1) O(1)

REGLA de la SUMA

Sean dos trozos de código independientes (C_1, C_2) con eficiencia $T_1(n)$ y $T_2(n)$. Entonces la eficiencia del código union es:

$$\left. \begin{array}{l} T_1(n) \in O(f(n)) \\ T_2(n) \in O(g(n)) \end{array} \right\} \begin{array}{l} T_1(n) + T_2(n) \in O(\max(f(n), g(n))) \end{array}$$

Ej

int a = 5, b = 100;

if (a < b) cout << b;

else

for (int i = 0; i < n; i++)
{
 v[i] = a;
}

O(1) O(1)

$C_1 \rightarrow T_1(n) \in O(1)$

O(1)

O(1)

$T_2(n) \in O(n)$

C_2