LECCION 2 continuación

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 $O(\log_2(n))$
- LINEAL $O(n)$
- Cuadrático $O(n^2)$
- 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

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

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. $a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0 \in O(n^d)$
3. La base de los logaritmos no importa
   $\log_a(n) \in O(\log_b(n))$
4. El exponente en los logaritmos no importa
   $\log_a(n^3) \subset O(\log_a(n))$
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

- $t_1(n) = 100n$ ← A1
- $t_2(n) = \frac{n^2}{5}$ ← A2

¿Cual escogeriamos?

- Donde se cruzan

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

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

con $c = 1$ no = 500

cruzan.

- Luego el algoritmo A1 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 implementación y mantenimiento.

(5)

LECCIÓN 2:

EJERCICIO

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

n, $\sqrt{n}$, $\log n$, $\log\log n$, $\log^2 n$, $n/\log n$, $\sqrt{n}\log^2 n$, $(1/3)^n$, $\left(\frac{3}{2}\right)^n$, 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 $n \to \infty$. Es decir $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

EJERCICIO:

Ordenar los siguientes funciones.

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

n^3

SOLUCIÓN

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

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

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 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	2 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	

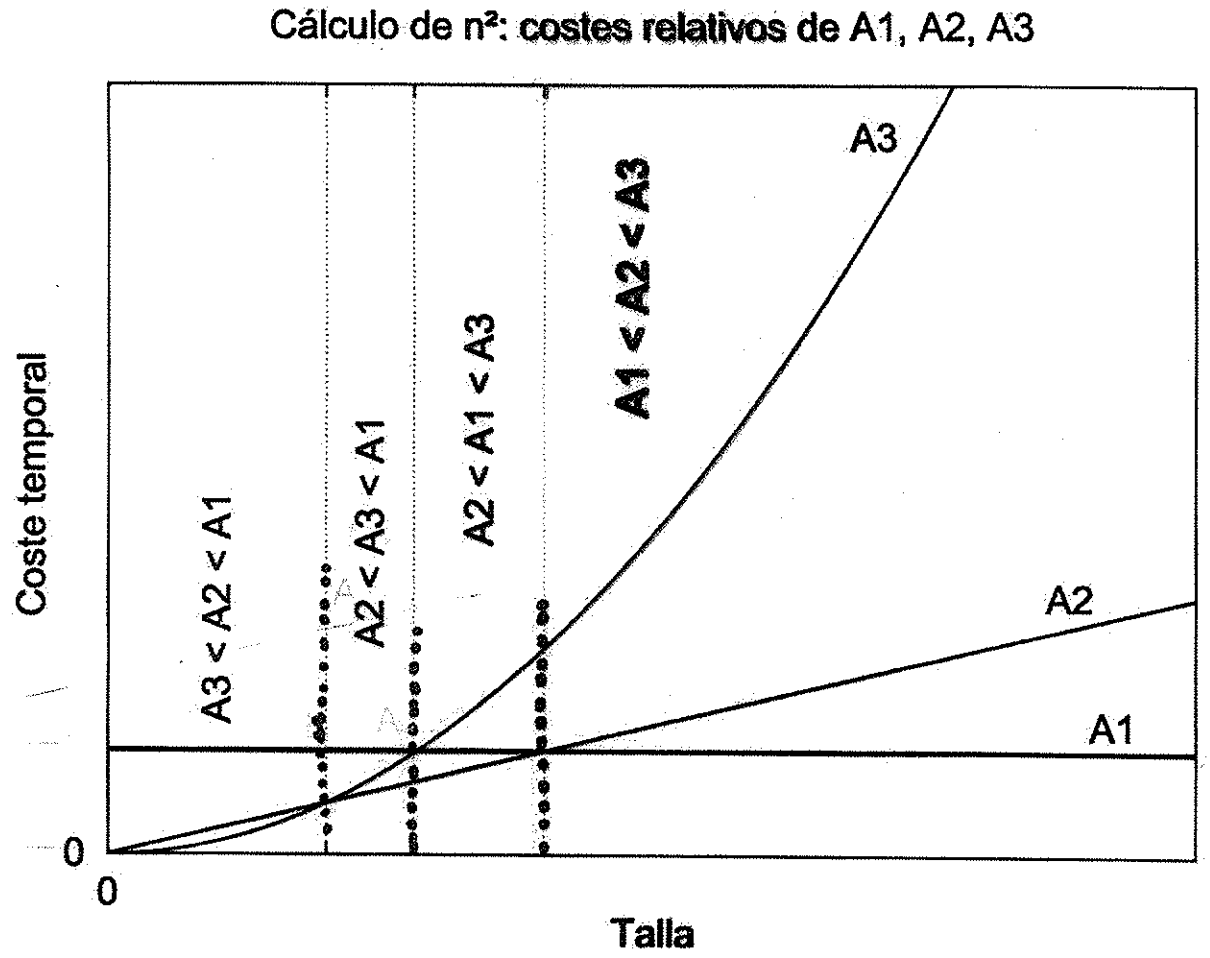
Jerarquía de costes computacionales: consecuencias prácticas (II)

Tiempos de ejecución en una máquina que ejecuta 10^9 *pasos por segundo* ($\sim$ *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	2 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

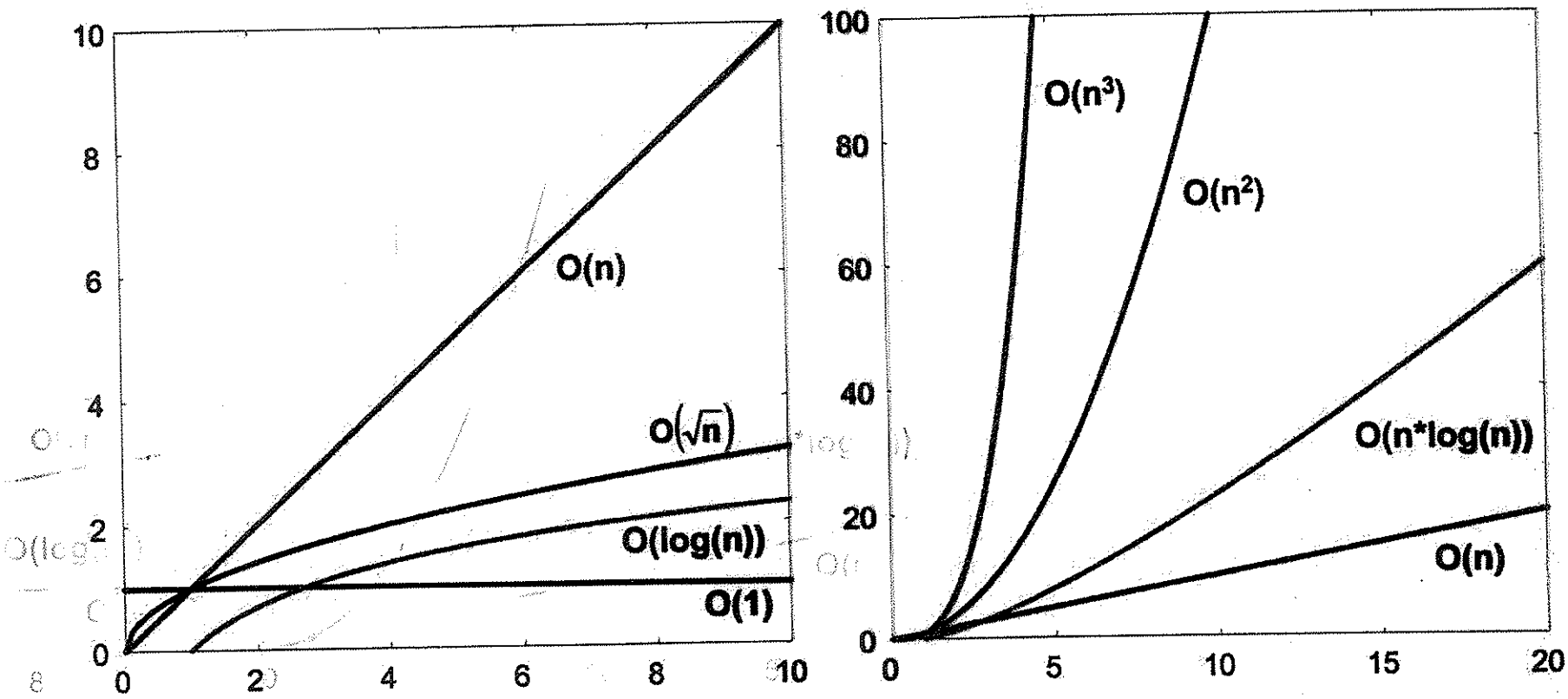
En general, tendríamos un comportamiento relativo de $A1$, $A2$, $A3$ tal como:



Una buena caracterización computacional de un programa:

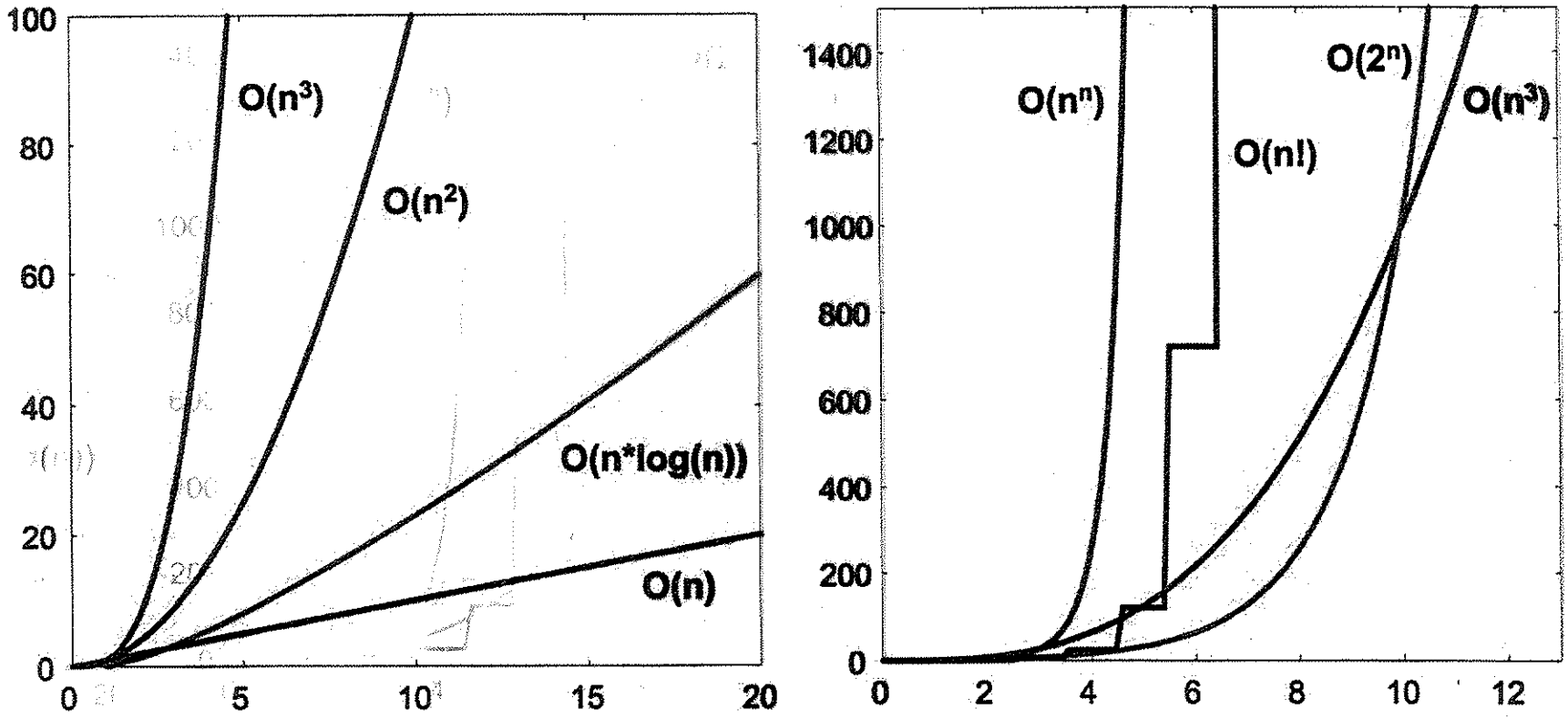
Dependencia funcional del coste con la talla – ¡para tallas grandes!

Notación asintótica: jerarquía de costes computacionales



sublineales, lineales y superlineales

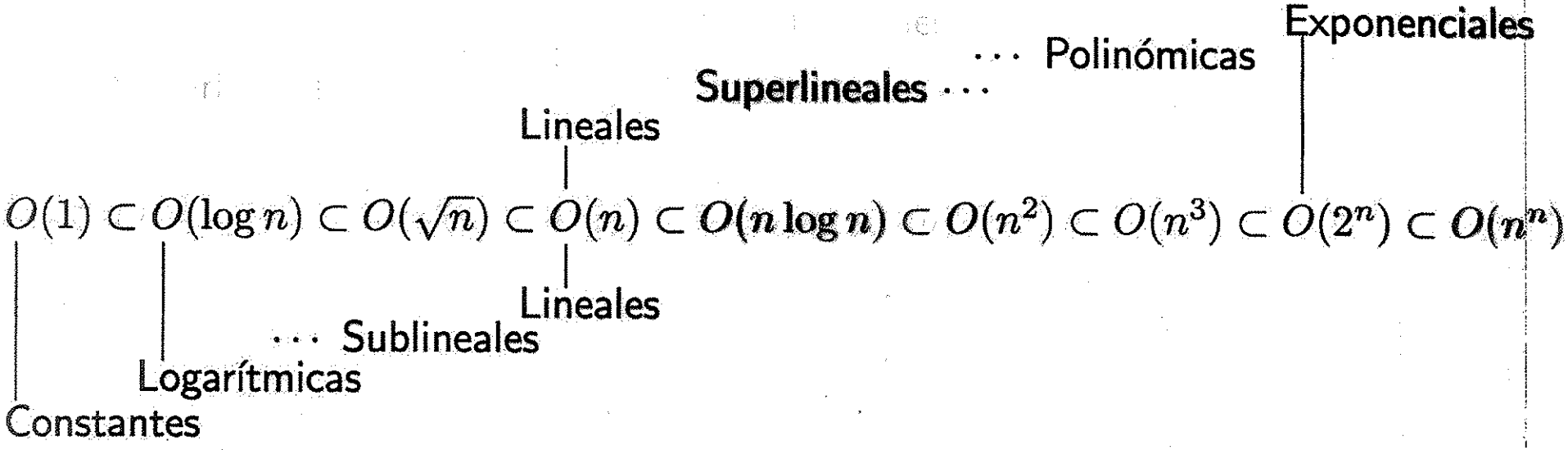
Notación asintótica: jerarquía de costes computacionales



superlineales, polinómicas y exponenciales

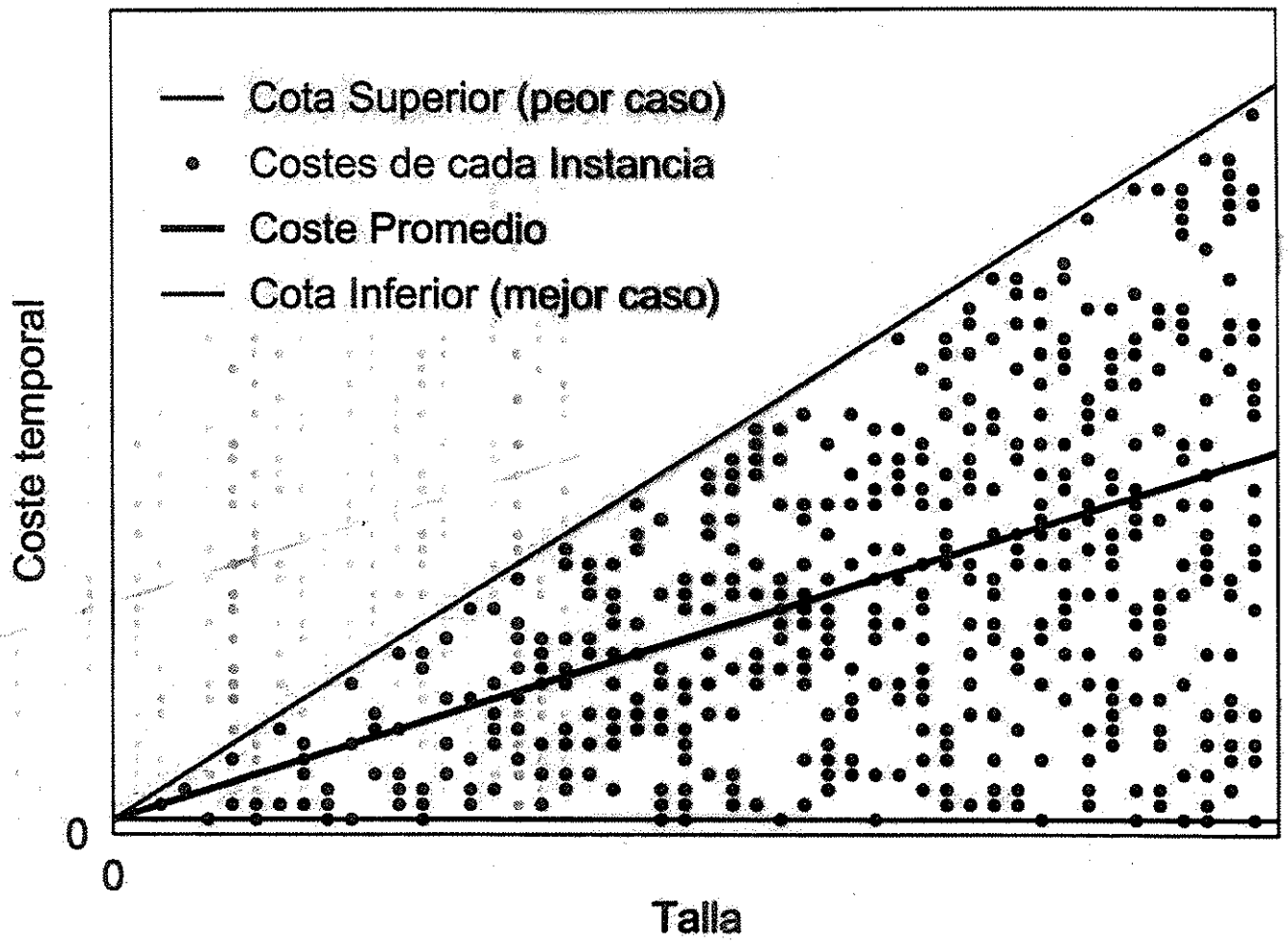
Notación asintótica: jerarquía de costes computacionales

Algunas relaciones entre órdenes usuales:



Extremos del coste: casos mejor, peor y promedio

Número de PASOS requeridos por 'busca'



$$T^{b}_{busca}(n) = 1 \quad T^{w}_{busca}(n) = n \quad T^{m}_{busca}(n) = n/2$$

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

- $a^{xy} = (a^x)^y$
- $a^{x-y} = \frac{a^x}{a^y}$
- $a^{x+y} = a^x a^y$
- $b = a^{\log_a b}$
- $b^x = a^{x \cdot \log_a b}$

$\lfloor x \rfloor$: 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

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

- Normas para obtener la eficiencia de un código

- Operacion Elemental $\longrightarrow O(1)$
  - Declaraciones
    `int a,b;` $\longrightarrow O(1)$
  - Asignaciones
    `a=b` $\longrightarrow O(1)$
  - Comparaciones simples
    `if (a<b)` $\longrightarrow O(1)$
  - Operaciones Antmética (+, -, *, /)
    `a*b` $\longrightarrow O(1)$
  - Entrada y salida de un dato basico
    `cin>>a;` $O(1)$ `cout<<b;` $O(1)$

REGLA de la SUMA

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

$T_1(n) \in O(f(n))$ y $T_2(n) \in O(g(n))$ $\Big\}$ $T_1(n) + T_2(n) \in O(\max(f(n), g(n)))$

Ej

```
int a=5, b=100;                          O(1)
if (a<b)                                 O(1)
    cout b;                              O(1)
else                                     O(1)
  for (int i=0; i<n; i++)                O(n)
    v[i]=a;                              O(1)
```

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

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

C_2