ALGORITMOS Y
ESTRUCTURA DE DATOS III

Sección TQ - 1er Semestre/2018

Análisis de Algoritmos

Prof. Cristian Cappo

28/febrero/2018

¿Que veremos hoy?

- Algoritmos como tecnología
- Análisis de algoritmos y cálculo asintótico
- Análisis de casos: *mejor*, *peor* y *promedio*
- Cálculo de tiempo de ejecución de un programa
- Notaciones asintóticas: O, Θ, Ω
- Ejercicios

Algoritmos como tecnología

- ¿Qué es un algoritmo?
  - Es una **secuencia** de pasos concreta que transforma unos datos de entrada.
  - Es un procedimiento bien definido que toma algún valor o conjunto de valores, como entrada, y produce algún valor o conjunto de valores, como salida [CLRS]
  - *Podemos ver al algoritmo como una **herramienta** que resuelve un determinado problema computacional.*

[CLRS] Cormen-Leiserson-Rivest-Stein. Introduction to Algorithms, 2nd Edition, MIT Press, 2001

Algoritmos como tecnología

Del libro "*Algorithms*" Sedgewick & Wyne. 2011

El término Algoritmo es utilizado en Ciencias de la Computación para describir un método finito, determinista y efectivo para resolver un problema, el cual es apropiado para su implementación como un programa de computadoras.

- **Propiedades de un algoritmo**
  - Es correcto
  - Tiene pasos concretos
  - Bien definido (no ambiguo)
  - Es finito
  - Termina

Algoritmos como tecnología

- *¿Por qué no es suficiente encontrar* ***un*** *solo algoritmo para un problema determinado?*
  - RECURSOS LIMITADOS (procesador,memoria, ancho de banda, etc)
  - TIEMPO DE RESPUESTA
- *¿Cómo escoger el algoritmo más conveniente?*
  - Buena codificación
  - Fácil de usar
  - Fácil de implementar
  - Eficiente (*¿qué significa eficiente?*)

Algoritmos como tecnología

- **Eficiencia de un algoritmo**
  - Cuando se escribe la solución a un problema generalmente existen dos cosas inmersas en el diseño:
    - Fácil de entender, codificar y depurar (***Ingeniería de Software***).
    - Uso eficiente de los recursos computacionales (tiempo y espacio) (***Análisis de algoritmos***)

Cuando logro ambas cosas tengo un programa "elegante".

Algoritmos como tecnología

- **Eficiencia de un algoritmo**

¿Una máquina más rápida o
un algoritmo más eficiente?

Algoritmos como tecnología

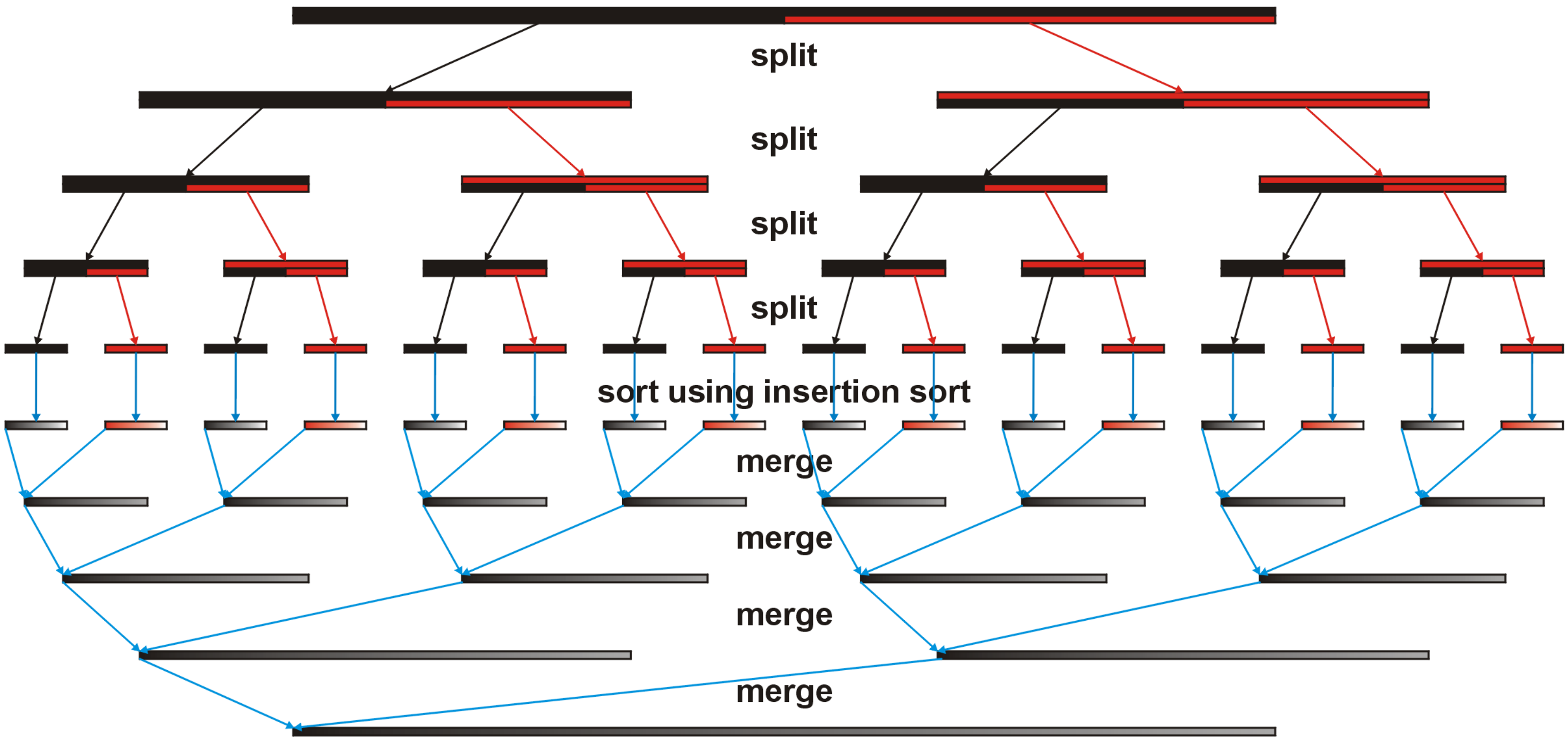
- **Eficiencia de un algoritmo**
  - Suponga el problema de ordenar un arreglo de números.
  - Existen varios algoritmos, nosotros elegiremos dos para comparar:

    Ordenación por inserción (***InsertionSort***)

    Ordenación por mezcla (***MergeSort***).

  Ambos son algoritmos correctos.

Merge Sort



split
split
split
split
sort using insertion sort
merge
merge
merge
merge

Algoritmos como tecnología

- **Eficiencia de un algoritmo** (cont.)
  - Suponga que tiene dos máquinas **A** y **B**
    - **A** puede ejecutar 1.000.000.000 (10^9) instrucciones por segundo
    - **B** puede ejecutar 10.000.000 (10^7) instrucciones por segundo (*Note que es 100 veces más lenta que A*)
    - c_1 es 2 y c_2 es 50 (constantes multiplicativas para ***InsertSort*** y ***MergeSort*** respectivamente). ¿Qué significa esto?
  - Se quiere ordenar 1 millón de números (n=1.000.000)... la pregunta es *¿quién ganará?*

Algoritmos como tecnología

Eficiencia de un algoritmo (cont.)

- *InsertSort* toma alrededor de un tiempo de $c_1 n^2$ en ordenar n items.
- *MergeSort* toma alrededor de un tiempo de $c_2 n \log_2 n$ (asumimos logaritmo en base 2)
- c_1 y c_2 son factores constantes.

Consideramos que el tiempo es una función del tamaño de la entrada, normalmente lo denominados T.

Así el algoritmo de *InsertSort* toma un tiempo $T(n) = c_1 n^2$

Algoritmos como tecnología

InsertSort en la máquina A

$$\frac{2.(10^6)^2\, instrucciones}{10^9\, instrucciones/segundo} = 2000\, segundos$$

33 minutos, 20 segundos

MergeSort en la máquina B

$$\frac{50.(10^6)\log 10^6\, instrucciones}{10^7\, instrucciones/segundo} \approx 100\, segundos$$

1 minuto, 40 segundos

- ¿Qué pasa cuando n = diez millones? 2d, 7h, 33m, 20s ***vs*** 19m, 22s
- ¿Con cuál algoritmo se quedaría?
- ¿Cuánto tardará *InsertSort* si ahora la máquina A es 10 veces más rápida?
- ¿Cuál es el efecto de la rapidez de la máquina?

Efecto del algoritmo y la rapidez de la máquina

Tiempo de ejecución del algoritmo - *T(n)*	**Máximo tamaño solucionable en 1 segundo**		
	Computador actual	**100 veces más rápida**	**1000 veces más rápida**
n	N_0=100 millones	$100N_0$	$1000N_0$
100n	N_1=1 millón	$100N_1$	$1000N_1$
n^2	N_2=10000	$10N_2$	$31.6N_2$
n^3	N_3=464	$4.64N_3$	$10N_3$
2^n	N_4=26	$N_4 + 6.64$	$N_4 + 9.97$

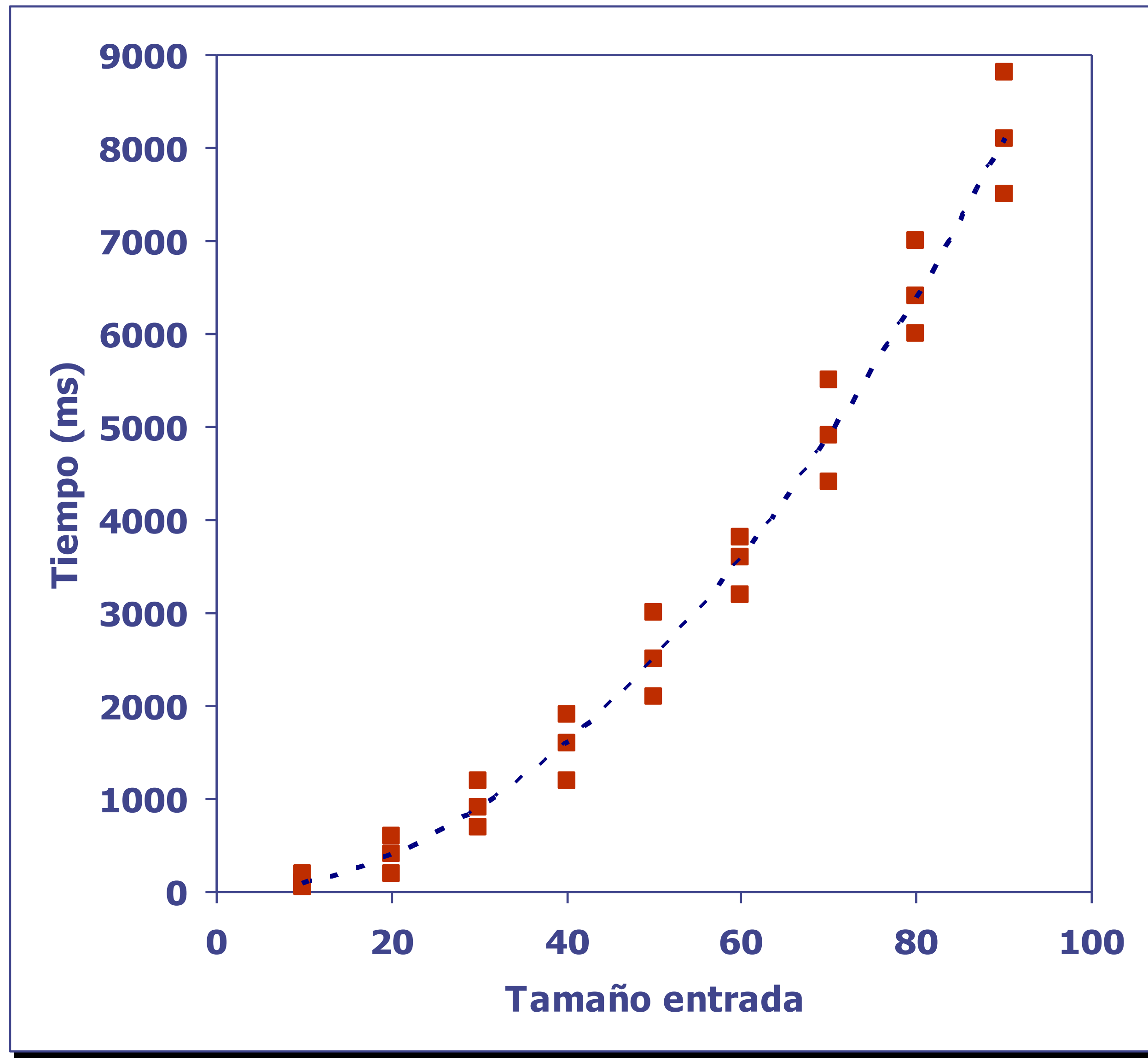
Algoritmos como tecnología

- ¿Como escojo el mejor algoritmo?
  - Medir el tiempo de respuesta.. pero dependo de:
    - Del subconjunto de pruebas escogidas
    - Del ordenador donde hago las pruebas
    - Del lenguaje de programación utilizado
    - Del compilador utilizado
    - De la calidad de la programación
  - Para hacer esto tengo que codificar/programar y en ocasiones esto no es trivial, no es sencillo.

Algoritmos como tecnología

¿Cómo elijo un algoritmo?

- *Escribo el programa*
- *Ejecuto el programa*
- *Mido el tiempo*
- *Grafico el resultado*
- *Encuentro la tendencia*



Algoritmos como tecnología

¿Cómo escojo el mejor algoritmo?

Otra alternativa:

- Predecir el comportamiento del algoritmo sin necesidad de implementarlo.
- Para ello nos ayudará el ***análisis de algoritmo*** y el ***cálculo asintótico***.

Análisis de algoritmos y cálculo asintótico

- El **análisis de algoritmos** estudia desde el punto de vista teórico, los recursos computacionales que necesita la ejecución de un programa de computadora: *su eficiencia.*

  Aparte del tiempo de cómputo, ¿Cuáles son otros recursos?

- Es un técnica de estimación. Es particularmente útil para determinar si vale o no la pena implementar un algoritmo en particular.
- El **cálculo asintótico** es una técnica de caracterización de los algoritmos para poder luego compararlos.

Análisis de algoritmos

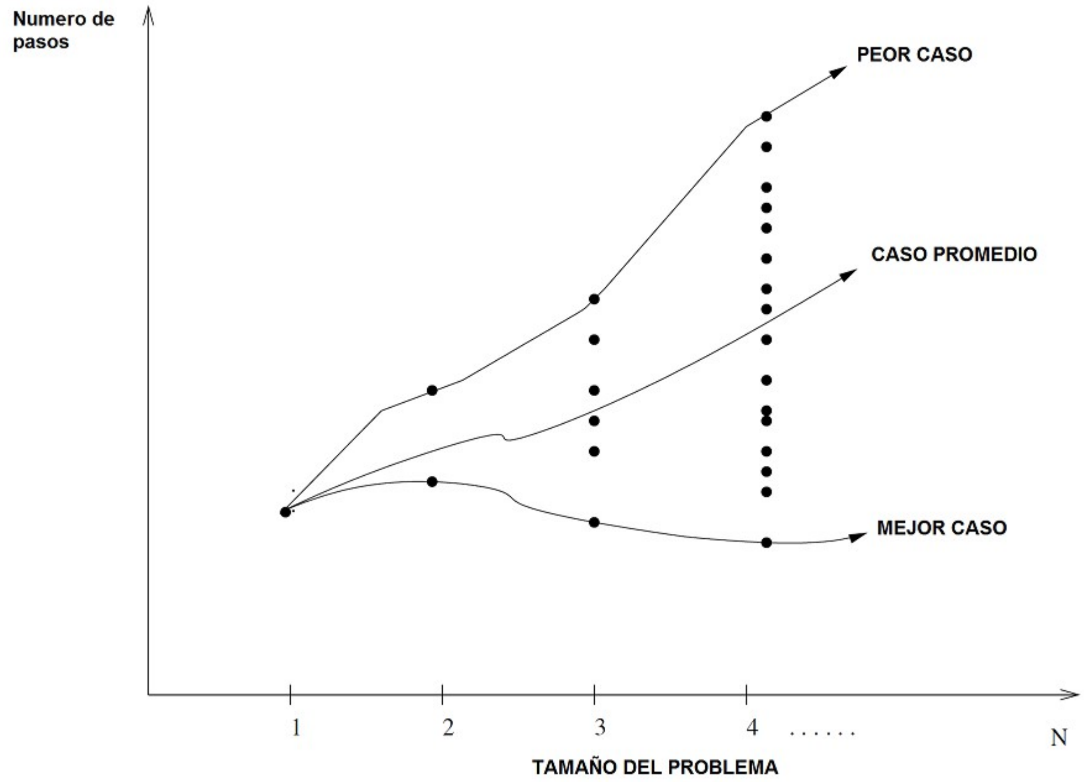
- Utilizamos una descripción de alto nivel para el algoritmo (*generalmente*)
- Caracterizamos el tiempo de ejecución como una función al tamaño de la entrada *N*, **T*(N)***
- Se toma en cuenta todos las posibles entradas (*instancias del problema*)
- Permite evaluar la velocidad del algoritmo independientemente del entorno de HW o SW.

Análisis de algoritmos

- El modelo RAM (*Random Access Machine*)
  - Las instrucciones son ejecutadas una tras otra sin concurrencia. Es un modelo secuencial.
  - Un banco de celdas de memoria potencialmente ilimitado.
  - Las celdas de memoria están numeradas y accederlas toma una unidad de tiempo constante.
  - Las operaciones son similares a cualquier computadora tradicional.
  - Cada operación simple (+,*,-,=,jump,call,..) toma exactamente 1 unidad de tiempo.
- **Importante**: esto es una simplificación de la realidad *¿Por qué le parece?*

¿Se imagina otro modelo diferente?

Análisis de algoritmos



Tipos de análisis

- Peor caso (usual) (*worst*)
  - Es la función definida por el máximo número de pasos que toma cualquier instancia de tamaño N
- Caso promedio (*average*)
  - Es la función definida por el promedio de número de pasos que toma cualquier instancia de tamaño N.
    Implica estudio estadístico de la distribución de las instancias del problema analizado.

    Es el mejor estudio aunque su cálculo es más complejo.
- Mejor caso (*best*)
  - Es la función definida por el mínimo número de pasos que toma cualquier instancia de tamaño N. Es el caso más engañoso

Ejemplo de los casos (1)

- Considere el algoritmo de buscar un elemento en un arreglo desordenado:
  - ¿Cuál es el peor caso?
  - ¿Cuál es el mejor caso?
  - ¿Cuál el el caso promedio?
  - ¿Cómo calcularía el tiempo para cada caso?

[16, 20, 30, 50, 26, 17, 59, 89, 78, 65, 7, 8, 1, 4, 5]

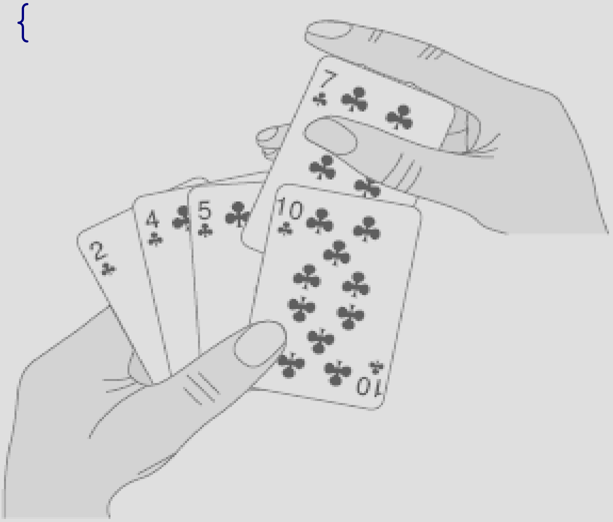
Buscamos el 16

Buscamos el 5

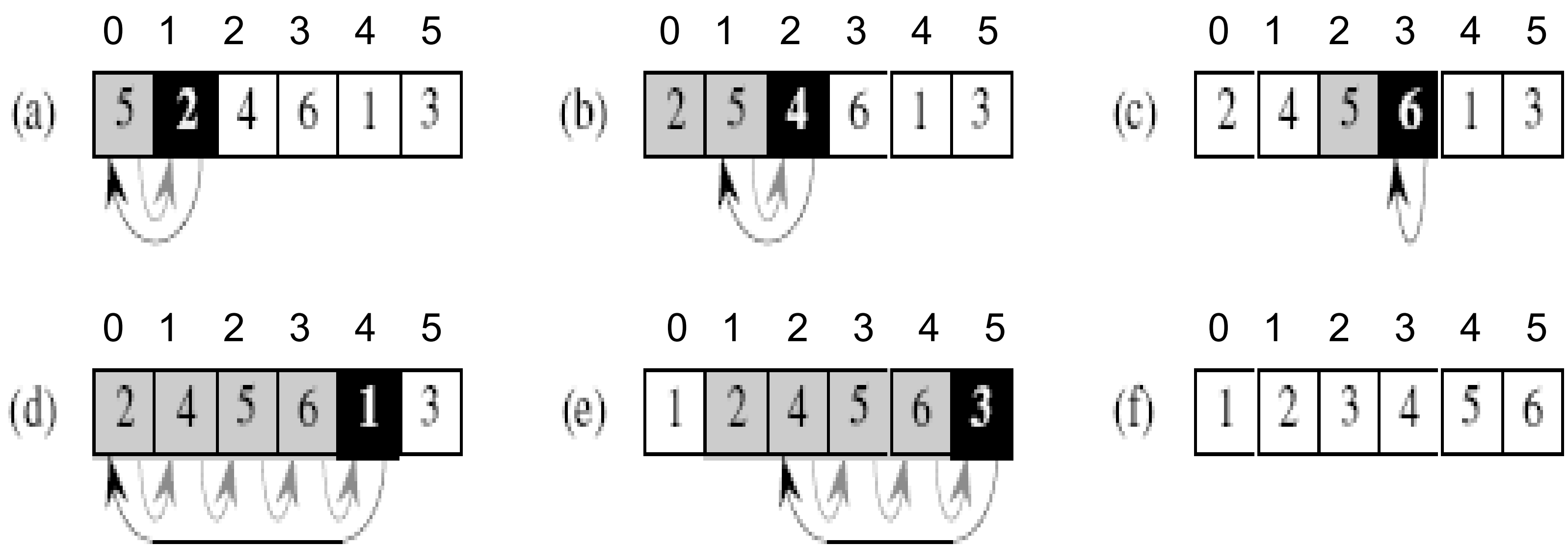
Ejemplo de casos (2)

Considere ahora el algoritmo de ordenación por inserción. ¿Cuál es el mejor, peor y caso promedio?

```
int key, i;

for ( int j=1; j < A.length; j++ ) {

   key = A[j];
   i   = j-1 ;

   while ( i >= 0 && A[i] > key ) {
      A[i+1] = A[i];
      --i;
   }

   A[i+1] = key ;
}
```



Funcionamiento de insertSort



A = { 5,2 4,6,1,3}

Ejemplo de los casos *(InsertSort)*

- Mejor caso del algoritmo
  - Cuando no se ingresa al ciclo mientras y ocurre cuando?
- Caso promedio
  - Cuando se ingresa la mitad de las veces... aunque depende..¿de qué?
- Peor caso
  - Cuando siempre se ingresa.

Estimación de tiempo

Calcular el tiempo de ejecución de un algoritmo

- Algunas consideraciones sobre las operaciones
  - Una asignación = 1 OE (Operación elemental)
  - Una comparación = 1 OE
  - Acceso a arreglo = 1 OE
  - Llamada a función= 1 OE
  - Una operación simple = 1 OE (resta, suma, .., etc)
  - Consideramos todas las operaciones del mismo costo (*aunque no es real, ¿porqué?*)

Calcular el tiempo de ejecución de un algoritmo

Inspeccionando el código podemos determinar el número de operaciones elementales.

Analicemos el siguiente algoritmo, encontrar el máximo elemento de en un arreglo (*Recibimos el vector A y su tamaño n*)

Algoritmo *findMax*(*A*, *n*)

	# Operaciones
Maximo $\leftarrow A[0]$	*2*
for $i \leftarrow 1$ **to** $n - 1$ **do**	$1 + n$
if $A[i] > Maximo$ **then**	$2(n-1)$
Maximo $\leftarrow A[i]$	$2(n-1)$ ***(peor caso)***
{ incremento del contado i}	$2(n-1)$
return *Maximo*	*2*
Total	$7n - 2$

Cálculo del tiempo de ejecución de un algoritmo

- El algoritmo ***findMax*** ejecuta *7n-2* operaciones en el peor caso. Definimos entonces :
  - *a* = Tiempo que toma la operación elemental más rápida
  - *b* = Tiempo que toma la operación elemental más lenta
- Decimos que *T(n)* es el tiempo para ***findMax***. Entonces

$$\boldsymbol{a}(7\mathrm{n}-2) \leq \boldsymbol{T}(\boldsymbol{n}) \leq \boldsymbol{b}(7\mathrm{n}-2)$$

De aquí concluimos que el tiempo de ejecución *T(n)* esta limitado por dos funciones lineales.

Cálculo del tiempo de ejecución de un algoritmo

- El cambio del entorno de HW o SW
  - Afecta a *T(n)* en un factor constante, pero
  - NO afecta la tasa de crecimiento de *T(n)*
- Por tanto la tasa de crecimiento lineal del tiempo de ejecución de *T(n)* es una propiedad intrínseca del algoritmo ***findMax***

Cálculo asintótico

- El cálculo asintótico[1] de un algoritmo me permite clasificarlo de acuerdo a su “tasa de crecimiento” (*growth rate*).
- Ignoramos las constantes que puedan tenerse y nos concentramos en el grado dominante. De esta forma puedo comparar algoritmos de una forma más sencilla.

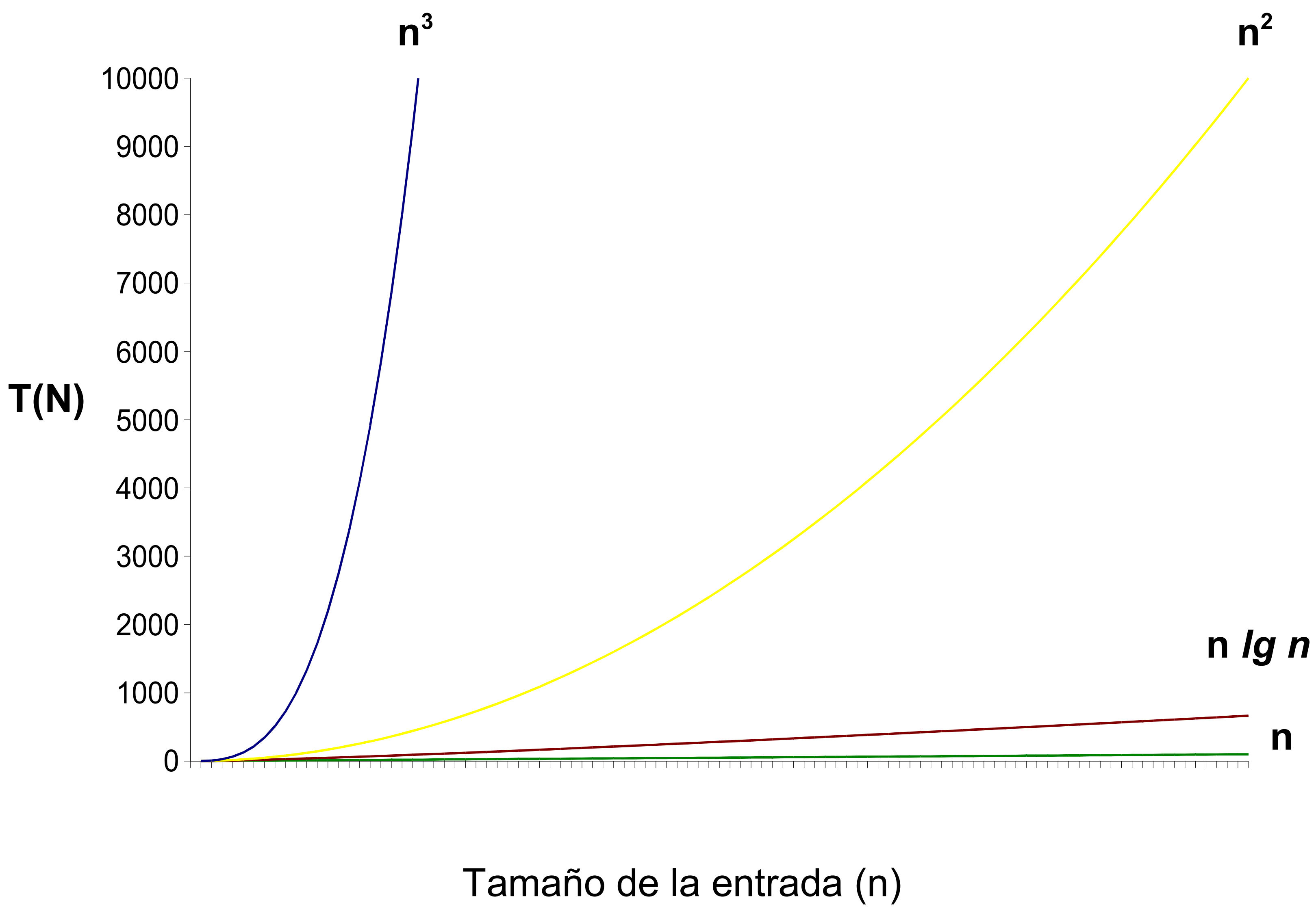
[1] *Propuesto por **Donal Knuth** en su libro “El arte de programar computadores” (primera edición : 1968)*

Tasas de crecimiento usuales en orden creciente

Clase	**Nombre de la función**
c	Constante
$lg\ n$	Logarítmica
$lg^{2}\ n$	Logarítmica al cuadrado
n	Lineal
$n\ lg\ n$	"n lg n"
n^2	Cuadrática
n^3	Cúbica
$n^m,\ m=0,1,2,3,..$	Polinomial
$c^{n},\ c>1$	Exponencial
$n!$	Factorial

Crecimiento

Crecimiento de algunas funciones



Tasas de crecimiento

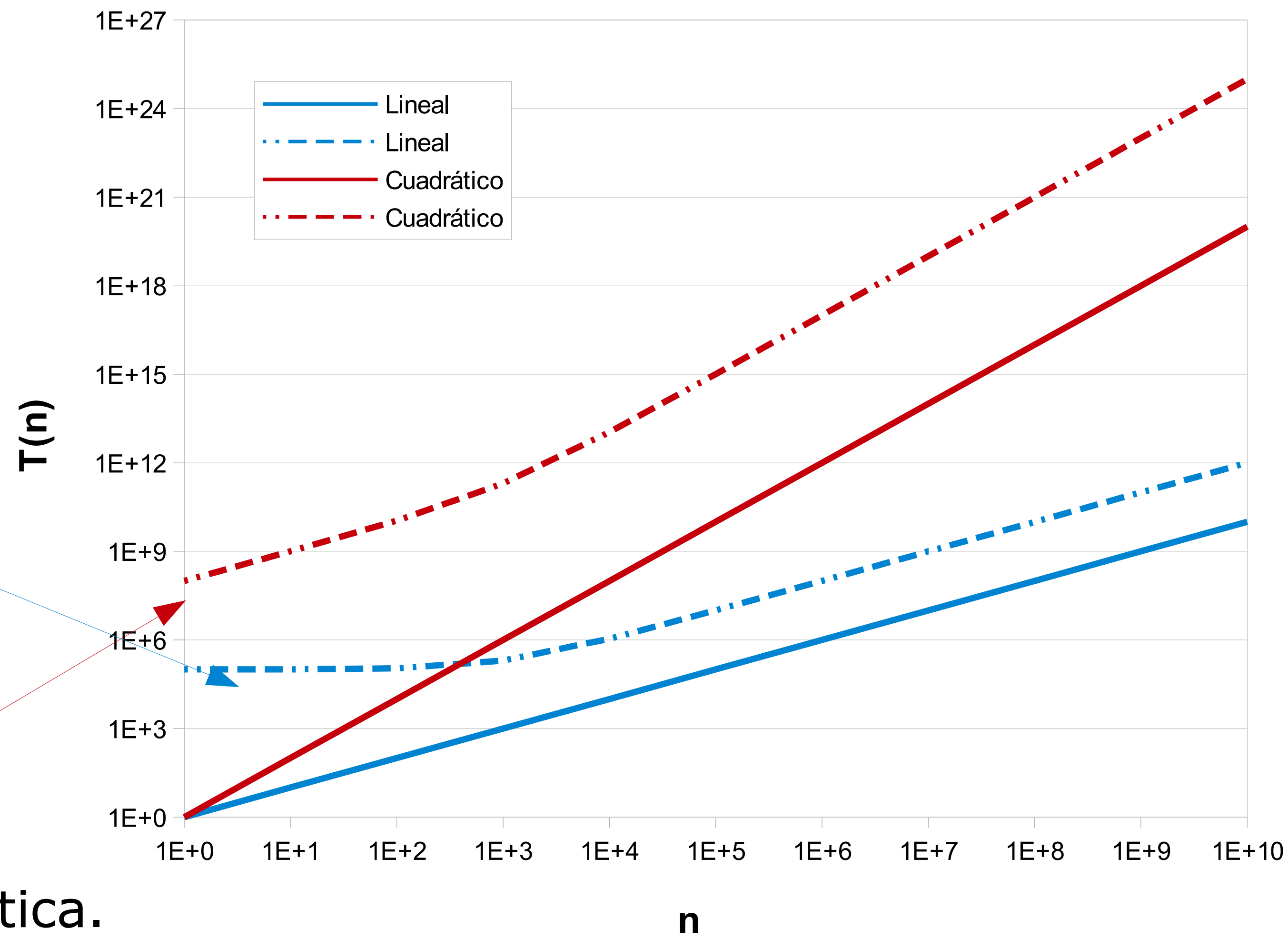
- No es afectada por factores constantes o por términos de menor grado.
- **Ejemplos:**

$$10^2 n + 10^5$$

tiene una tasa de crecimiento lineal

$$10^5 n^2 + 10^8 n$$

tiene una tasa de crecimiento cuadrática.



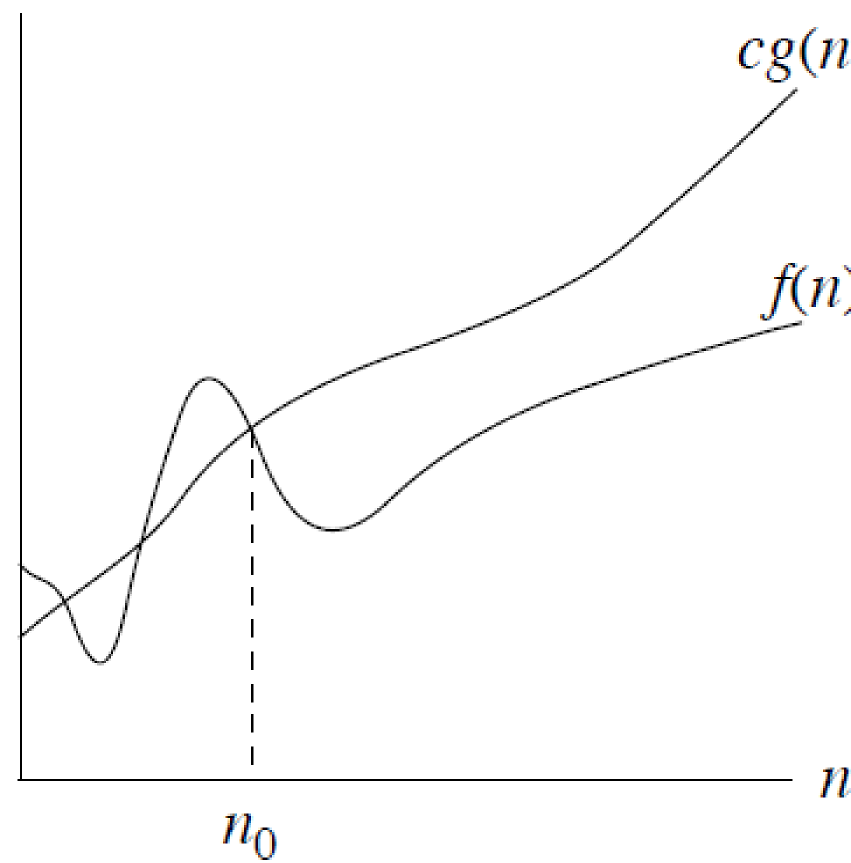
Análisis asintótico

- Para realizar el análisis debemos:
  - Encontrar el número de operaciones en el peor caso
  - Expresar esta función en una notación asintótica (por el momento en la notación *O (o grande*)
- Ejemplo:
  - Nosotros determinamos que ***findMax*** ejecuta en a lo sumo *7n-2* operaciones primitivas.
  - Entonces decimos que el algoritmo ***findMax*** "esta en ***O(n)***"

Notación *O* (*O grande*) *(cota superior)*

Dada las funciones *f(n)* y *g(n)*, se dice que f(n) "está en $O(g(n))$" si existen las constantes positivas c y n_0 tal que

$$f(n) \leq cg(n)$$
$$para\ n \geq n_0$$



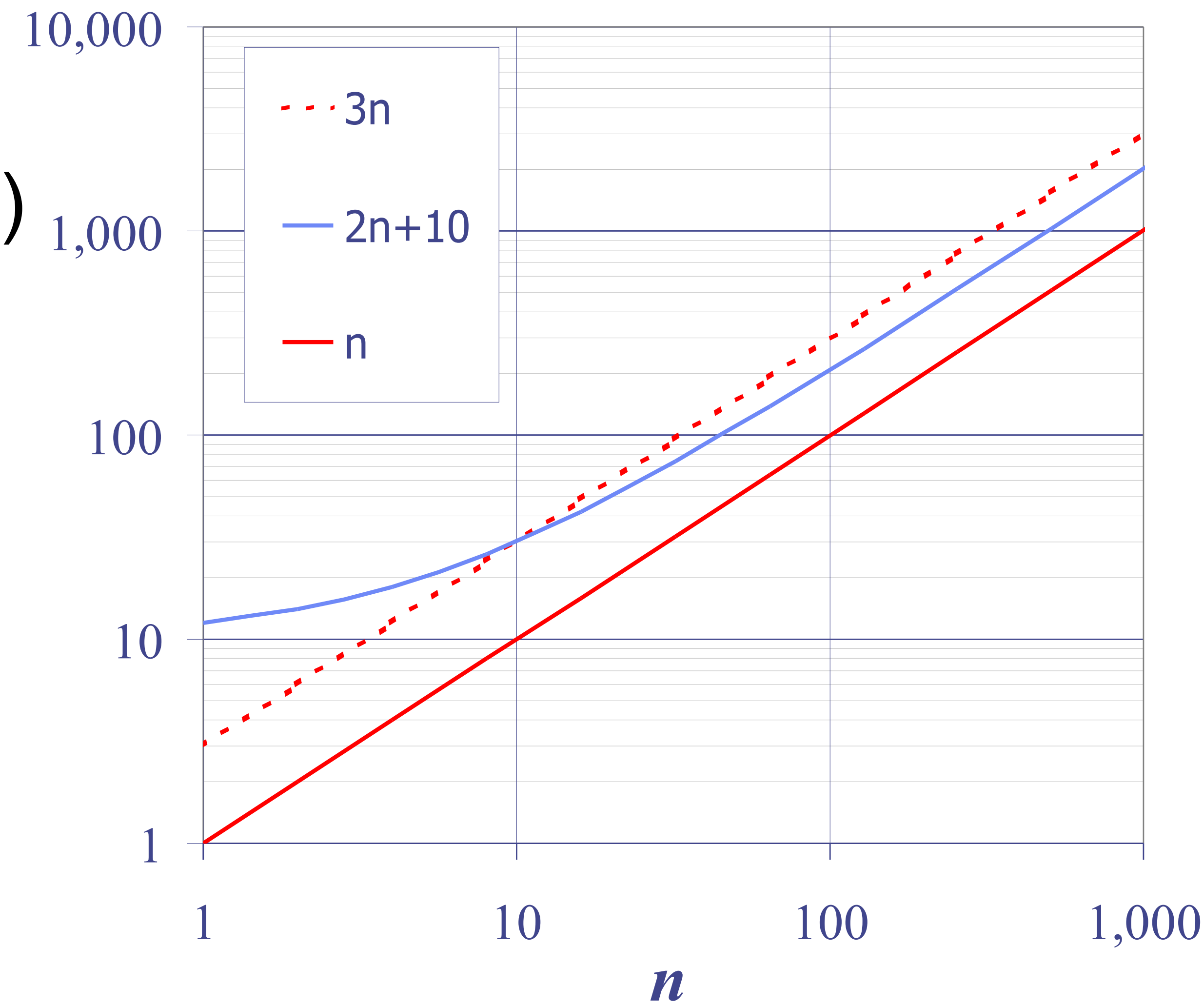
Notación *O*

Ejemplo 1:

$$2n + 10 \quad esta\ en \quad O(n)$$

$$2n + 10 \leq cn$$
$$(c-2)n \geq 10$$
$$n \geq 10/(c-2)$$

$$c = 3 \ y \ n_0 = 10$$



Notación *O*

Ejemplo 2:

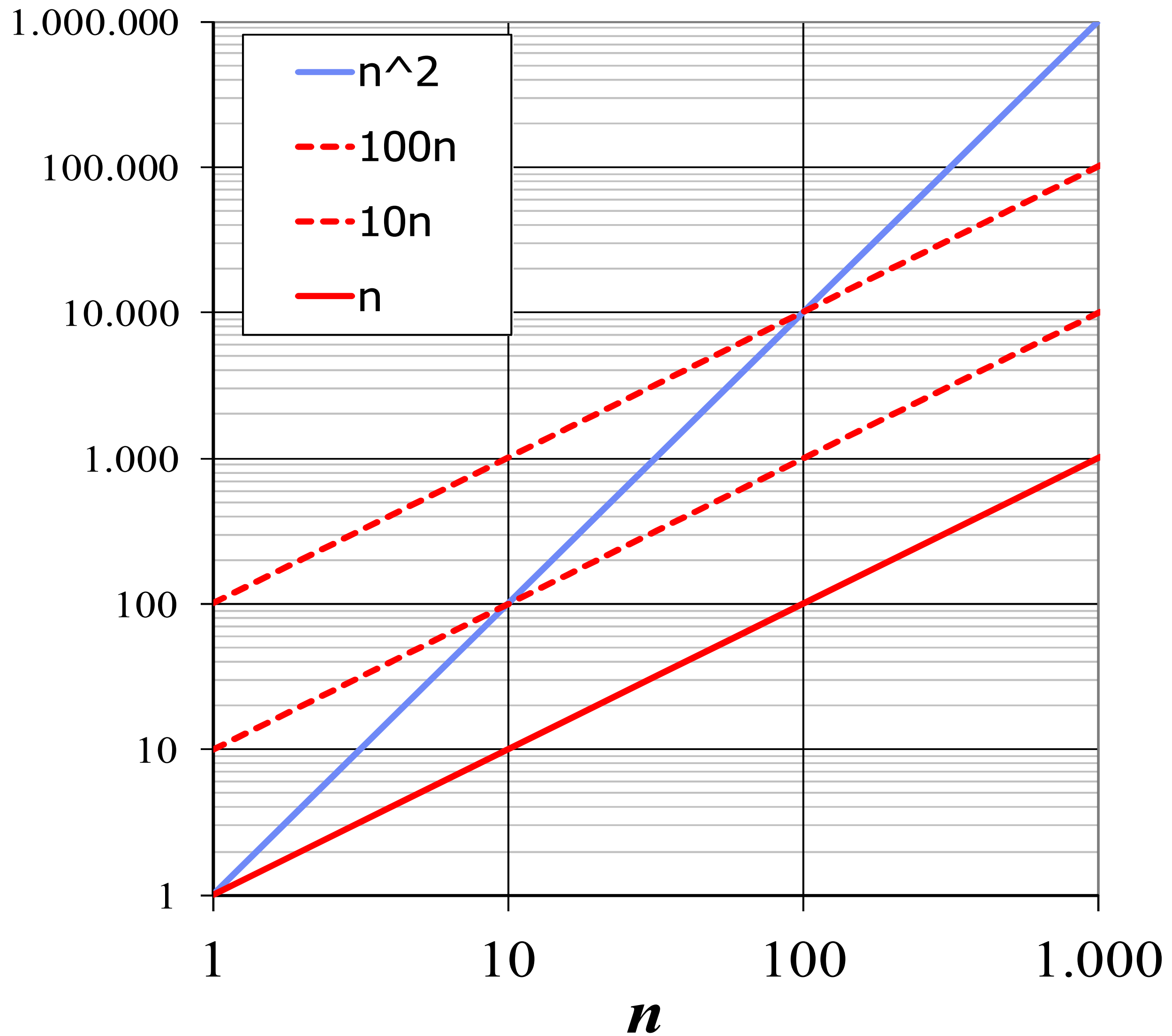
La función

$$n^2 \quad no\, esta\, en \quad O(n)$$

$$n^2 \leq cn$$

$$n \leq c$$

La desigualdad no puede ser satisfecha ya que c debe ser una constante



Ejemplos de Notación O-grande

- $7n\text{-}2$ esta en O(n)

- $3n^3 + 20n^2 + 5$ esta en O(?)

- $3 \lg n + \lg \lg n$ esta en O(?)

Reglas de la notación O-grande

- Si es $f(n)$ un polinomio de grado d, entonces $f(n)$ estan en $O(n^d)$, esto implica:
  - Extraer los términos de menor orden
  - Extraer los factores constantes
- Utilizar la clase de función más ajustada
  - Decir 2n esta en $O(n)$ y no 2n esta en $O(n^2)$
- Utilizar la expresión simple de la clase
  - Decir $3n + 5$ esta en $O(n)$
    y NO $3n+5$ esta en $O(3n)$

Otras notaciones asintóticas

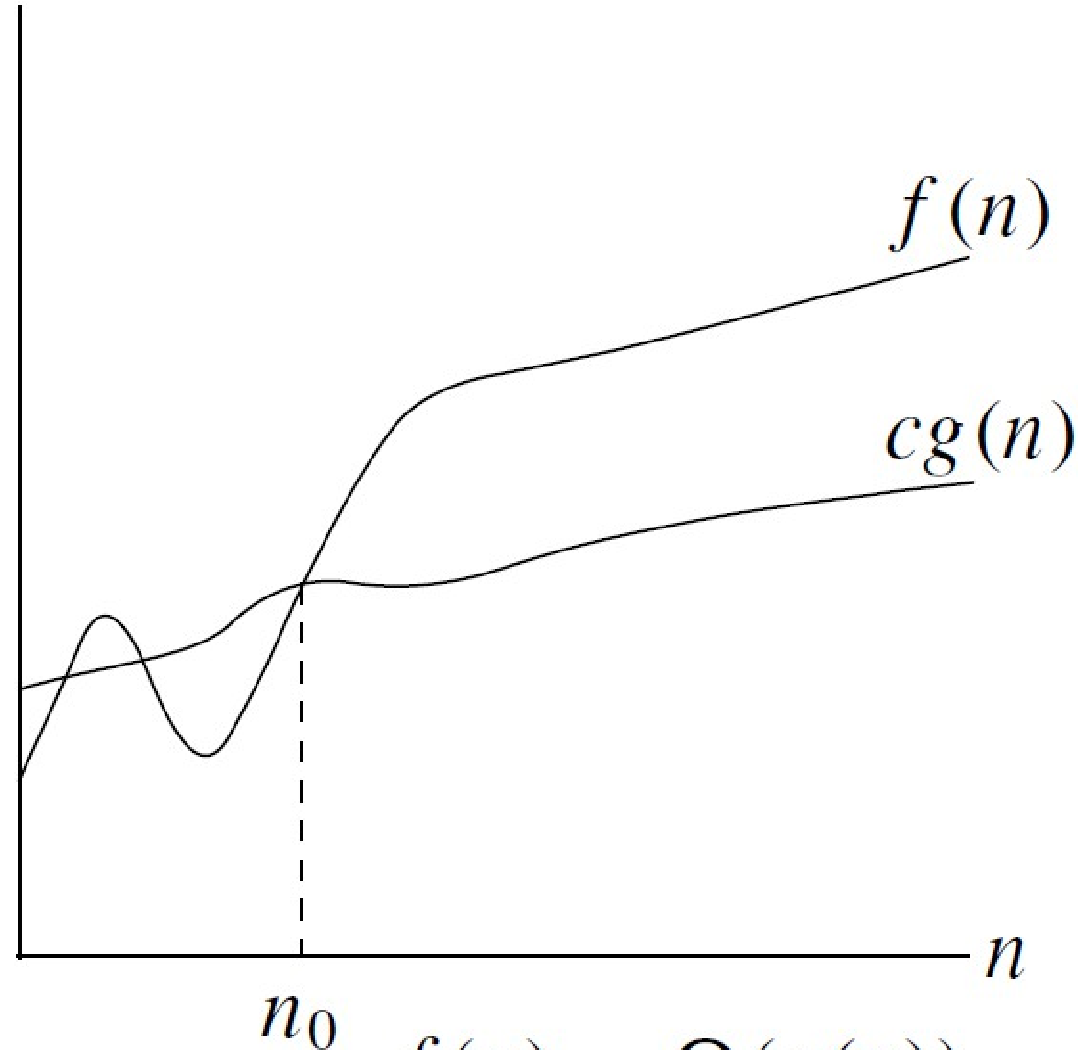
Notación Ω (Omega) *(cota inferior)*

f(n) esta en Ω*(g(n))* si existe una constante c y un entero constante n_0 >= 1 tal que

f(n) >= c.g(n) para n >= n_0

Por ejemplo

$$\sqrt{n} = \Omega(\log n)$$
$$10n^2 = \Omega(n^{3/2})$$



$f(n) = \Omega(g(n))$

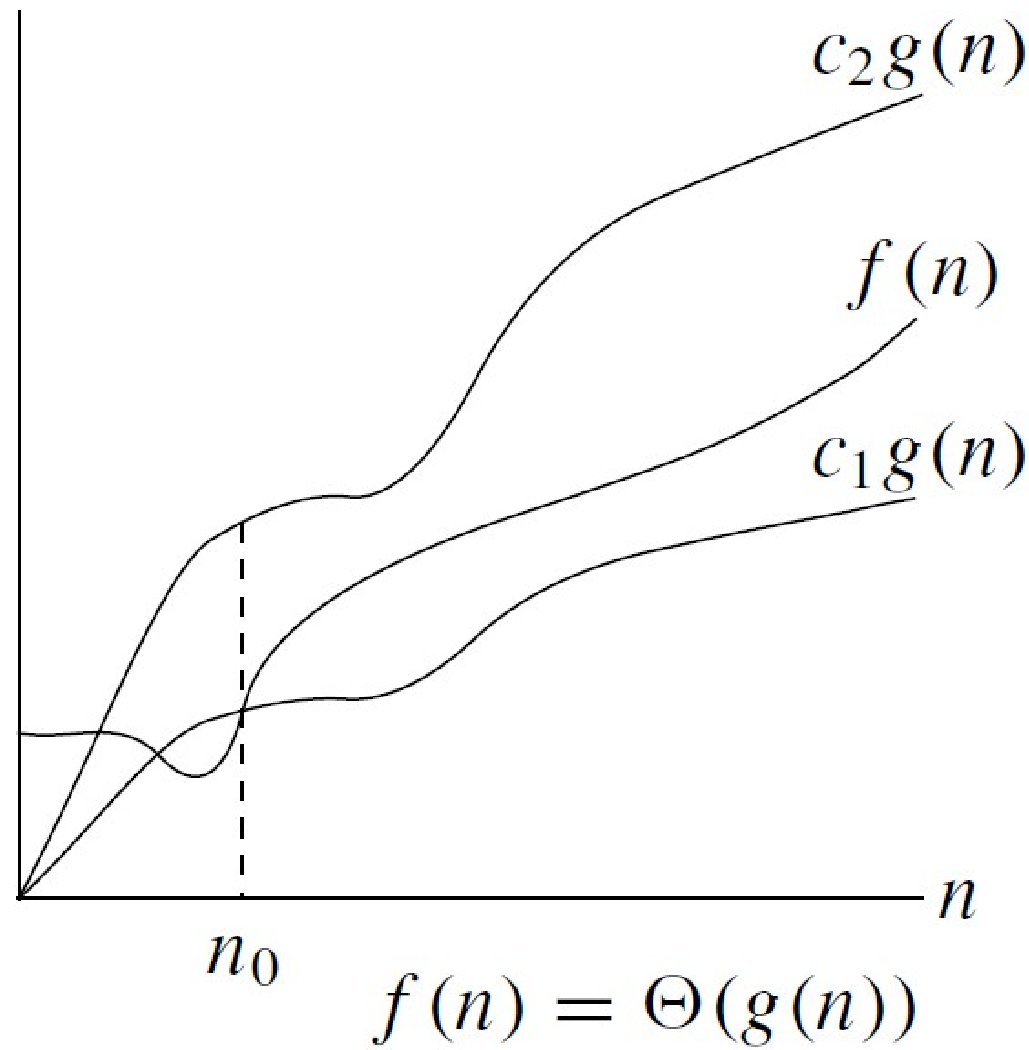
Otras notaciones asintóticas

Notación Θ (theta)

f(n) esta en Θ*(g(n))* si existen las constantes c1 y c2 y un entero constante n_0 >= 1 tal que

c1.g(n) <= f(n) <= c2.g(n)

para n >= n_0



Notaciones asintóticas, una visión intuitiva

- O grande
  - *f(n)* esta en *O(g(n))* si *f(n)* es asintóticamente **menor o igual** que *g(n)*
- Ω Grande
  - *f(n)* esta en Ω*(g(n))* si *f(n)* es asintóticamente **mayor o igual** a *g(n)*
- Θ Grande
  - f(n) esta en Θ*(g(n))* si *f(n)* es asintóticamente **igual** a *g(n)*

Ejercicio #1

Calcule el t(n) y O() para cada uno de los trozos de código mostrados abajo

Ejercicio #1

```
sum = 0;
for ( i=1; i<=n ; i++ )
  for (j=1; j<=n; j++ )
       sum++;
```

Ejercicio #2

```
sum  = 0;
for (k=1; k<=n; k*=2)
   for (j=1; j<= n; j++)
       sum++;
```

Ejercicio #3

```
sum  = 0;
for (i=1; i<=n; i++)
   for (j=1; j<=i; j++)
       sum++;
```

Ejercicio #4

```
sum = 0;
while (n>=1) {
   sum++;
   n=n/2;
}
```

Ejercicio #5

```
sum = 0;
while (n >= 1) {
    sum++;
    n=n/3;
}
```

Ejercicio #6

```
sum = 0;
for (i=n; i>=1; i/=2)
   for (j=1;j<=i;j++)
      sum++;
```

Ejercicio #2

Demostrar la veracidad de las siguientes expresiones (aplique las definiciones)

$$2^{n+1} = \Theta(2^n)?$$

$$(x+y)^2 = O(x^2+y^2)?$$

Ejercicio #3 - completar

Función	= o $\neq$		c	n_0
$3n^2 - 100\,n$		$O(n^2)$		
$3n^2 - 100\,n$		$O(n^3)$		
$3n^2 - 100\,n$		$O(n)$		

$3n^2 - 100\,n$		$\Omega(n^2)$		
$3n^2 - 100\,n$		$\Omega(n^3)$		
$3n^2 - 100\,n$		$\Omega(n)$		

$3n^2 - 100\,n$		$\Theta(n^2)$
$3n^2 - 100\,n$		$\Theta(n^3)$
$3n^2 - 100\,n$		$\Theta(n)$

Ejercicio #3 - Soluciones

Función	$=$ o $\neq$		c	n_0
$3n^2 - 100\ n$	$=$	$O(n^2)$	3	1
$3n^2 - 100\ n$	$=$	$O(n^3)$	1	1
$3n^2 - 100\ n$	$\neq$	$O(n)$	N/A	N/A

$3n^2 - 100\ n$	$=$	$\Omega(n^2)$	2	>100
$3n^2 - 100\ n$	$\neq$	$\Omega(n^3)$	N/A	N/A
$3n^2 - 100\ n$	$=$	$\Omega(n)$	cualquiera	100c

$3n^2 - 100\ n$	$=$	$\Theta(n^2)$
$3n^2 - 100\ n$	$\neq$	$\Theta(n^3)$
$3n^2 - 100\ n$	$\neq$	$\Theta(n)$

- Aproveche y resuelva los ejercicios planteados. No es obligatorio pero es importante.
- LEA las referencias mencionadas en el Plan Semestral (al menos el capítulo 5 de [Weiss2000]).

Siguiente :
Análisis de algoritmos recursivos