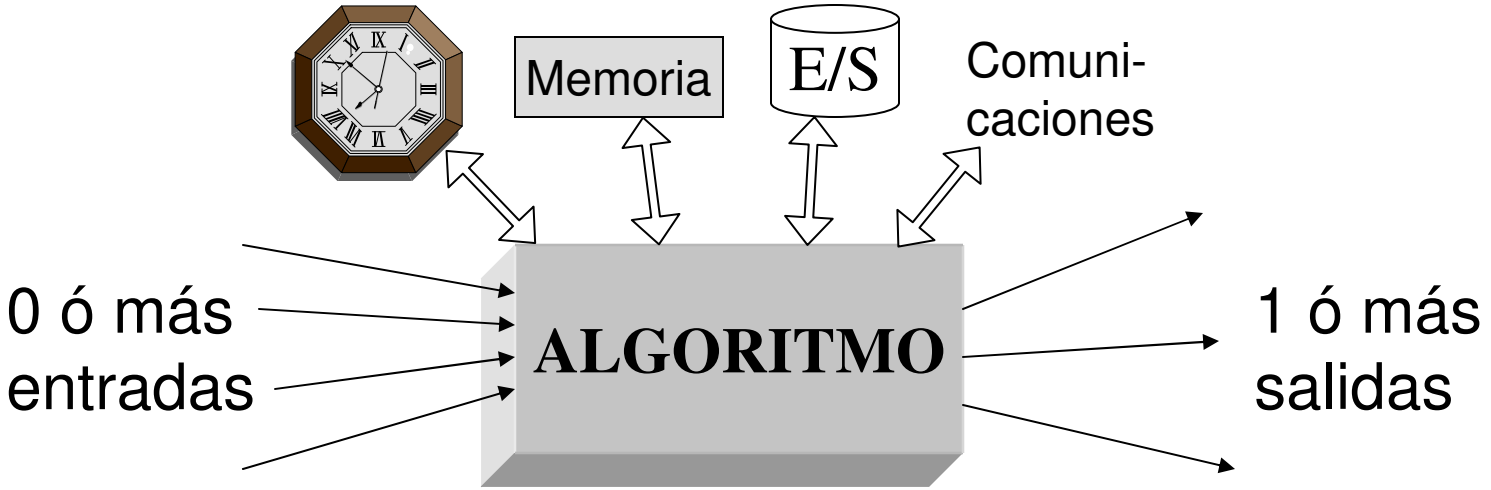
Análisis de algoritmos.

- Introducción.
- Notaciones asintóticas.
- Ecuaciones de recurrencia.
- Ejemplos.

Introducción

- **Algoritmo:** Conjunto de reglas para resolver un problema. Su ejecución requiere unos recursos.



- Un algoritmo es mejor cuantos menos recursos consuma. Pero....
- **Otros criterios:** facilidad de programarlo, corto, fácil de entender, robusto...

Introducción

- **Criterio empresarial:** Maximizar la eficiencia.
- **Eficiencia:** Relación entre los recursos consumidos y los productos conseguidos.
- **Recursos consumidos:**
  - **Tiempo de ejecución.**
  - **Memoria principal.**
  - Entradas/salidas a disco.
  - Comunicaciones, procesadores,...
- **Lo que se consigue:**
  - Resolver un problema de forma exacta.
  - Resolverlo de forma aproximada.
  - Resolver algunos casos...

Introducción

- **Recursos consumidos.**

  Ejemplo. ¿Cuántos recursos de tiempo y memoria consume el siguiente algoritmo sencillo?

```
i:= 0
a[n+1]:= x
repetir
      i:= i + 1
hasta a[i] = x
```

- **Respuesta:** Depende.
- ¿De qué depende? De lo que valga **n** y **x**, de lo que haya en **a**, de los tipos de datos, de la máquina...

Introducción

- En general los recursos dependen de:
  - **Factores externos.**
    - El ordenador donde lo ejecutemos: 286, Pentium III, Cray,...
    - El lenguaje de programación y el compilador usado.
    - La implementación que haga el programador del algoritmo. En particular, de las estructuras de datos utilizadas.
  - **Tamaño de los datos de entrada.**
    - Ejemplo. Calcular la media de una matriz de NxM.
  - **Contenido de los datos de entrada.**
    - Mejor caso. El contenido favorece una rápida ejecución.
    - Peor caso. La ejecución más lenta posible.
    - Caso promedio. Media de todos los posibles contenidos.
- Los factores externos no aportan información sobre el algoritmo.

Introducción

- **Conclusión**: Estudiar la variación del tiempo y la memoria necesitada por un algoritmo respecto al tamaño de la entrada y a los posibles casos, de forma aproximada (y parametrizada).
- **Ejemplo**. Algoritmo anterior. Conteo de instrucciones.
  - Mejor caso. Se encuentra x en la 1ª posición:

    $Tiempo(N) = a$
  - Peor caso. No se encuentra x:

    $Tiempo(N) = b \cdot N + c$
  - Caso medio. Se encuentra x con probabilidad P:

    $Tiempo(N) = b \cdot N + c - (d \cdot N + e) \cdot P$

Introducción

Normalmente usaremos la notación T(N)=..., pero **¿qué significa T(N)?**

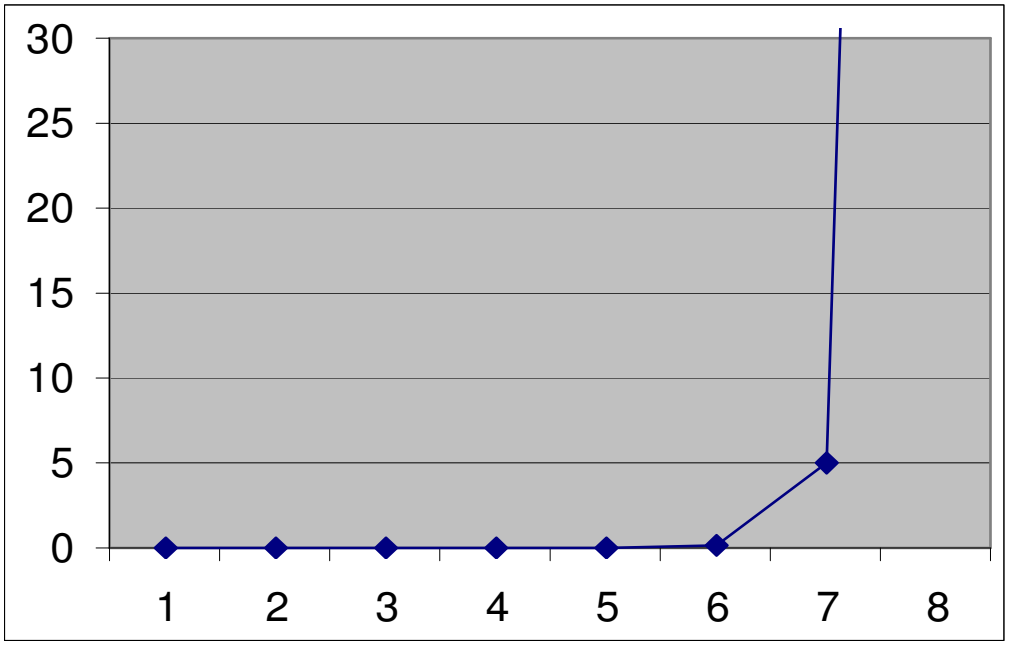
- Tiempo de ejecución en segundos. T(N) = bN + c.
  - Suponiendo que b y c son constantes, con los segundos que tardan las operaciones básicas correspondientes.
- Instrucciones ejecutadas por el algoritmo. T(N) = 2N + 4.
  - ¿Tardarán todas lo mismo?
- Ejecuciones del bucle principal. T(N) = N+1.
  - ¿Cuánto tiempo, cuántas instrucciones,...?
  - Sabemos que cada ejecución lleva un tiempo constante, luego se diferencia en una constante con los anteriores.

Introducción

- **Asignación de tiempos, para el conteo de instrucciones. Algunas reglas básicas.**
  - **Operaciones básicas** (+, -, *, :=,...): Una unidad de tiempo, o alguna constante.
  - **Operaciones de entrada salida**: Otra unidad de tiempo, o una constante diferente.
  - **Bucles FOR**: Se pueden expresar como un sumatorio, con los límites del FOR.
  - **IF** y **CASE**: Estudiar lo que puede ocurrir. Mejor caso y peor caso según la condición. ¿Se puede predecir cuándo se cumplirán las condiciones?
  - **Llamadas a procedimientos**: Calcular primero los procedimientos que no llaman a otros.
  - **Bucles WHILE y REPEAT**: Estudiar lo que puede ocurrir. ¿Existe una cota inferior y superior del número de ejecuciones? ¿Se puede convertir en un FOR?

Introducción

- El análisis de algoritmos también puede ser **a posteriori:** implementar el algoritmo y contar lo que tarda para distintas entradas.
- **Ejemplo.** Programa "prueba.exe":
  - N= 4, T(4)= 0.1 ms
  - N= 5, T(5)= 5 ms
  - N= 6, T(6)= 0.2 s
  - N= 7, T(7)= 10 s
  - N= 8, T(8)= 3.5 min



- ¿Qué conclusiones podemos extraer?
- **Análisis a priori:** Evitamos la implementación, si el algoritmo es poco eficiente. Podemos hacer previsiones. Podemos comparar con otros algoritmos.

Notaciones asintóticas
Definiciones

- El tiempo de ejecución T(n) está dado en base a unas constantes que dependen de factores externos.
- Nos interesa un análisis que sea independiente de esos factores.
- **Notaciones asintóticas:** Indican como crece T, para valores suficientemente grandes (asintóticamente) sin considerar constantes.
- O(T): Orden de complejidad de T.
- Ω(T): Orden inferior de T, u omega de T.
- Θ(T): Orden exacto de T.

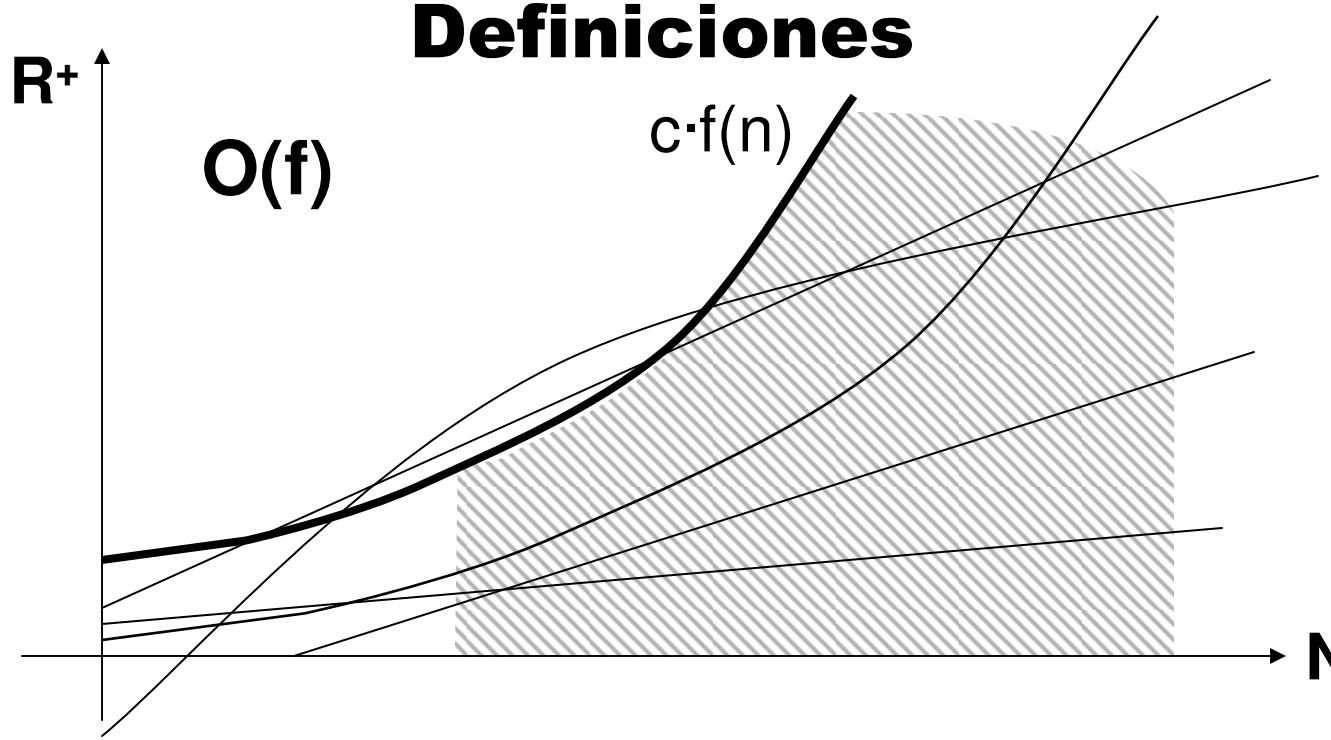
Definiciones

Orden de complejidad de f(n): O(f)

- Dada una función **f: N** $\rightarrow$ **R⁺**, llamamos **orden de f** al conjunto de todas las funciones de **N** en **R⁺** acotadas superiormente por un múltiplo real positivo de **f**, para valores de **n** suficientemente grandes.

$$O(f) = \{ t: N \rightarrow R^+ / \exists\, c \in R^+, \exists\, n_0 \in N, \forall\, n \geq n_0: t(n) \leq c \cdot f(n) \}$$

Definiciones



Ojo:

- **O(f)** es un **conjunto de funciones**, no una función.
- "Valores de n sufic. grandes...": no nos importa lo que pase para valores pequeños.
- "Funciones acotadas superiormente por un múltiplo de f...": nos quitamos las constantes.
- La definición es aplicable a cualquier función de N en R, no sólo tiempos de ejec.

Definiciones

- **Uso de los órdenes de complejidad:** dado un tiempo **t(n)**, encontrar la función **f** más simple tal que **t** $\in$ **O(f)**, y que más se aproxime asintóticamente.
- **Ejemplo**. $t(n) = 2n^2/5 + 3\pi/2$; $t(n) \in O(n^2)$.

- **Relación de orden entre O(..) = Relación de inclusión entre conjuntos.**
  - $O(f) \leq O(g) \Leftrightarrow O(f) \subseteq O(g) \Leftrightarrow$ Para toda $t \in O(f)$, $t \in O(g)$
- Se cumple que:
  - $O(c) = O(d)$, siendo **c** y **d** constantes positivas.
  - $O(c) \subset O(n)$
  - $O(cn + b) = O(dn + e)$
  - $O(p) = O(q)$, si **p** y **q** son polinomios del mismo grado.
  - $O(p) \subset O(q)$, si **p** es un polinomio de menor grado que **q**.

Definiciones

Orden inferior u omega de f(n): Ω(f)

- Dada una función **f: N** $\rightarrow$ **R⁺**, llamamos **omega de f** al conjunto de todas las funciones de **N** en **R⁺** acotadas **inferiormente** por un múltiplo real positivo de **f**, para valores de **n** suficientemente grandes.

$$\Omega(f) = \{ t: N \rightarrow R^+ / \exists\, c \in R^+, \exists\, n_0 \in N, \forall\, n \geq n_0: t(n) \geq c \cdot f(n) \}$$

- La notación omega se usa para establecer cotas inferiores del tiempo de ejecución.
- **Relación de orden:** igual que antes.

Definiciones

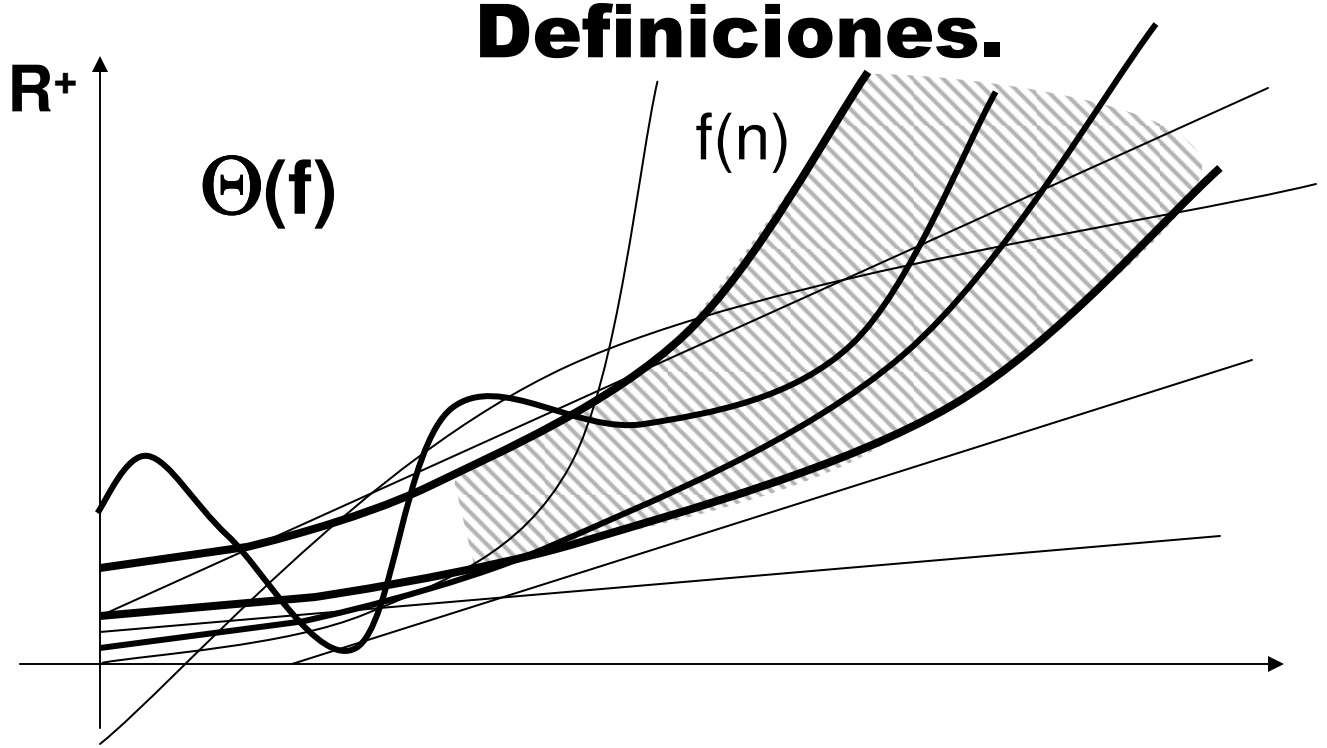
Orden exacto de f(n): $\Theta(f)$

- Dada una función **f: N $\rightarrow$ R$^+$**, llamamos orden **exacto de f** al conjunto de todas las funciones de **N** en **R$^+$** que crecen igual que f, asintóticamente y salvo constantes.

$$\Theta(f) = O(f) \cap \Omega(f) =$$

$$= \{ t: N \rightarrow R^+ / \exists\, c, d \in R^+, \exists\, n_0 \in N, \forall\, n \geq n_0: c \cdot f(n) \geq t(n) \geq d \cdot f(n) \}$$

Definiciones.



- **Ejemplos. ¿Cuáles son ciertas y cuáles no?**

$3n^2 \in O(n^2)$	$n^2 \in O(n^3)$	$n^3 \in O(n^2)$
$3n^2 \in \Omega(n^2)$	$n^2 \in \Omega(n^3)$	$n^3 \in \Omega(n^2)$
$3n^2 \in \Theta(n^2)$	$n^2 \in \Theta(n^3)$	$n^3 \in \Theta(n^2)$
$2^{n+1} \in O(2^n)$	$(2+1)^n \in O(2^n)$	$(2+1)^n \in \Omega(2^n)$
$O(n) \in O(n^2)$	$(n+1)! \in O(n!)$	$n^2 \in O(n!!)$

Definiciones

Notación o pequeña de f(n): o(f)

- Dada una función **f: N** $\rightarrow$ **R+**, llamamos **o pequeña de f** al conjunto de todas las funciones de **N** en **R+** que crecen igual que f asintóticamente:

$$\mathbf{o}(f) = \{ t: N \rightarrow R^+ / \lim_{n\rightarrow\infty} t(n)/f(n) = 1\}$$

- Esta notación conserva las constantes multiplicativas para el término de mayor orden.
- **Ejemplo**. $t(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$

$$t(n) \in o(a_m n^m) \neq o(n^m)$$

- ¿$o(a_m n^m) \subseteq O(a_m n^m)$? ¿$o(t) \subseteq O(t)$?

Definiciones

- **Ejemplos**. Estudiar t(n) y expresarlo con O, Ω, Θ y o.

```
for i:= 1 to N
   for j:= 1 to N
      suma:= 0
      for k:= 1 to N
         suma:=suma+a[i,k]*a[k,j]
      end
      c[i, j]:= suma
   end
end
```

```
Funcion Fibonacci (N: int): int;
if N<0 then
   error('No válido')
case N of
   0, 1: return N
else
   fnm2=0
   fnm1= 1
   for i:= 2 to n
      fn:= fnm1 + fnm2
      fnm2:= fnm1
      fnm1:= fn
   end
   return fn
end
```

Definiciones

- **Ejemplos**. Estudiar t(n) y expresarlo con O, Ω, Θ y o.

```
A[0, (n-1) div 2]:= 1
key:= 2
i:= 0
j:= (n-1) div 2
cuadrado:= n*n
while key<=cuadrado do
   k:= (i-1) mod n
   l:= (j-1) mod n
   if A[k, l] ≠ 0 then
      i:= (i + 1) mod n
   else
      i:= k
      j:= l
   end
   A[i, j]:= key
   key:= key+1
end
```

```
for i:= 1 to N do
   if Impar(i) then
      for j:= i to n do
         x:= x + 1
      end
      for j:= 1 to i do
         y:= y + 1
      end
   end
end
```

Propiedades de las notaciones asintóticas

- **P1.** Si $f \in O(g)$ y $g \in O(h)$ entonces $f \in O(h)$.
  - Si $f \in \Omega(g)$ y $g \in \Omega(h)$ entonces $f \in \Omega(h)$
  - Ej. $2n+1 \in O(n)$, $n \in O(n^2) \Rightarrow 2n+1 \in O(n^2)$

- **P2.** Si $f \in O(g)$ entonces $O(f) \subseteq O(g)$.
  - ¿Cómo es la relación para los Ω?

- **P3.** Dadas f y g de N en R^+, se cumple:
  - i) $O(f) = O(g) \Leftrightarrow f \in O(g)$ y $g \in O(f)$
  - ii) $O(f) \subseteq O(g) \Leftrightarrow f \in O(g)$

Propiedades de las notaciones asintóticas

- ¿La relación de orden entre O(..) es completa? Dadas f y g, ¿se cumple $O(f) \subseteq O(g)$ ó $O(g) \subseteq O(f)$?

- **P4.** Dadas f y g, de N en R^+, $O(f+g) = O(\max(f, g))$.
  - $\Omega(f+g) = \Omega(\max(f+g))$
  - ¿Y para los $\Theta(f+g)$?
  - ¿Es cierto que $O(f - g) = O(\max(f, -g))$?

- **P5.** Dadas f y g de N en R^+, se cumple:
  - i) $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in R^+ \Rightarrow O(f)=O(g), \Omega(f)=\Omega(g), \Theta(f)=\Theta(g)$
  - ii) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow O(f) \subseteq O(g), \Omega(g) \subseteq \Omega(f)$

Propiedades de las notaciones asintóticas

- **P5.** Ej. ¿Qué relación hay entre $O(\log_2 n)$ y $O(\log_{10} n)$?

- **P6.** Dadas f y g de N en R^+, $O(f)=O(g) \Leftrightarrow \Theta(f)=\Theta(g) \Leftrightarrow f \in \Theta(g) \Leftrightarrow \Omega(f)=\Omega(g)$

- **P7.** Dadas f y g de N en R^+, se cumple:
  - i) $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in R^+ \Rightarrow f \in \Theta(g)$
  - ii) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f \in O(g)$, pero no necesariamt $f \in \Theta(g)$
  - iii) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty \Rightarrow f \in \Omega(g)$, pero no neces. $f \in \Theta(g)$

Notaciones con varios parámetros

- En general, el tiempo y la memoria consumidos pueden depender de muchos parámetros.
- **f: $N^m \rightarrow R^+$** (f: Nx...^{m}..xN $\rightarrow R^+$)
- **Ej.** Memoria en una tabla hash. M(B,n, l, k) = kB+l+n+2kn

Orden de complejidad de $f(n_1, n_2, ..., n_m)$: O(f)

- Dada una función **f: $N^m \rightarrow R^+$**, llamamos **orden de f** al conjunto de todas las funciones de N^m en R^+ acotadas superiormente por un múltiplo real positivo de **f**, para valores de $(n_1, ..., n_m)$ suficientemente grandes.

$$O(f)= \{ t: N^m \rightarrow R^+ / \exists\, c \in R^+, \exists\, n_1, n_2, .., n_m \in N, \forall\, k_1 \geq n_1 ,$$
$$\forall\, k_2 \geq n_2 ,..,\forall\, k_m \geq n_m : t(k_1, k_2, ..., k_m) \leq c \cdot f(k_1, k_2, ..., k_m) \}$$

Notaciones con varios parámetros

- De la misma forma, podemos extender los conceptos de $\Omega(f)$ y $\Theta(f)$, para funciones con varios parámetros.
- Las propiedades se siguen cumpliendo $\rightarrow$ Demostrarlo.
- **Ejemplo**. $T(N) = T(N, a, b) = a \cdot N + b$
  - El tiempo depende del tamaño del problema **N**, y del tiempo de inicialización **b** y de ejecución de un paso **a**.
  - Podemos suponerlos constantes T(N), o variables T(N,a,b).
- ¿Qué relación hay entre los siguientes órdenes?

  $O(n+m)$, $O(n^m)$ $O(n^2)$, $O(n+2^m)$ $O(1)$, $O(n)$

Notaciones condicionales

- En algunos casos interesa estudiar el tiempo sólo para ciertos tamaños de entrada.
- **Ejemplo**. Algoritmo de búsqueda binaria: Si N es potencia de 2 el estudio se simplifica.

Orden condicionado de f(n): O(f | P)

- Dada una función **f: N $\rightarrow$ R⁺**, y **P: N $\rightarrow$ B**, llamamos **orden de f según P** (o condicionado a P) al conjunto:

$$O(f \mid P) = \{ t: N \rightarrow R^+ / \exists\, c \in R^+, \exists\, n_0 \in N, \forall\, n \geq n_0: P(n) \Rightarrow t(n) \leq c \cdot f(n) \}$$

- De igual forma, tenemos $\Omega(f \mid P)$ y $\Theta(f \mid P)$.
- $O(f) = O(f \mid true)$. Para cualquier f y g, $f \in O(g \mid false)$.
- ¿$O(f) \leftrightarrow O(f \mid P)$?

Cotas de complejidad frecuentes

- **Algunas relaciones entre órdenes frecuentes.**

  $O(1) \subset O(\log n) \subset O(n) \subset O(n \cdot \log n) \subset O(n \cdot (\log n)^2) \subset O(n^{1.001...}) \subset O(n^2) \subset O(n^3) \subset ... \subset O(2^n) \subset O(n!) \subset O(n^n)$

- ¿Qué pasa con las omegas? ¿Y con los órdenes exactos?

- El orden de un polinomio $a_n x^n + ... + a_1 x + a_0$ es $O(x^n)$.

- $\sum_{i=1}^{n} 1 = n \in O(n)$; $\sum_{i=1}^{n} i = n(n+1)/2 \in O(n^2)$; $\sum_{i=1}^{n} i^m \in O(n^{m+1})$

- Si hacemos una operación para n, otra para n/2, n/4, ..., aparecerá un orden logarítmico $O(\log_2 n)$.

- Los logaritmos son del mismo orden, independientemente de la base.

Ecuaciones de recurrencia

- Es normal que un algoritmo se base en procedimientos auxiliares, haga llamadas recursivas para tamaños menores o reduzca el tamaño del problema progresivamente.
- En el análisis, el tiempo T(n) se expresa en función del tiempo para T(n-1), T(n-2)...→ **Ecuaciones de recurrencia.**

- **Ejemplo**. ¿Cuántas operaciones **mover** se ejecutan?

```
Hanoi (n, i, j, k)
    if n>0 then
            Hanoi (n-1, i, k, j)
            mover (i, j)
            Hanoi (n-1, k, j, i)
    else
            mover (i, j)
```

Ecuaciones de recurrencia

- En general, las ecuaciones de recurrencia tienen la forma:

$t(n) = b$	Para $0 \leq n \leq n_0$	**Casos base**
$t(n) = f\,(t(n), t(n-1), ..., t(n-k), n)$		En otro caso

Ecuaciones lineales homogéneas

- La ecuación de recurrencia es de la forma:
  $a_0 t(n) + a_1 t(n-1) + ... + a_k t(n-k) = 0$; a_i constante

- **Caso sencillo:** $t(n) = x \cdot t(n-1)$; $t(0) = 1$.
- **Solución:** $t(n) = x \cdot t(n-1) = x \cdot x \cdot t(n-2) = x^3\, t(n-3) = ... = x^n \cdot t(0) \Rightarrow$ **$t(n) = x^n$**

Ecuaciones lineales homogéneas

- Suponiendo que las soluciones son de la forma **$t(n) = x^n$**, la ecuación de recurrencia homogénea:

$$a_0 t(n) + a_1 t(n-1) + \ldots + a_k t(n-k) = 0$$

- Se transforma en:

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_k x^{n-k} = 0 \Rightarrow /x^{n-k} \Rightarrow$$

$$\mathbf{a_0 x^k + a_1 x^{k-1} + \ldots + a_k = 0}$$

Ecuación característica de la ecuación recurrente lineal homogénea

- **k**: conocida. **a_i**: conocidas. **x**: desconocida.
- Resolver el sistema para la incógnita **x**. El resultado es:

$$t(n) = x^n$$

- Pero... Un polinomio de grado **k** tendrá **k** soluciones...

Ecuaciones lineales homogéneas

- Sean las soluciones $x = (s_1, s_2, ..., s_k)$, todas distintas.
- La solución será:

$$\mathbf{t(n) = c_1 \cdot s_1^n + c_2 \cdot s_2^n + ... + c_k \cdot s_k^n = \sum_{i=1}^{k} c_i \cdot s_i^n}$$

- Siendo $\mathbf{c_i}$ constantes, cuyos valores dependen de las condiciones iniciales.

- **Ejemplo**. El tiempo de ejecución de un algoritmo es:

  $t(n) = 3 \cdot t(n-1) + 4 \cdot t(n-2)$ para $n>1$, $t(0)=0$, $t(1)=1$

- Encontrar una fórmula explícita para t(n), y calcular el orden de complejidad del algoritmo.

Ecuaciones lineales homogéneas

- Si no todas las soluciones $x= (s_1, s_2, ..., s_k)$ son distintas, entonces el polinomio característico será:

  $a_0x^n + a_1x^{n-1} + ... + a_kx^{n-k} = (x - s_1)^m \cdot (x - s_2) \cdot ... (x - s_p) \cdot x^{n-k}$

- Las derivadas valen 0 en s_1, hasta la m-1-ésima.

  $a_0n \cdot x^{n-1} + a_1(n-1) \cdot x^{n-2} + ... + a_{k-1}(n-k) \cdot x^{n-k-1} = 0 \Rightarrow \cdot x \Rightarrow$

  $a_0n \cdot x^n + a_1(n-1) \cdot x^{n-1} + ... + a_1(n-k)x^{n-k} = 0$

- **Conclusión: $t(n) = n \cdot s_1^n$** también será solución de la ecuación característica.
- Para la segunda derivada: **$t(n) = n^2 s_1^n$** será solución...

Ecuaciones lineales homogéneas

- Si $\mathbf{s_i}$ tiene multiplicidad **m**, entonces tendremos:

  $s_i^n \quad n \cdot s_i^n \quad n^2 \cdot s_i^n \quad \ldots \quad n^{m-1} \cdot s_i^n$

- Dadas las soluciones $x = (s_1, s_2, \ldots, s_k)$ siendo $\mathbf{s_k}$ de multiplicidad **m**, la solución será:

  $$\mathbf{t(n)} = c_1 \cdot s_1^n + c_2 \cdot s_2^n + \ldots + c_k \cdot s_k^n + c_{k+1} \cdot n \cdot s_k^n + $$
  $$+ c_{k+2} \cdot n^2 \cdot s_k^n + \ldots + c_{k+1+m} \cdot n^{m-1} \cdot s_k^n$$

- **Ejemplo.** Calcular t(n) y el orden de complejidad para:

  $t(n) = 5\, t(n-1) - 8\, t(n-2) + 4\, t(n-3)$

  $t(0) = 0,\ t(1) = 3,\ t(2) = 10$

Recurrencias no homogéneas

- ¿Qué pasa si tenemos algo como $t(n) = 2 \cdot t(n-1) + 1$?
- Términos que no tienen $t(x) \rightarrow$ **Rec. no homogénea**

- **Ejemplo.** Calcular $t(n)$ para: $t(n) = 2t(n-1) + 3^n(n+1)$
  - $t(n) - 2t(n-1) = 3^n(n+1) \Rightarrow$
  - $t(n+1) - 5t(n) + 6t(n-1) = 3^{n+1} \Rightarrow$
  - $t(n+2) - 8t(n+1) + 21t(n) - 18t(n-1) = 0 \Rightarrow$

    Ec. característica: $(x-2)(x-3)^2 = 0$

- **Conclusión:** Si en la ec. de recurrencia aparece un término de la forma $b^n \cdot p(n)$ ($p(n)$ polinomio de n), entonces en la ec. Característica habrá un factor:

  $\mathbf{(x-b)^{Grado(p(n))+1}} \rightarrow$ Sol. **b** con multiplicidad Grado(p(n))+1

Recurrencias no homogéneas

- En general, tendremos recurrencias de la forma:
  $a_0t(n) + a_1t(n-1) + ... + a_kt(n-k) = b_1^n p_1(n) + b_2^n p_2(n) + ...$

- Y la ecuación característica será:
  $(a_0x^k + a_1x^{k-1} + ... + a_k)(x-b_1)^{G(p_1(n))+1}(x-b_2)^{G(p_2(n))+1}... = 0$

- **Ejemplo.** Calcular t(n) y O(t(n)).
  $t(n) = 1 + n$ $\quad n = 0, 1$
  $t(n) = 4t(n-2) + (n+5)3^n + n^2$ $\quad$ Si $n>1$

Cambio de variable

- $t(n) = a \cdot t(n/4) + b \cdot t(n/8) + ...$
- **Cambio de variable:** Convertir las ecuaciones anteriores en algo de la forma $t'(k) = a \cdot t'(k-c_1) - b \cdot t'(k-c_2)$
- Resolver el sistema en **k**.
- Deshacer el cambio, y obtener el resultado en **n**.

- **Ejemplo 1.** Resolver:

  $t(n) = a$ Si $n=1$

  $t(n) = 2\, t(\lfloor n/2 \rfloor) + b \cdot n$ Si $n>1$, con $b>0$

- **Ejemplo 2.** Resolver:

  $t(n) = n$ Si $n<b$

  $t(n) = 3 \cdot t(n-b) + n^2 + 1$ En otro caso

Cambio de variable

- Los órdenes que obtenemos son condicionados a que se cumplan las condiciones del cambio. $t(n) \in O(f|P(n))$
- ¿Cómo quitar la condición?

- **Teorema.** Sea **b** un entero ≥ 2, **f: N** $\rightarrow$ **R**$^+$ una función no decreciente a partir de un n_0 (f es **eventualmente no decreciente**) y $f(bn) \in O(f(n))$ (f es **b-armónica**) y **t: N** $\rightarrow$ **R**$^+$ eventualmente no decreciente. Entonces, si $t(n) \in \Theta(f(n) \mid n=b^k)$ se cumple que $t(n) \in \Theta(f(n))$.

- **Quitar la condición del ejemplo 1.** $t(n) \in \Theta(n \cdot \log n | n=2^k)$
  - $n \cdot \log n$, es eventualmente no decreciente y 2-armónica.
  - t(n) es eventualmente no decreciente.

Otras técnicas

Transformación de la imagen

- Se utiliza en algunos casos, donde las ecuaciones recurrentes son no lineales. **Ejemplo.**

  $t(1) = 6;$ $\quad t(n) = n\, t^2(n/2)$

- Suponiendo **n** potencia de 2, hacemos el cambio $\mathbf{n=2^k}$:

  $t(2^0) = 6;$ $\quad t(2^k) = 2^k\, t^2(2^{k-1})$

- Tomando logaritmos (en base 2):

  $\log t(2^0) = \log 6;$ $\quad \log t(2^k) = k + 2 \cdot \log t(2^{k-1})$

- Se hace una transformación de la imagen:

  $v(x) = \log t(2^x) \Rightarrow$

  $v(0) = \log 6;$ $\quad v(k) = k + 2 \cdot v(k-1)$

Otras técnicas

Transformación de la imagen

- Resolver la ecuación recurrente:

  $v(0) = \log 6; \qquad v(k) = k + 2 \cdot v(k-1)$

- Resultado:

  $v(k) = c_1 \cdot 2^k + c_2 + c_3 \cdot k \Rightarrow v(k) = (3+\log 3) \cdot 2^k - k - 2$

- Ahora deshacer el cambio $v(x) = \log t(2^x)$:

  $\log t(2^k) = \log t(n) = (3+\log 3) \cdot 2^k - k - 2$

- Y quitar los logaritmos, elevando a 2:

  $t(n) = 2^{(3+\log 3)n - \log n - 2} = 2^{3n} \cdot 2^{\log 3 \cdot n} \cdot 2^{-\log n} \cdot 2^{-2} =$

  $= (2^{3n-2} \cdot 3^n)/n$

- Quitar la condición de que n sea potencia de 2.
- ¿Cuánto vale O(t)?

Otras técnicas

Expansión de recurrencias

- Aplicar varias veces la fórmula recurrente hasta encontrar alguna “regularidad”.
- **Ejemplo.** Calcular el número de **mover**, para el problema de las torres de Hanoi.

  $t(0) = 1$

  $t(n) = 2\, t(n-1) + 1.$

- Expansión de la ecuación recurrente:

$$t(n) = 2\, t(n-1) + 1 = 2^2\, t(n-2) + 2 + 1 = 2^3\, t(n-3) + 4 + 2 + 1 =$$

$$= \ldots\ldots^{n} \ldots\ldots = 2^n\, t(n-n) + \sum_{i=0}^{n-1} 2^i = \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

Otras técnicas

Inducción constructiva

- Se usa cuando las ecuaciones son no lineales y no se puede aplicar ninguna de las técnicas anteriores.
- **Inducción**: Dado t(n), suponer que pertenece a algún orden O(f(n)) y demostrarlo por inducción.
  - Para algún valor pequeño, $t(n) \leq c_1 \cdot f(n)$
  - Suponiendo que $t(n-1) \leq c_1 \cdot f(n-1)$, entonces se demuestra que $t(n) \leq c_1 \cdot f(n)$
- **Ejemplo**. Dado lo siguiente,demostrar que $t(n) \in \Theta(n!)$:

  $t(1) = a$

  $t(n) = b \cdot n^2 + n \cdot t(n - 1)$
  - Demostrar por inducción que $t(n) \in \Omega(n!)$.
  - Demostrar por inducción que $t(n) \in O(n!)$.

Condiciones iniciales

- ¿Cuál es el significado de las condiciones iniciales?
- **Condición inicial:** caso base de una ecuación recurrente.

- Dada una ecuación de recurrencia t(n), su valor depende de las condiciones iniciales.

  $t(n) = 5 \cdot t(n-1) - 8 \cdot t(n-2) + 4 \cdot t(n-3)$ Si $n > 10$

  $t(n) = n$ Si $n \leq 10$

- Resultado: $t(n) = c_1 + c_2 2^n + c_3 n \cdot 2^n$

- $\mathbf{c_1}$, $\mathbf{c_2}$, $\mathbf{c_3}$ dependen de los casos base.
- Aplicar las condiciones iniciales para despejar c_1, c_2, c_3.
- ¿Cuántas aplicar? ¿Cuáles?

Condiciones iniciales

- ¿Cuántas?
  - Tenemos 3 incógnitas: $\mathbf{c_1, c_2, c_3}$.
  - Aplicar 3 condiciones iniciales.

- ¿Cuáles?
  - La fórmula $t(n) = c_1 + c_2 2^n + c_3 n \cdot 2^n$ debe dar el mismo resultado que expandir la recurrencia, para valores de n >10.
  - Las condiciones iniciales deben ser casos alcanzables para los valores de n > 10.
  - **Ejemplo.** El caso base t(0)=0 no se alcanza para n>10 $\rightarrow$ no influye en t(n), para valores grandes.
  - Posibles condiciones a aplicar:

    $t(10) = 10 = c_1 + c_2\, 2^{10} + c_3 \cdot 10 \cdot 2^{10}$

    $t(9) = 9 = c_1 + c_2\, 2^{9} + c_3 \cdot 9 \cdot 2^{9}$

    $t(8) = 8 = c_1 + c_2\, 2^{8} + c_3 \cdot 8 \cdot 2^{8}$

Condiciones iniciales

- También sería válido aplicar t(12), t(11), t(10) ó t(30), t(20), t(10). El resultado sería el mismo, pero...
- Para obtener el valor de t(11), t(12),... debemos aplicar la recurrencia:

  $t(11) = 5\ t(10) - 8\ t(9) + 4\ t(8) = 5 \cdot 10 - 8 \cdot 9 + 4 \cdot 8 = 10$

  $t(12) = 5\ t(11) - 8\ t(10) + 4\ t(9) = ...$

- Si hemos calculado t(n) para $n=2^k$, entonces la fórmula será correcta cuando $n=2^k$.
- Las condiciones iniciales deben ser también potencias de 2.

- **Hipótesis:** Las condiciones iniciales sólo influyen en las constantes de t(n), luego no aparecen en O(t(n)).
- ¿Es siempre cierta? **Ejemplo.** $t(n) = t(n-1)^2$; $t(0) = a$

Ejemplos

- **Ejemplo 1.** Dada la siguiente ecuación de recurrencia, con a, b, c y d $\in R^+$ y e, $n_0 \in N^+$:

$$f(n) = \begin{cases} d & \text{Si } n \leq n_0 \\ a \cdot f(n-e) + bn + c & \text{Si } n > n_0 \end{cases}$$

Demostrar que: $a < 1 \Rightarrow f \in O(n)$

$a = 1 \Rightarrow f \in O(n^2)$

$a > 1 \Rightarrow f \in O(a^{n/c})$

- **Ejemplo 2.** Calcular el tiempo de ejecución (en segundos y en número de instrucciones) de la siguiente función. Escribir también el valor final de la función.

```
procedure Recursiva (n: integer): integer;
        if n≤0  then Return 1
        else Return 2*Recursiva(n-1)
```

Ejemplos

- **Ejemplo 3.** Dada la siguiente ecuación de recurrencia, con a, b, c y $p \in R^+$ y d, $n_0 \in N^+$:

$$f(n) = \begin{cases} c & \text{Si } n \leq n_0 \\ a \cdot f(n/d) + bn^p & \text{Si } n > n_0 \end{cases}$$

Demostrar que: $a < d^p \Rightarrow f \in O(n^p)$

$a = d^p \Rightarrow f \in O(n^p \cdot \log n)$

$a > d^p \Rightarrow f \in O(n^{\log_d a})$

- **Ejemplo 4.** Calcular el número de instrucciones de asignación del siguiente algoritmo.

```
procedure Otro (n: integer): integer;
        for i:= 1 to n do
                M[i]:= M[i] + 1;
        if i>0 then Otro(n-4)
```

Ejemplos

- **Ejemplo 5.** El tiempo de ejecución de un determinado programa se puede expresar con la siguiente ecuación de recurrencia:

$$t(n) = \begin{cases} 2n & \text{Si } n \leq 10 \\ 2t(\lfloor n/2 \rfloor) + 3t(\lfloor n/4 \rfloor) + 2n + 1 & \text{En otro caso} \end{cases}$$

- Calcula el tiempo de ejecución para los valores de **n** que sean potencia de 2. Exprésalo usando las notaciones O, Ω ó Θ.
- Muestra las condiciones iniciales que se deberían aplicar.
- Eliminar la condición de que **n** sea potencia de 2.
- La afirmación **t(n)** $\in \Omega(\log n)$ ¿es correcta en este caso?, ¿es una buena cota para el orden de complejidad del programa?