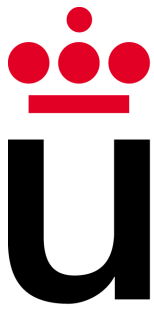
Tema 2.2: Notaciones Asintóticas

Diseño y Análisis de Algoritmos



Universidad
Rey Juan Carlos

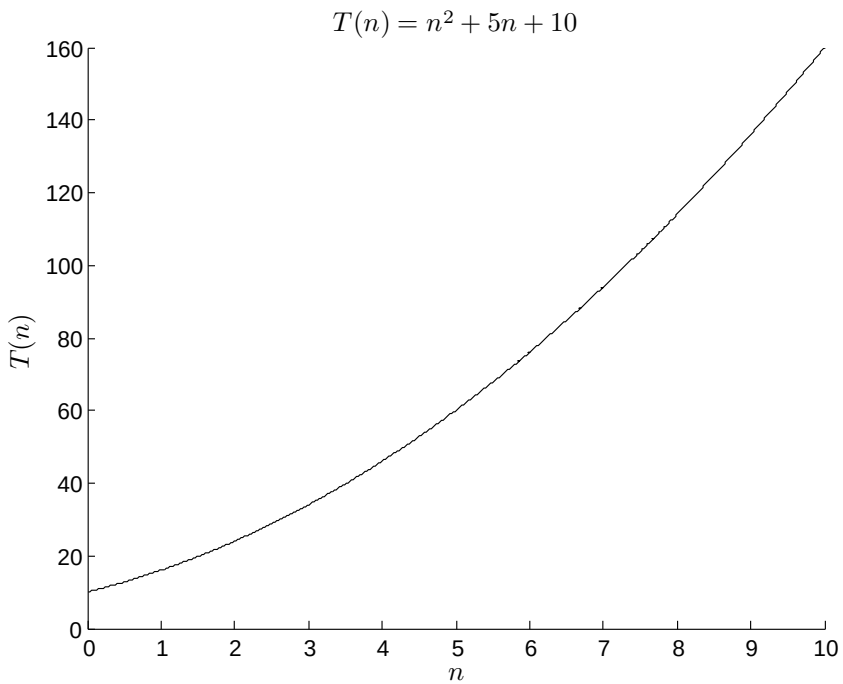
Contenidos

Notaciones asintóticas

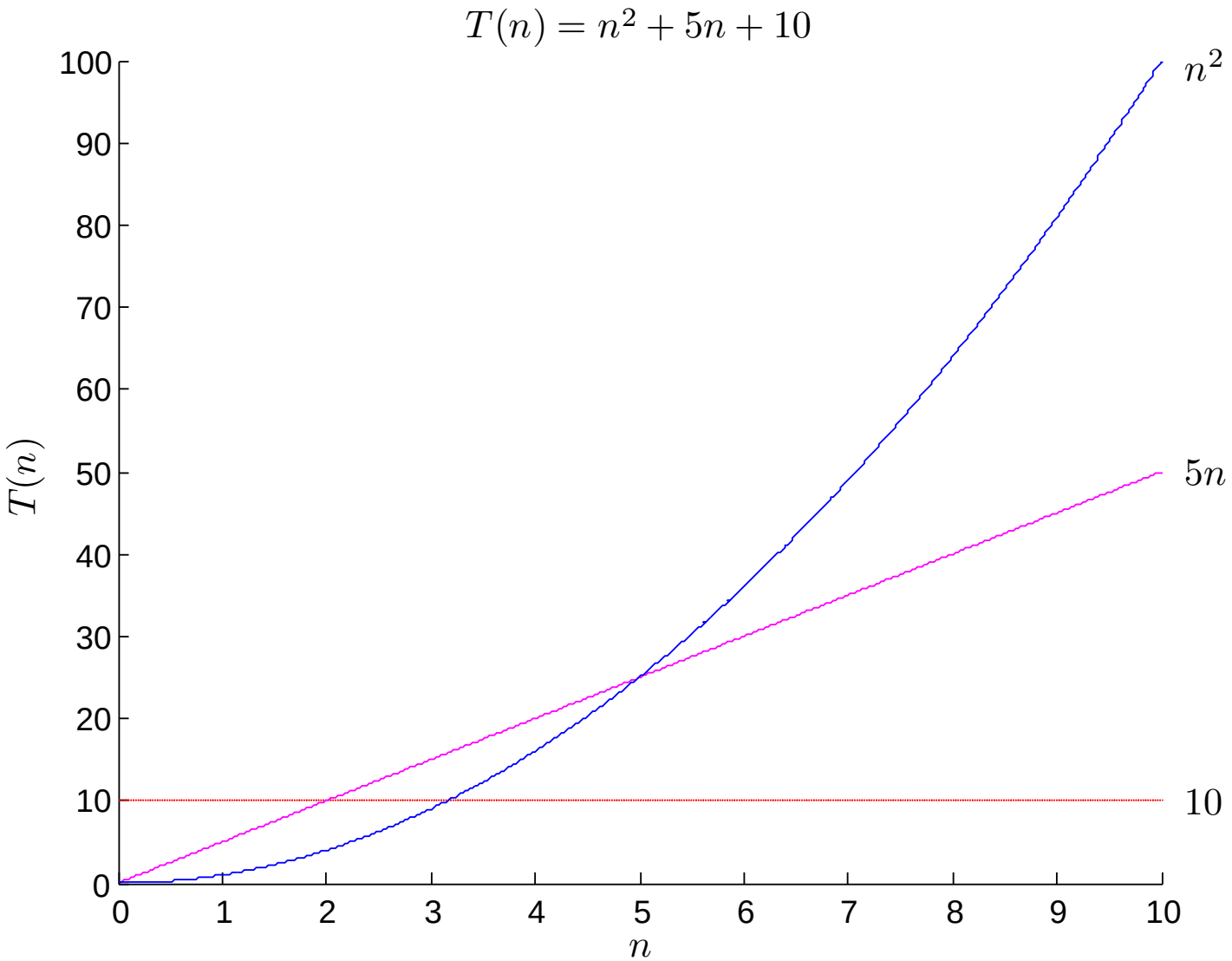
- Nos interesa cómo crece el tiempo de ejecución
  - Según aumenta el tamaño de la entrada
  - "En el límite", según el tamaño crece sin cota
- Eficiencia asintótica de algoritmos
  - Asumimos que las entradas son muy grandes
  - Nos interesa el "orden de crecimiento"
  - Las constantes y términos de orden inferior no son relevantes, al ser *dominados* por un término de orden superior
- El algoritmo con mejor coste o eficiencia asintótica suele ser la mejor elección
  - Salvo para entradas muy pequeñas

Tiempo de ejecución de un algoritmo

- El tiempo de ejecución lo deberemos expresar mediante una fórmula (función) matemática
  - Es importante saber qué argumentos debe tomar dicha función
- Consideremos la siguiente función:



Descomposición



Términos de mayor orden

- El término que más nos importa es n^2
  - Es el **término de mayor orden**
  - Puede haber varios (si la función depende de más de un parámetro)
- Para valores pequeños de n todos los términos influyen
- Mediante la notación asintótica vamos a simplificar y a aislar dichos términos que más influyen cuando n toma valores muy grandes

Primeras nociones informales

- Supongamos que tenemos dos funciones $f(n)$ y $g(n)$
  - $f(n)$ es *asintóticamente menor* que $g(n)$ cuando:

$$f(n) < g(n) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

  - $f(n)$ es *asintóticamente mayor* que $g(n)$ cuando:

$$f(n) > g(n) \iff g(n) < f(n)$$

  - $f(n)$ es *asintóticamente igual* que $g(n)$ cuando:

$$f(n) = g(n) \iff f(n) \not< g(n) \quad \text{y} \quad g(n) \not< f(n)$$

Órdenes que más aparecen

- Considerados generalmente como "tratables"

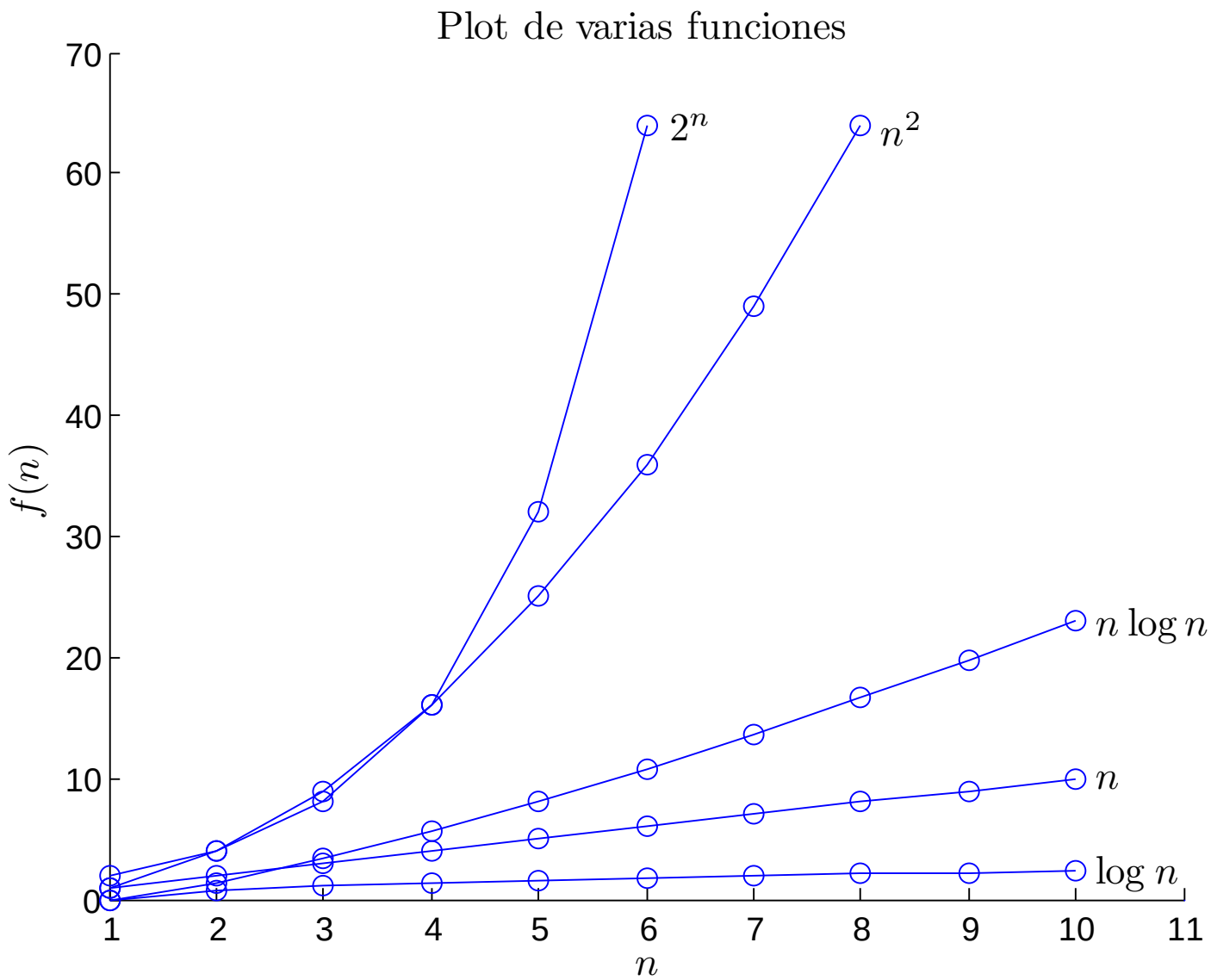
$$1 \quad < \quad \log n \quad < \quad n \quad < \quad n \log n \quad < \quad n^2$$

- Considerados generalmente como "intratables"

$$n^2 \quad < \quad n^3 \quad < \quad 2^n \quad < \quad n!$$

- n^2 se encuentra en el límite
- Siempre hay que tener en cuenta el tamaño de la entrada (n) para poder decir si un problema es tratable o intratable para cierto algoritmo

Curvas


Plot de varias funciones
70
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10
11
$f(n)$
n
2^n
n^2
$n \log n$
n
$\log n$

Tiempos

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65.536
5	32	160	1024	32.768	4.295.967.296

- Un orden exponencial es extremadamente costoso, incluso frente a ordenes polinómicos
- Un orden factorial es incluso más costoso que un orden exponencial

Noción informal de cota superior $\mathcal{O}$

- $f(n)$ es asintóticamente menor o igual que $g(n)$
- $g(n)$ es una cota superior de $f(n)$ (asintóticamente)

$$f(n) \in \mathcal{O}(g(n)) \iff f(n) \leq g(n)$$

- Ejemplos:
  - $2n + 5 \in \mathcal{O}(3n^2 - 8n)$
  - $2n + 5 \in \mathcal{O}(n + 10)$
  - $2n + 5 \in \mathcal{O}(n!)$
  - $2n + 5 \in \mathcal{O}(n) \Longleftarrow$ Querremos la cota superior más baja

Noción informal de cota inferior Ω

- $f(n)$ es asintóticamente mayor o igual que $g(n)$
- $g(n)$ es una cota inferior de $f(n)$ (asintóticamente)

$$f(n) \in \Omega(g(n)) \iff f(n) \geq g(n)$$

- Ejemplos:
  - $2n + 5 \in \Omega(3 \log n)$
  - $2n + 5 \in \Omega(4n + 10)$
  - $2n + 5 \in \Omega(1)$
  - $2n + 5 \in \Omega(n) \Longleftarrow$ Querremos la cota inferior más alta

Noción informal de cota ajustada Θ

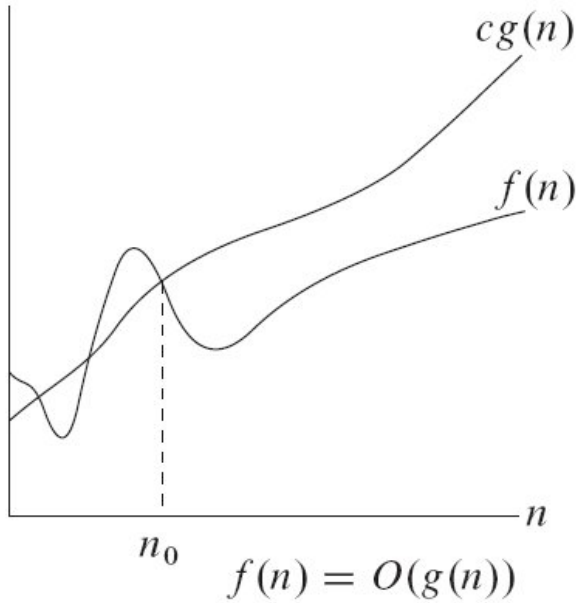
- $f(n)$ es asintóticamente igual que $g(n)$
- $g(n)$ es una cota ajustada de $f(n)$ (asintóticamente)

$$\boxed{f(n) \in \Theta(g(n)) \iff f(n) = g(n)}$$

- Ejemplos:
  - $2n + 5 \in \Theta(8n + 10)$
  - $2n + 5 \in \Theta(n)$
- Hay otras cotas (por ejemplo, o, ω), que no veremos en la asignatura

Definición formal de cota superior $\mathcal{O}$

$$\mathcal{O}(g(n)) = \left\{ f(n) : \exists\ c > 0 \text{ y } n_0 > 0 \ / \ 0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_0 \right\}$$



- Idea principal: a partir de n_0, $c \cdot g(n)$ siempre supera (o iguala) a $f(n)$

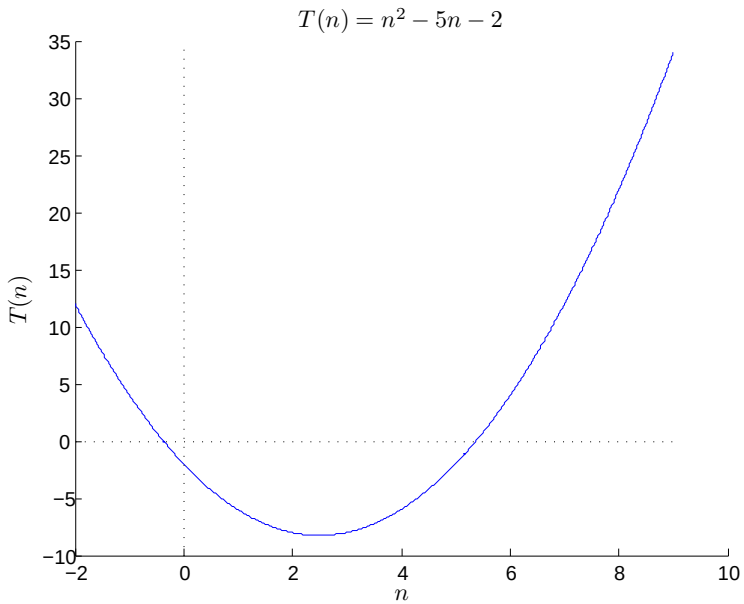
Demostración de $f(n) \in \mathcal{O}(g(n))$

- Para demostrar que una función $f(n) \in \mathcal{O}(g(n))$ será necesario encontrar **una** (cualquier) pareja de constantes $c > 0$ y $n_0 > 0$, de tal forma que se verifiquen las condiciones de la definición

- Ejemplo: demostrar que $5n + 2 \in \mathcal{O}(n)$
  - Hay que encontrar $c > 0$ y $n_0 > 0$ tales que $5n + 2 \leq cn$, $\forall n \geq n_0$
  - Para ello, seguimos los siguientes pasos:
    1. Elegir una constante adecuada (por ejemplo $c = 6$)
    2. Buscar un $n > 0$ tal que se cumpla que $5n + 2 \leq cn$
  - Para $c = 6$ se cumple para todo $n \geq 2$, luego podemos tomar $n_0 = 2$, y hemos encontrado una pareja de constantes (hay infinitas parejas más, pero basta con encontrar una)

Demostración de $f(n) \in \mathcal{O}(g(n))$

- Ejemplo: demostrar que $5n + 2 \in \mathcal{O}(n^2)$
  1. Elegir una constante adecuada (por ejemplo $c = 1$)
  2. Buscar qué valores de n hacen que se cumpla que $5n + 2 \leq n^2$
     - Para ello analizamos la desigualdad $n^2 - 5n - 2 \geq 0$



Demostración de $f(n) \in \mathcal{O}(g(n))$

- continuación...
  - $n^2 - 5n - 2$ es una función cuadrática (convexa) con raíces en $-0,37$ y $5,37$
  - Por tanto, siempre será positiva para $n \geq 6$
  - Para $c = 1$ y $n_0 = 6$ se cumplen las condiciones de la definición y queda demostrado
- Si escogemos $c = 5$, $n_0 = 2$ es suficiente
- Si escogemos $c = 8$, $n_0 = 1$ es suficiente

Demostración de $f(n) \in \mathcal{O}(g(n))$

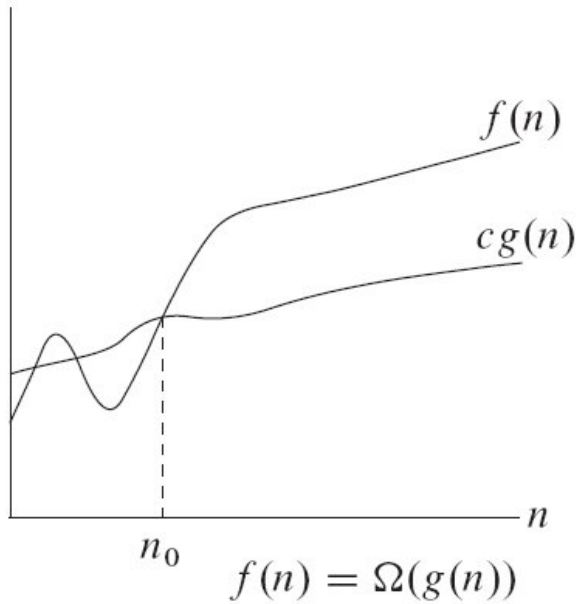
- Ejemplo: demostrar que $3n^2 + 2n - 2 \in \mathcal{O}(n)$
  - En este caso no vamos a poder encontrar las constantes (obviamente, ya que no lo podremos demostrar al no ser cierto)
  - Cojamos la constante que cojamos tendríamos que demostrar:

$$3n^2 + 2n - 2 \leq cn \quad \Rightarrow \quad 3n^2 + (2 - c)n - 2 \leq 0$$

    para todo $n \geq n_0$
  - Como $3n^2 + (2 - c)n - 2$ es una función cuadrática convexa, que crece hasta el $+\infty$ según aumenta n, no va a ser negativa **siempre** a partir de ningún n_0
  - Por tanto, es imposible encontrar una pareja de constantes c y n_0

Definición formal de cota inferior Ω

$$\Omega(g(n)) = \left\{ f(n) : \exists\ c > 0 \text{ y } n_0 > 0 \ / \ 0 \leq c \cdot g(n) \leq f(n), \forall n \geq n_0 \right\}$$



- Idea principal: a partir de n_0, $f(n)$ siempre supera (o iguala) a $c \cdot g(n)$

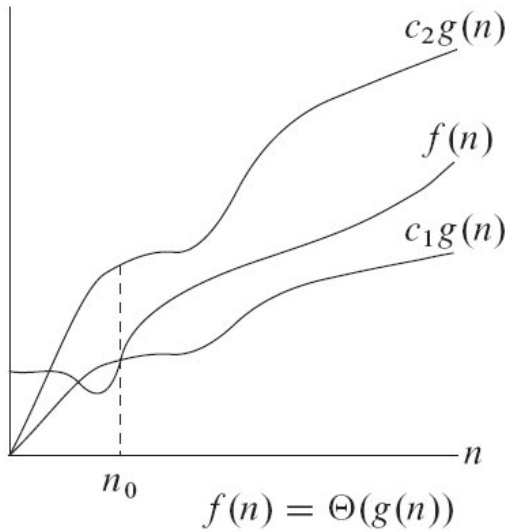
Demostración de $f(n) \in \Omega(g(n))$

- Ejemplo: demostrar que $3n^2 + 2 \in \Omega(n)$
  1. Elegir una constante adecuada (por ejemplo $c = 5$)
  2. Buscar valores de n tales que se cumpla $3n^2 + 2 \geq 5n$
     - Hay que ver para qué valores de n se cumple que $3n^2 - 5n + 2 \geq 0$
     - $3n^2 - 5n + 2$ es una función cuadrática (convexa) con raíces en $2/3$ y 1
     - Por tanto, siempre será positiva para $n \geq 1$
     - Para $c = 5$ y $n_0 = 1$ se cumplen las condiciones de la definición y queda demostrado

Definición formal de cota ajustada Θ

$$\Theta(g(n)) = \Big\{ f(n) : \exists\ c_1 > 0,\ c_2 > 0 \text{ y } n_0 > 0\ /$$

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall n \geq n_0 \Big\}$$



- Idea principal: a partir de n_0, $f(n)$ siempre queda en medio de $c_1 g(n)$ y $c_2 g(n)$

Demostración de $f(n) \in \Theta(g(n))$

$$f(n) \in \Theta(g(n)) \iff \begin{cases} f(n) \in \mathcal{O}(g(n)) \\ \text{y} \\ f(n) \in \Omega(g(n)) \end{cases}$$

- Ejemplo: demostrar que $n^2/2 - 3n \in \Theta(n^2)$
  - Se busca que $c_1 n^2 \leq n^2/2 - 3n \leq c_2 n^2$
  - Encontramos, por ejemplo: $c_1 = 1/14$ para $n \geq 7$ (Ω)
  - Y, por ejemplo: $c_2 = 1/2$ para $n \geq 1$ ($\mathcal{O}$)
  - En este momento queda demostrado
  - Según la definición habríamos encontrado $c_1 = 1/14$, $c_2 = 1/2$, y $n_0 = 7$

Funciones de dos parámetros

- Ejemplo: mezclar dos vectores ordenados de longitudes n y m
  - $T(n, m) \in \Theta(n + m)$
- Definición formal de $\mathcal{O}$:

$$\mathcal{O}(g(n,m)) = \Big\{ f(n,m) : \exists\ c > 0,\ n_0 > 0, \text{ y } m_0 > 0\ /$$

$$0 \leq f(n,m) \leq c \cdot g(n,m), \quad \forall n \geq n_0, \text{ y } m \geq m_0 \Big\}$$

Funciones de dos parámetros

- En la práctica usaremos límites

$$f(n,m) > g(n,m) \iff \begin{cases} \lim_{n\to\infty} \frac{g(n,m)}{f(n,m)} = 0 \quad \text{y} \quad \lim_{m\to\infty} \frac{g(n,m)}{f(n,m)} \neq \infty \\ \text{o} \\ \lim_{m\to\infty} \frac{g(n,m)}{f(n,m)} = 0 \quad \text{y} \quad \lim_{n\to\infty} \frac{g(n,m)}{f(n,m)} \neq \infty \end{cases}$$

Simplificación

- Simplificar $\Theta(3m^2n + m^3 + 10mn + 2n^2)$

1. Eliminar constantes

$$\Theta(m^2n + m^3 + mn + n^2)$$

2. Simplificar términos "contenidos" en otros
   - $m^2n > mn$, por tanto, se puede eliminar el término mn

$$\lim_{n\to\infty} \frac{mn}{m^2n} = \frac{1}{m} \neq \infty \quad \text{y} \quad \lim_{m\to\infty} \frac{mn}{m^2n} = 0$$

$$\boxed{\Theta(m^2n + m^3 + n^2)}$$

- Si probamos las tres combinaciones de parejas de funciones que aparecen en la fórmula final veremos que ninguna es superior a otra

Comentarios adicionales

- Sea ρ alguna medida de complejidad computacional asintótica
  - Las constantes no importan

$$\rho(kg(n)) = \rho(g(n))$$

  - Término de mayor orden de un polinomio

$$\rho(a_m x^m + a_{m-1} x^{m-1} + \ldots + a_1 x^1 + a_0) = \rho(x^m)$$

  - La base de los logaritmos no importa

$$\rho(\log_x g(n)) = \rho(\frac{\log_y g(n)}{\log_y x}) = \rho(\frac{1}{\log_y x} \log_y g(n)) = \rho(\log_y g(n)) = \rho(\log g(n))$$

Comentarios adicionales

- $\mathcal{O}$, Ω, y Θ definen **conjuntos**
  - Lo correcto es escribir $f(n) \in \mathcal{O}(g(n))$
  - A veces se escribe $f(n) = \mathcal{O}(g(n))$, aunque es un "abuso" de notación
  - Y lo mismo con Ω y Θ
- Las funciones ($f(n)$, $g(n)$, $f(n, m)$, etc.) son siempre positivas
- Es un error decir que si $f(n) \in \mathcal{O}(g(n))$, entonces $f(n)$ tarda al menos $g(n)$
  - Al contrario, tarda como mucho $g(n)$ ($g(n)$ es **cota superior**)
- Con estas definiciones las constantes no influyen: se proporcionan cotas hasta un factor constante multiplicativo
- Hay notaciones en las que tratan a los términos logarítmicos como irrelevantes también

Comentarios adicionales

- ¡Que un algoritmo tarde $\mathcal{O}(n^2)$, $\Omega(n^2)$, o $\Theta(n^2)$ para algunas entradas no quiere decir que tarde $\mathcal{O}(n^2)$, $\Omega(n^2)$, o $\Theta(n^2)$ para todas, o en general!
- Cuando hablamos de $\mathcal{O}$ normalmente lo hacemos en referencia al **peor caso**, que es cuando un algoritmo tarda más
  - En ese caso damos una cota **superior** del tiempo o número de operaciones
  - Es como una "garantía" de que el coste nunca va a superar la cota proporcionada
- Cuando se indica una cota, siempre hay que asociarla a un determinado tipo de entrada:
  - Caso mejor, peor, o medio

Comentarios adicionales

- Si se pide realizar una demostración **utilizando la definición** debéis encontrar las **constantes** que verifiquen las condiciones de las definiciones
- Si no se pide, podréis usar límites:

$$f(n) \in \Theta(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = \text{constante} > 0$$

$$f(n) \in \Omega(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \quad \text{constante o infinito}$$

$$f(n) \in \mathcal{O}(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \quad \text{constante o cero}$$

Comentarios adicionales

- ¡La elección del tamaño de entrada es crucial!

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

- ¿Qué representa n, el número de bits (3), o el número total de palabras (8)?
  - Ambas posibilidades son válidas, pero hay que especificar claramente cuál se va a usar

Ordenación por comparación

- Supongamos que tenemos un problema de ordenación **por comparación** de una secuencia de datos
  - Solo se usan comparaciones para determinar el orden final
- Los algoritmos que resuelven el problema pueden aplicarse para cualquier tipo de datos (enteros, reales, cadenas,...)
- Si los datos son reales, ¿merece la pena realizar un estudio sobre la distribución de éstos?
  - NO. Solo interesa el puesto en la ordenación (primero, segundo,..., último), y no los valores de los reales
- Entonces, ¿cómo se analizan?
  - Escogiendo el caso mejor, peor, medio...
- La cota inferior del tiempo para una secuencia de n datos es:

$$\boxed{\Omega(n \log n)}$$

Ordenación en tiempo lineal

- Existen algoritmos de ordenación que tardan $\mathcal{O}(n)$, pero usan otras operaciones diferentes de las comparaciones
- *Counting-sort*
  - Los elementos a ordenar son enteros y pertenecen al intervalo $[0, k]$
  - Si $k \in \mathcal{O}(n)$, entonces el algoritmos tarda $\Theta(n)$
  - Usa tres vectores:
    - A[1..n], es el vector de entrada
    - B[1..n], es el vector de salida
    - C[0..k], es un vector auxiliar

Idea del counting-sort

- Se recorre la secuencia A y se cuenta el número de veces que aparece cada entero

A	2	5	3	0	2	3	0	3	n = 8

C	2	0	2	3	0	1	k = 5
	0	1	2	3	4	5	

- Hay varias formas de obtener el vector ordenado

B	0	0	2	2	3	3	3	5

- A continuación se describe un algoritmo eficiente

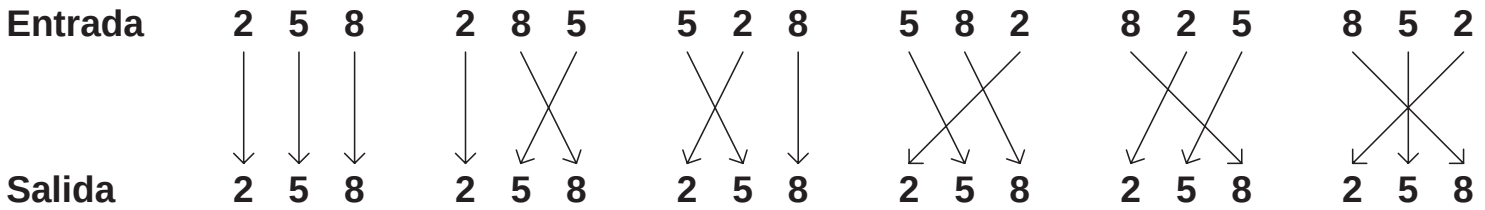
Counting-sort

```
Counting-sort(A)

FOR i=0..k                          Θ(k)
  C[i] = 0

FOR j=1..length(A)                  Θ(n)
  C[A[j]]++

FOR j=1..k                          Θ(k)
  C[i] = C[i] + C[i-1]

FOR j=length(A)..1                  Θ(n)
  B[C[A[j]]] = A[j]
  C[A[j]]--
```

- $T(n) \in \Theta(n+k)$
- Si $k \in O(n)$, entonces $T(n) \in \Theta(n)$

Algoritmos de ordenación y permutaciones

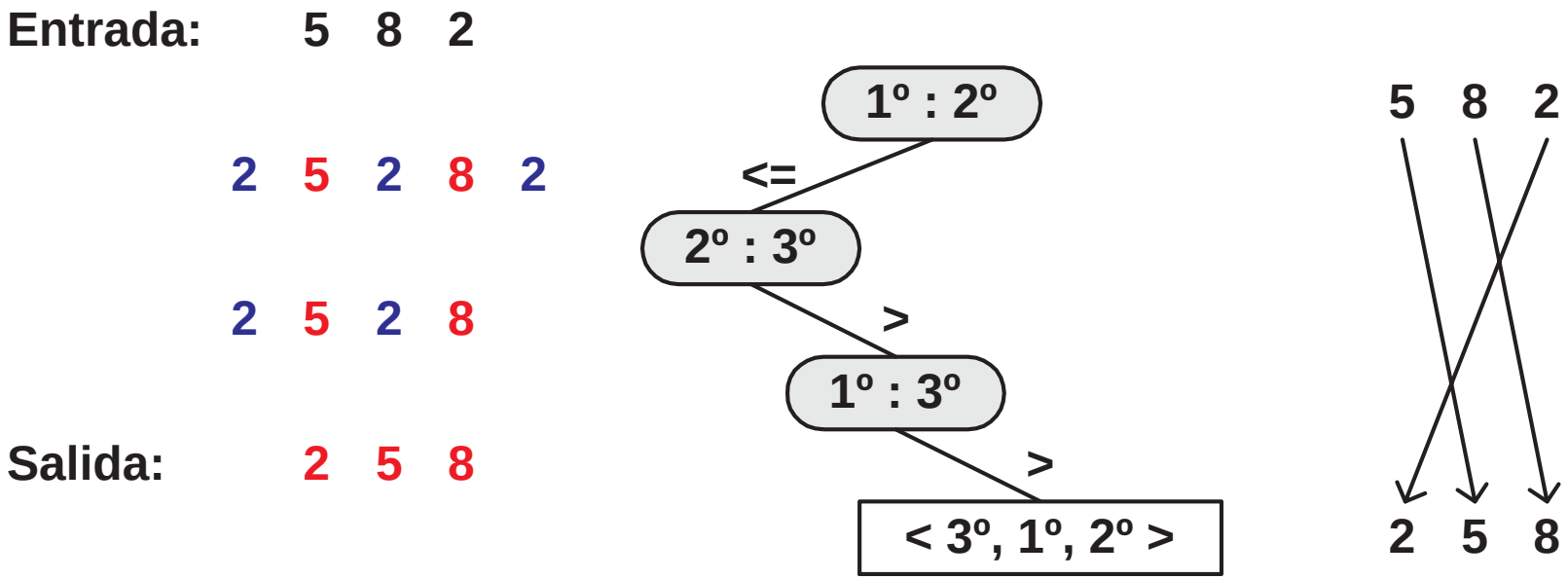
- Los algoritmos de ordenación por comparación tienen que ser capaces de generar cualquier permutación de un vector:



- Para un vector de n elementos, hay $n!$ posibles permutaciones

Algoritmos de ordenación y comparaciones

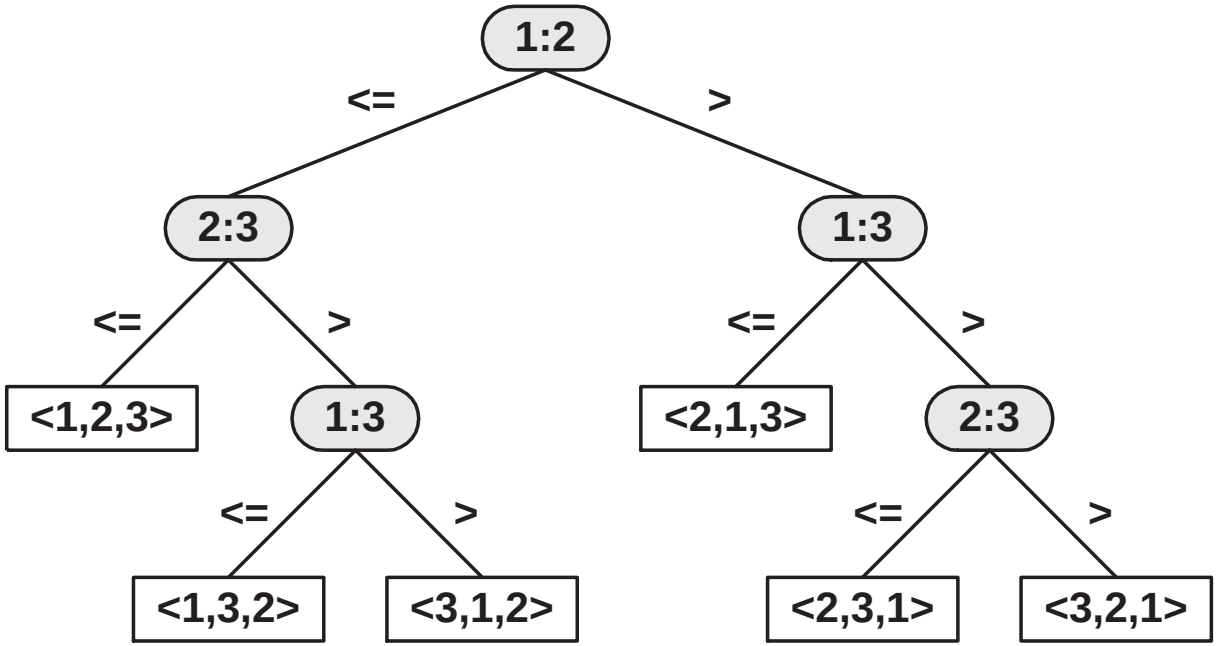
- Deben realizar varias comparaciones hasta hasta determinar la permutación correcta para cualquier entrada



- Incertidumbre
- Ordenados correctamente

Algoritmos de ordenación como árboles de decisión

- Los algoritmos de ordenación pueden verse de manera abstracta en términos de un árbol de decisión:



X:Y — Se comparan los elementos Xº e Yº de la secuencia original

<X,Y,Z> — Secuencia final <Xº, Yº, Zº>

Cota inferior para algoritmos de ordenación

- Cualquier permutación de los n elementos debe aparecer como hoja del árbol
- Hay $n!$ permutaciones posibles
- La profundidad o altura máxima de una hoja determina en nº de comparaciones en el peor caso
- Nos interesaría diseñar un algoritmo cuyo árbol tuviera la profundidad mínima
- Una cota inferior de la altura de árbol en el peor caso es una cota inferior del nº de comparaciones para cualquier algoritmo de ordenación por comparación

Cota inferior para algoritmos de ordenación

Teorema

Cualquier algoritmo de ordenación por comparaciones necesita $\Omega(n \log n)$ *comparaciones en el peor caso*

Demostración

Tenemos un árbol de decisión de altura h *con* l *hojas (hay que demostrar que* $h \in \Omega(n \log n)$*)*

- $n! \leq l$, *tiene que haber por lo menos* $n!$ *hojas*
- $l \leq 2^h$, *un árbol binario de altura* h *tiene como mucho* 2^h *hojas*

$$n! \leq l \leq 2^h \quad \Rightarrow \quad n! \leq 2^h$$

$$\log_2(n!) \leq \log_2(2^h) \quad \Rightarrow \quad h \geq \log_2(n!) \in \Omega(n \log n)$$

Demostrar que $\log_2(n!) \in \Omega(n \log n)$ es un ejercicio para casa/práctica