Programa de teoría

Parte I. Estructuras de Datos.

1. Abstracciones y especificaciones.
2. Conjuntos y diccionarios.
3. Representación de conjuntos mediante árboles.
4. Grafos.

Parte II. Algorítmica.

1. **Análisis de algoritmos.**
2. Divide y vencerás.
3. Algoritmos voraces.
4. Programación dinámica.
5. Backtracking.
6. ~~Ramificación y poda.~~

PARTE II: ALGORÍTMICA

Tema 1. Análisis de algoritmos.

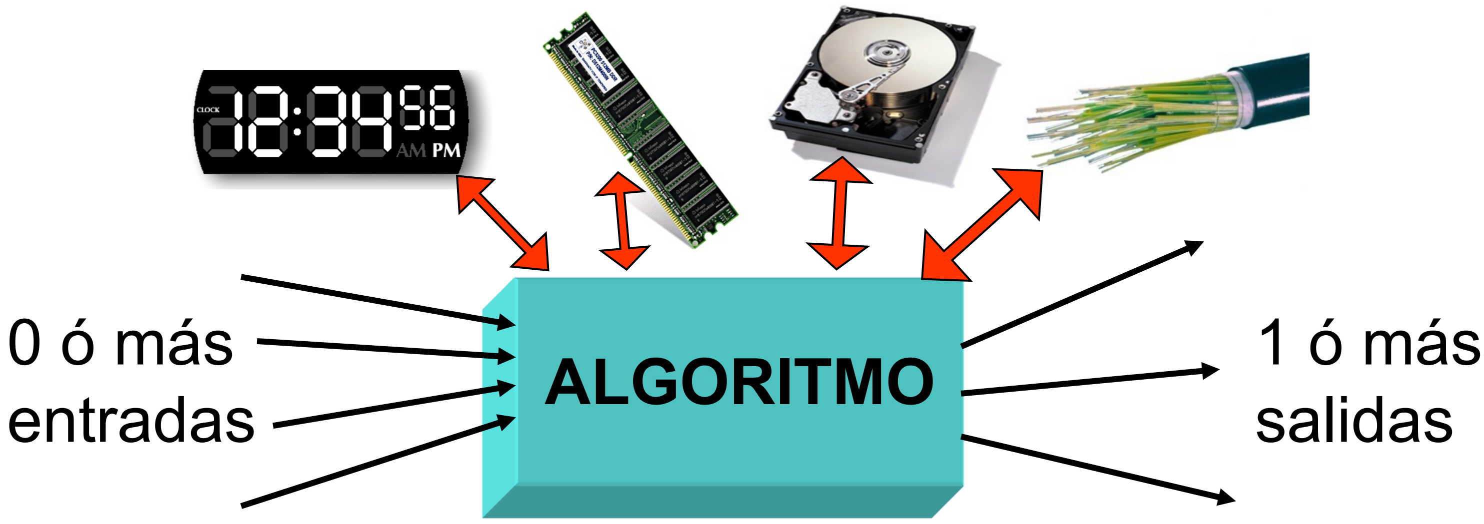
1.1. Introducción.

1.2. Notaciones asintóticas.

1.3. Ecuaciones de recurrencia.

1.1. Introducción

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



- Algoritmo **mejor** cuantos **menos** recursos consuma para conseguir **cierto** **resultado**…

1.1. Introducción

Criterio empresarial: Maximizar la eficiencia:

>> Eficiencia = Relación entre:

- **Recursos consumidos:**
  - **Tiempo de ejecución (t).**
  - **Memoria principal (m).**
  - Entradas/salidas a disco.
  - Comunicaciones, procesadores,...
- **Productos conseguidos:**
  - Resolver un problema de forma exacta vs. aproximada.
  - Resolver para todos los casos vs. algunos casos.

Otros criterios: fácil de programar, corto, legible, robusto...

1.1. Introducción

- **Recursos consumidos.**

  Ejemplo. ¿Qué recursos de **t**iempo y **m**emoria consume algoritmo ***BC*** ? (*búsqueda con centinela*)

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

- **Respuesta:**

>> Depende de:

- lo que valga ***n***
- de lo haya en **A** y en **x**
- de los tipos de datos, de la máquina...

1.1. Introducción

Clasificación de **factores** para consumo de recursos:

- **Factores externos** (al algoritmo).
  - El ordenador donde se ejecute.
  - El lenguaje de programación y el compilador usado.
  - La implementación que haga el programador. Estructuras de datos.
  - **No dan información sobre el algoritmo**.
- **Datos de entrada:**
  - **Tamaño:** Ej.: Procesar un blog: ***n*** = número de mensajes.
  - **Contenido:** Más/menos recursos según *contenido* datos.

    No posible estudiar todos… estudiar casos especiales:

    - **Mejor caso.** Contenido que produce ejecución + rápida.
    - **Peor caso.** Contenido que produce ejecución + lenta.
    - **Caso promedio.** Media de t para todos los posibles contenidos.

1.1. Introducción

>> Estudiaremos los **recursos** que necesita el algoritmo en **función** de los **datos de entrada**, caracterizados por su:

- tamaño (***n***)
- contenido (posibles casos: mejor, peor, promedio)

>> Algo así como "t(*n*, caso)", para casos especiales…

>> **Ejemplo**. Algoritmo ***BC*** (*búsqueda con centinela*)

– **Mejor** caso, x está en 1ª posición:

"t(***n***, caso **mejor**)" = $t_m(\boldsymbol{n}) = 1$ (comprobar 1ª posición)

– **Peor** caso, x no está:

"t(***n***, caso **peor**)" = $t_M(\boldsymbol{n}) = \boldsymbol{n} + 1$ (comprobar *n*+1 posiciones)

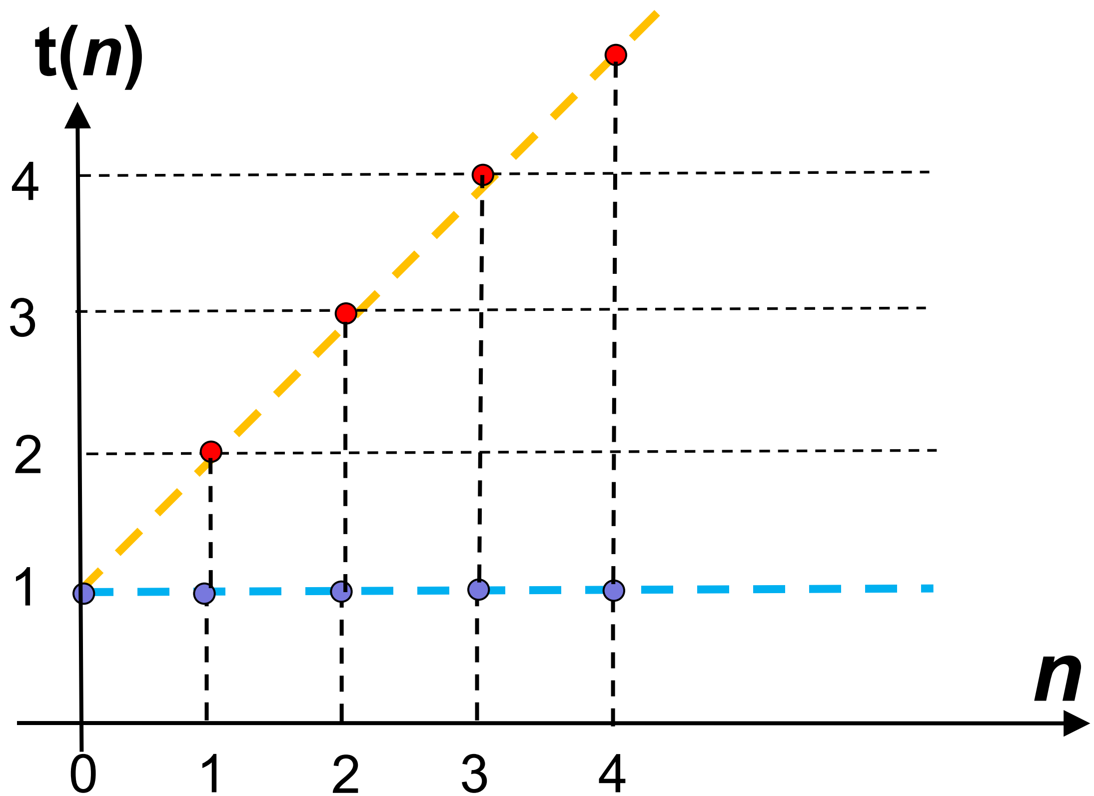
Ojo: ¡¡El mejor caso no significa *n* (tamaño) pequeño!!

1.1. Introducción

BC *(búsqueda centinela)*:

– **Mejor** caso, $t_m(n) = 1$

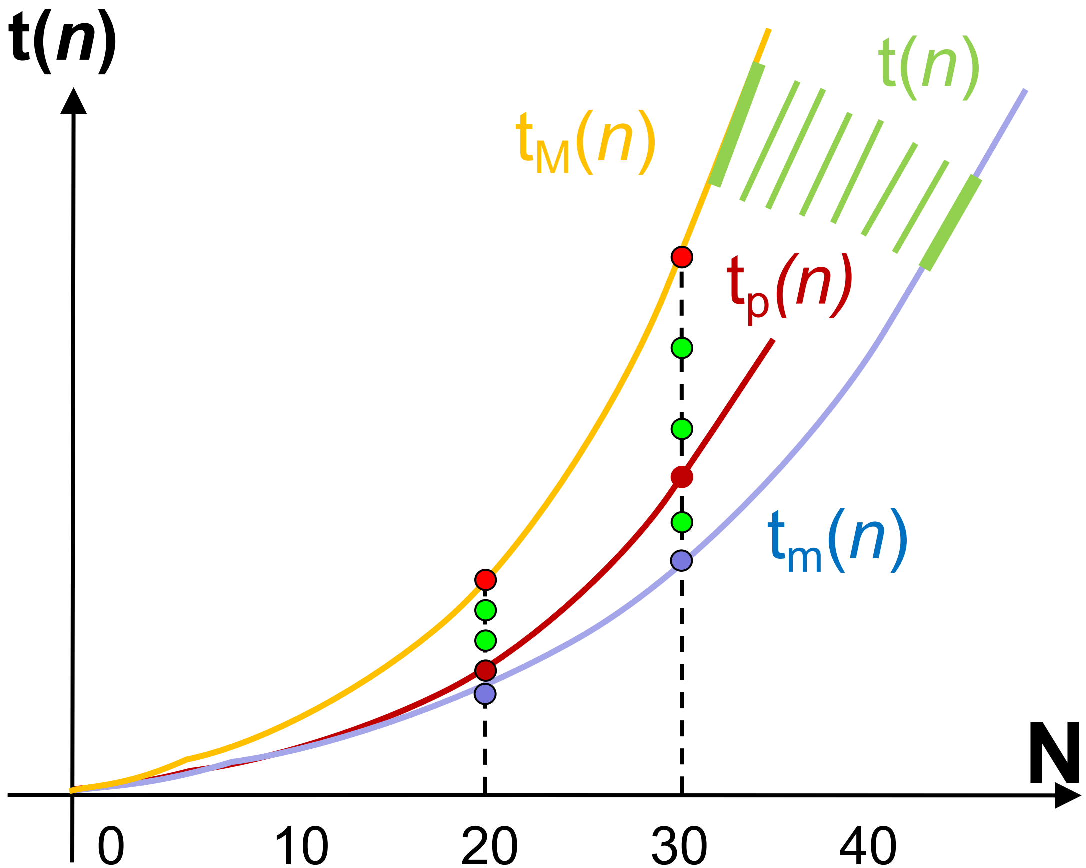
– **Peor** caso, $t_M(n) = n + 1$



Ojo: ¡¡El mejor caso no significa *n* (tamaño) **pequeño!!**

1.1. Introducción

>> En general, tendremos algo así…



>> Siempre se cumple….

$t_m(n) <= t(n) <= t_M(n)$

$t_m(n) <= t_p(n) <= t_M(n)$

1.1. Introducción

¿Qué significa notación t(*n*)?

Algunas posibilidades,

de más a menos detalle:

BC (peor caso)

```
i:= 0                 a
A[n+1]:= x            b
repetir
  i:= i + 1           c
hasta A[i] == x       d
```

- Tiempo de ejecución (segs): $\mathbf{t}(n) = (n+1)\,(c+d) + (a+b)$
  - Con a, b, c y d constantes = segundos de operaciones básicas
- Instrucciones ejecutadas: $\mathbf{t}(n) = (n+1)\,(1+1) + (1+1)$
  - Simplificación: tardan todas igual, a = b = c = d = 1
- Ejecuciones del bucle principal: $\mathbf{t}(n) = n+1$
  - ¿Cuánto tiempo, cuántas instrucciones,...?
  - Sabemos que cada ejecución requiere un tiempo constante.

1.1. Introducción >> Conteo

Proceso básico de análisis de eficiencia, **conteo de:**

- **instrucciones:** seguir ejecución del algoritmo, sumando las instrucciones que se ejecutan.
- **de memoria:** ídem, sumando memoria usada (variables globales, locales, dinámicas).

BC (peor caso)	
i:= 0	a
A[*n*+1]:= x	b
repetir	
i:= i + 1	c
hasta A[i] == x	d

>> Normalmente, interesa máxima memoria usada

Alternativa a conteo: Si no se puede predecir flujo de ejecución, intentar predecir el **trabajo total realizado**.

- Ejemplo ***BC***: se accede a n+1 posiciones de un array.
- Ejemplo recorrido sobre grafos: se recorren todas las adyacencias, aplicando un tiempo cte. en cada una.

1.1. Introducción >> Conteo

Reglas básicas para conteo:

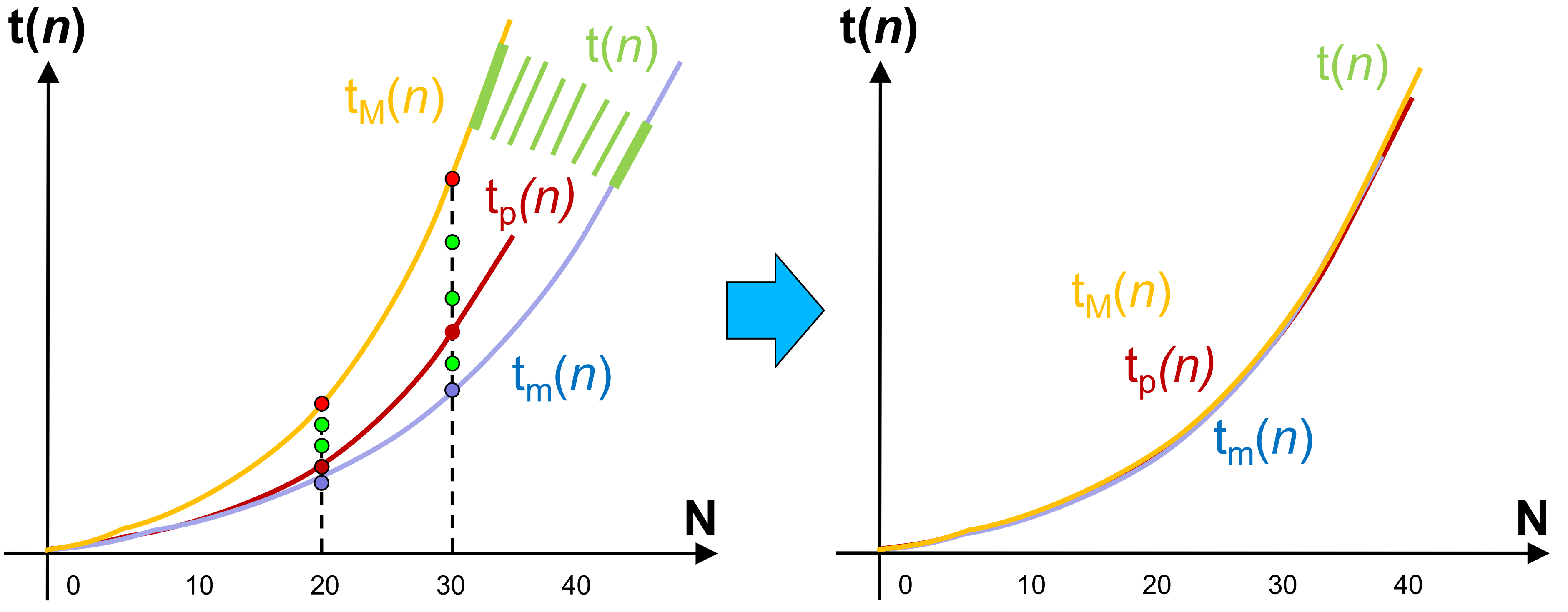
- De instrucciones → sumar 1 por cada instrucción o línea de código de ejecución constante.
- De tiempo de ejecución → sumar una constante (a, b...) por tipo de instrucción o grupo de instrucciones en secuencia.
- **Bucles FOR**: Se pueden expresar como un sumatorio, con los límites del FOR como límites del sumatorio.

$$\sum_{i=1}^{n} k = k\,n \qquad \sum_{i=a}^{b} k = k(b-a+1) \qquad \sum_{i=1}^{n} i = n(n+1)/2$$

$$\sum_{i=a}^{b} r^i = \frac{r^{b+1} - r^a}{r - 1} \qquad \sum_{i=1}^{n} i^2 \approx \int_0^n i^2\,di = (i^3)/3\,\Big]_0^n = (n^3)/3$$

>> FOR=nº fijo iteraciones >> $t_m(n) = t_p(n) = t_M(n)$; $t(n)$ único

1.1. Introducción >> Conteo



>> FOR=nº fijo iteraciones >> $t_M(n)$ = $t_p(n)$ = $t_m(n)$; $t(n)$ único

1.1. Introducción >> Conteo

Reglas básicas para conteo (2):

- **Llamadas a procedimientos**: Calcular primero los procedimientos que no llaman a otros: $t_1(n)$, $t_2(n)$, ...
- **IF** y **CASE**: Estudiar lo que puede ocurrir. ¿Se puede predecir cuándo se cumplirán las condiciones?
  - Mejor caso $t_m(n)$ y peor caso $t_M(n)$ según la condición.
  - Caso promedio $t_p(n)$: suma del tiempo de cada caso, por probabilidad de ocurrencia de ese caso.
- **Bucles WHILE y REPEAT**: Estudiar lo que puede ocurrir... ¿Se puede convertir en un FOR? ($t_m(n)=t_M(n)$)

  >> Si no, estudiar si se puede acotar nº iteracaiones.

1.1. Introducción >> Conteo

Ejercicio: Estudia t(*n*) en estos ejemplos donde $t_m(n)=t_M(n)$

```
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
```

1.1. Introducción >> Conteo

Ejercicio: Estudia t(n) en este ejemplo donde $t_m(n) < t_M(n)$

```
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
   key:= key+1
end
```

1.1. Introducción >> Conteo

Tiempo promedio: si $t_m(n) < t_M(n)$ puede ser útil calcular $t_p(n)$

Concepto: $t_p(n)$ = media de t(n) para todos los casos posibles (contenidos posibles de los datos de entrada).

En la **práctica**:

- Para IF/CASE, agrupar los casos en conjuntos de casos que tarden lo mismo y hacer suma ponderada por probabilidad:

$$\mathbf{t_{p\_if}(n) = t_{p\_true} * p_{true} + t_{p\_false} * p_{false}}$$

- Para los WHILE/REPEAT, IF-ELSE encadenado, constante si cada iteración es constante (serie geométrica con razón<1).
- A veces útil parametrizar la probabilidad con índice bucle. (segundo ejemplo de página siguiente)

1.1. Introducción >> Conteo

Ejercicio (tiempo promedio):

Estudiar $t_p(n)$ para las instrucciones de asignación…

```
cont:=0
para i:= 1,...,n hacer
    para j:= 1,...,i-1 hacer
        si a[i] < a[j] entonces
            cont:= cont + 1
        finsi
    finpara
finpara
```

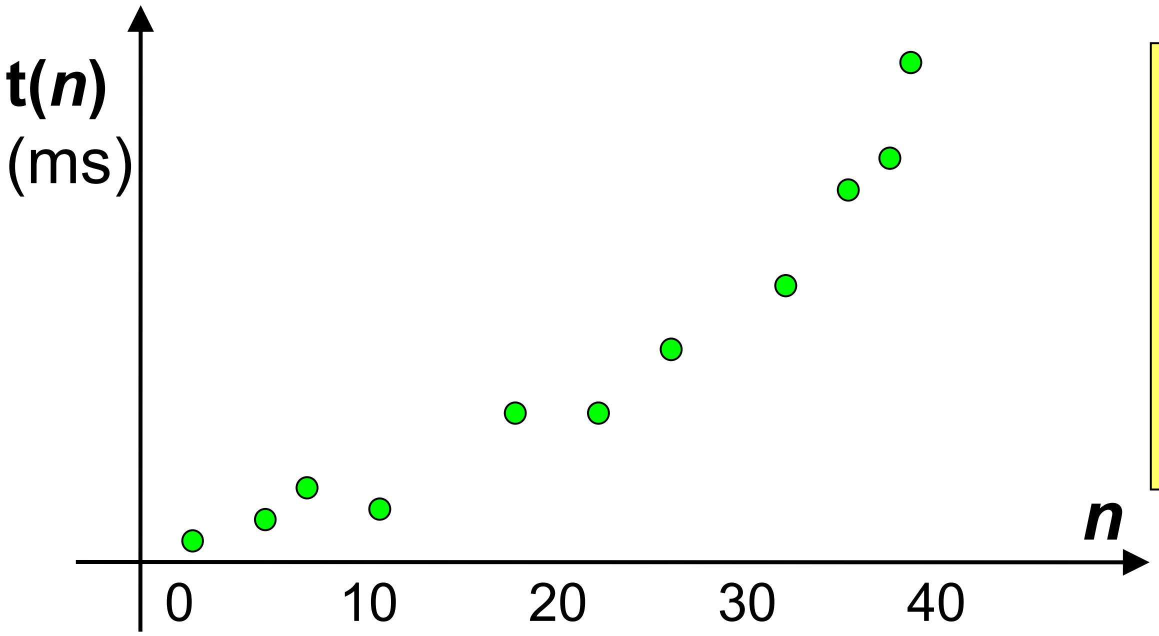
```
i:= 1
mientras i ≤ n hacer
    si a[i] ≥ a[n] entonces
        a[n]:=a[i]
    finsi
    i:= i + 1
finmientras
```

1.1. Introducción >> Análisis experimental

- Análisis algoritmos también puede ser **a posteriori**:

  >> Implementar algoritmo y (después, a posteriori…) hacer un

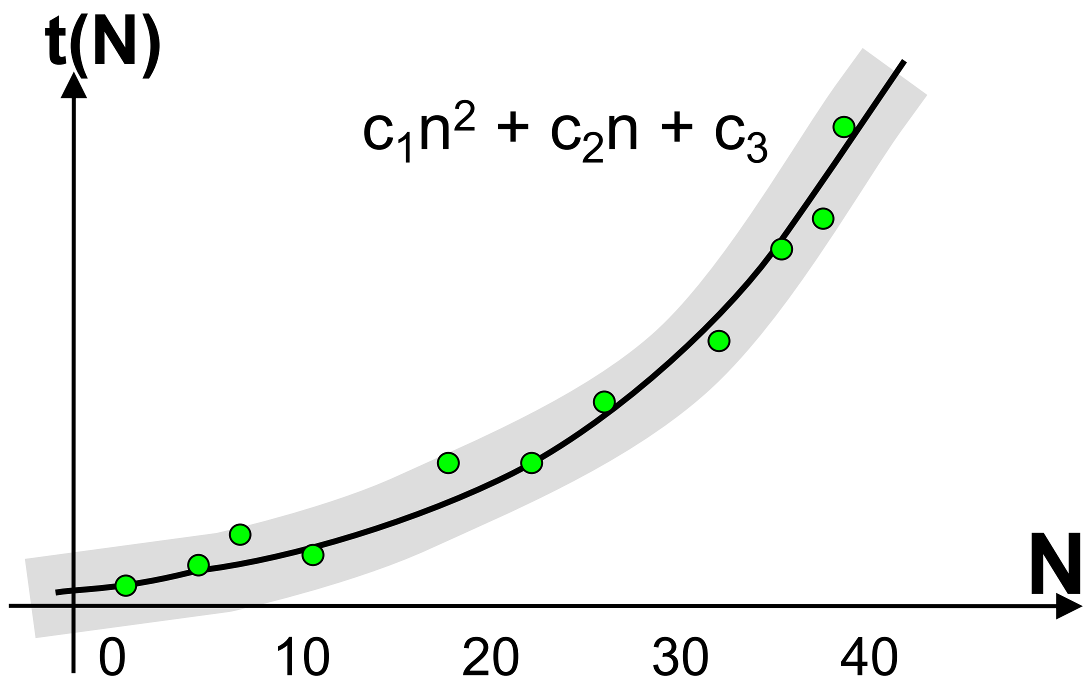
  >> **Estudio experimental** = medir tiempo para diferentes entradas, variando su tamaño (*n*) y contenido (casos).
- Cobran especial importancia **herramientas estadísticas**: gráficas, técnicas de muestreo, regresiones, tests de hipótesis, etc.
- Ser **específico**, indicar **factores externos**: ordenador, S.O., condiciones de ejecución, opciones de compilación, etc.



i5 a 3 Ghz
8 Gb de RAM, DDR4
SSD 512 Gb.
S.O. Linux **RHEL** 8.3
(único proceso en ejec.)
Compilado con -o3

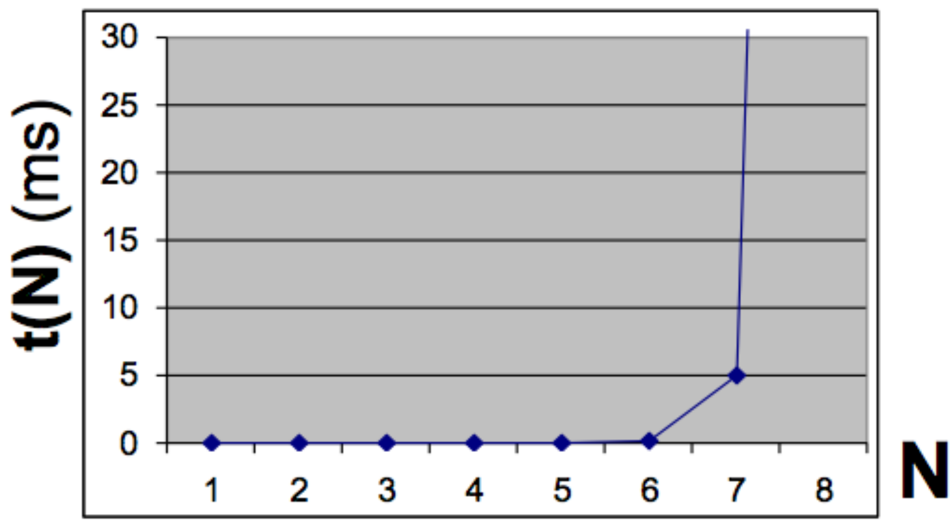
1.1. Introducción >> Análisis experimental

- Indicamos los **factores externos**, porque influyen en los tiempos (multiplicativamente), y son útiles para comparar tiempos tomados bajo condiciones distintas.
- El análisis a posteriori suele complementarse con un **estudio teórico** (previo) y un **contraste teórico/experimental**.
- **Ejemplo**. Haciendo el estudio teórico del anterior programa, deducimos que su tiempo es de la forma: $c_1n^2 + c_2\, n + c_3$
- Podemos hacer una regresión. → ¿Se ajusta bien? ¿Es correcto el estudio teórico?



1.1. Introducción >> Análisis experimental

- Contraste teórico/experimental **permite**: detectar posibles errores de implementación, hacer previsiones para tamaños inalcanzables, comparar implementaciones...
- Sin el estudio teórico, extraer conclusiones relevantes del tiempo de ejecución puede ser complejo.
- **Ejemplo.** Programa "cifras.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?
- El **análisis a priori** es siempre un estudio teórico previo a la implementación. Puede servir para evitar la implementación, si el algoritmo es poco eficiente.

1.1. Introducción >> Análisis experimental

Resumiendo:

- **Análisis a priori,** estudio teórico previo a la implementación.
- **Análisis a posteriori,** estudio experimental del código.
- **Contraste** teórico/experimental, compara ambos.

¿Se puede hacer experimental a priori?

\>> No, no hay código que medir.

¿Se puede hacer un teórico a posteriori?

\>> Sí, pero mejor a priori… Ya que lo hacemos, mejor antes de implementar, porque quizás el teórico nos dice que ese diseño no es bueno, nos evita perder tiempo en implementar.

1.2. Notaciones asintóticas.

- El tiempo de ejecución **t(n)** está dado en base a constantes (a, b..,) que dependen de factores externos.
- Nos interesa un análisis que sea:
  - Independiente de esos factores externos.
  - Resuma lo importante para n “grande”.
- **Notaciones asintóticas:** Indican como crece **t**:
  - sin considerar constantes (factores externos) y para
  - valores suficientemente grandes (asintóticamente)

  a) $O(t)$: Orden de complejidad de t

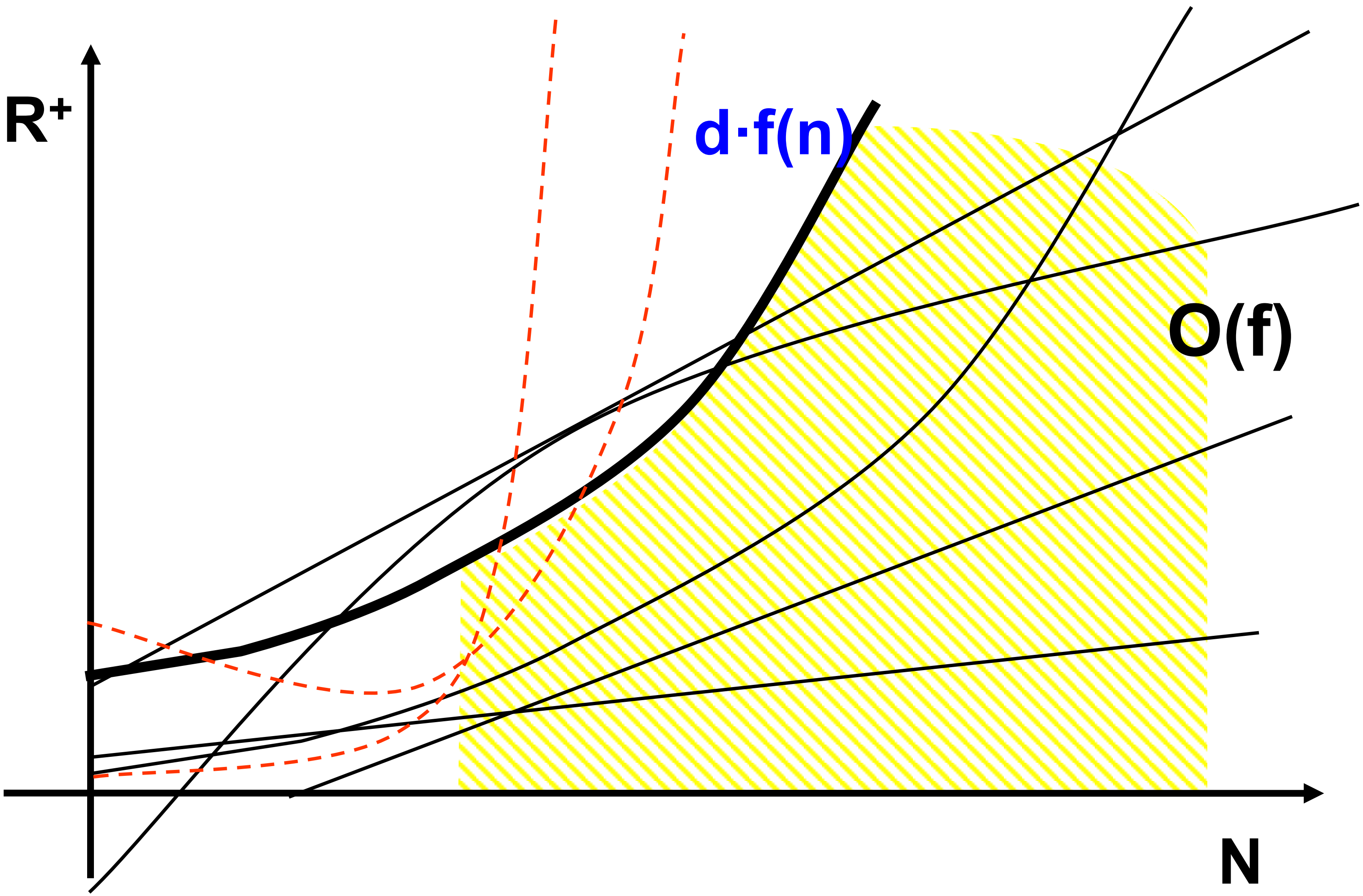
  b) $\Omega(t)$: Orden inferior de t, u omega de t

  c) $\Theta(t)$: Orden exacto de t

  d) $o(t)$: o pequeña de t

1.2.1. Definiciones.

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



1.2.1. Definiciones.

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

- Dada una función **f: N → 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.

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

1.2.1. Definiciones.

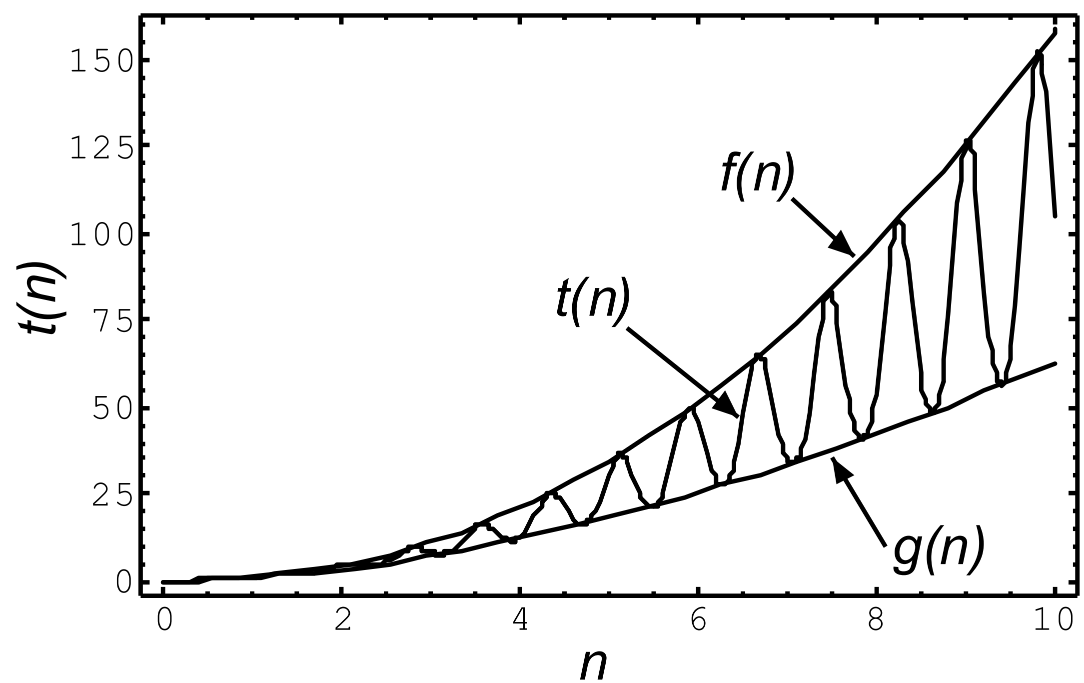
Observaciones:

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

1.2.1. Definiciones.

Uso de los órdenes de complejidad

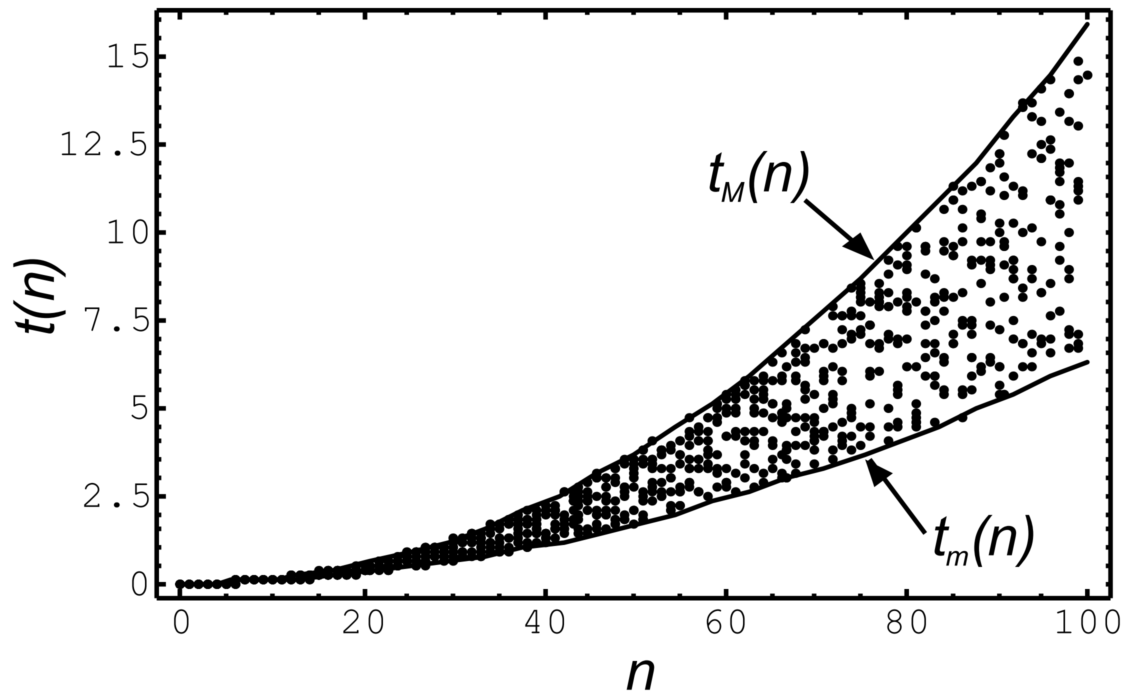
- 1) Dado un tiempo **t(n)**, encontrar la función **f** más simple tal que **t** $\in$ **O(f)**, y que más se aproxime asintóticamente.
- **Ejemplo**. $t(n) = 2n^2/5 + 6n + 3\pi \cdot \log_2 n + 2 \Rightarrow t(n) \in O(n^2)$
- 2) Acotar una función difícil de calcular con precisión.
- **Ejemplo.** $t(n) \in O(f(n))$



1.2.1. Definiciones.

Uso de los órdenes de complejidad

- 3) Acotar una función que no tarda lo mismo para el mismo tamaño de entrada (distintos casos, mejor y peor).
- **Ejemplo.**
  $t(n) \in O(t_M(n))$



- Igual que con la cota superior, podríamos hacer con la cota inferior...

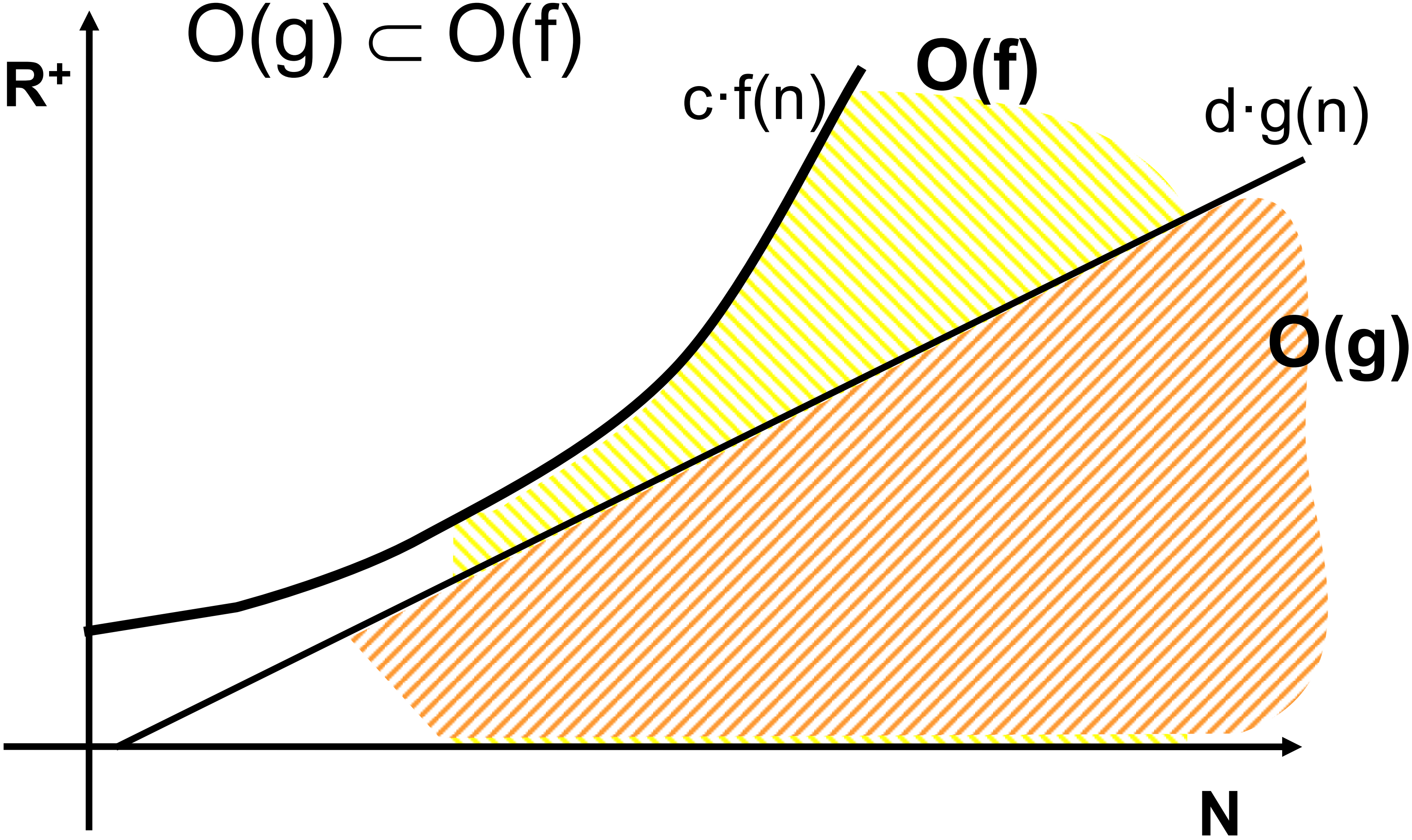
1.2.1. Definiciones.

- **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**.

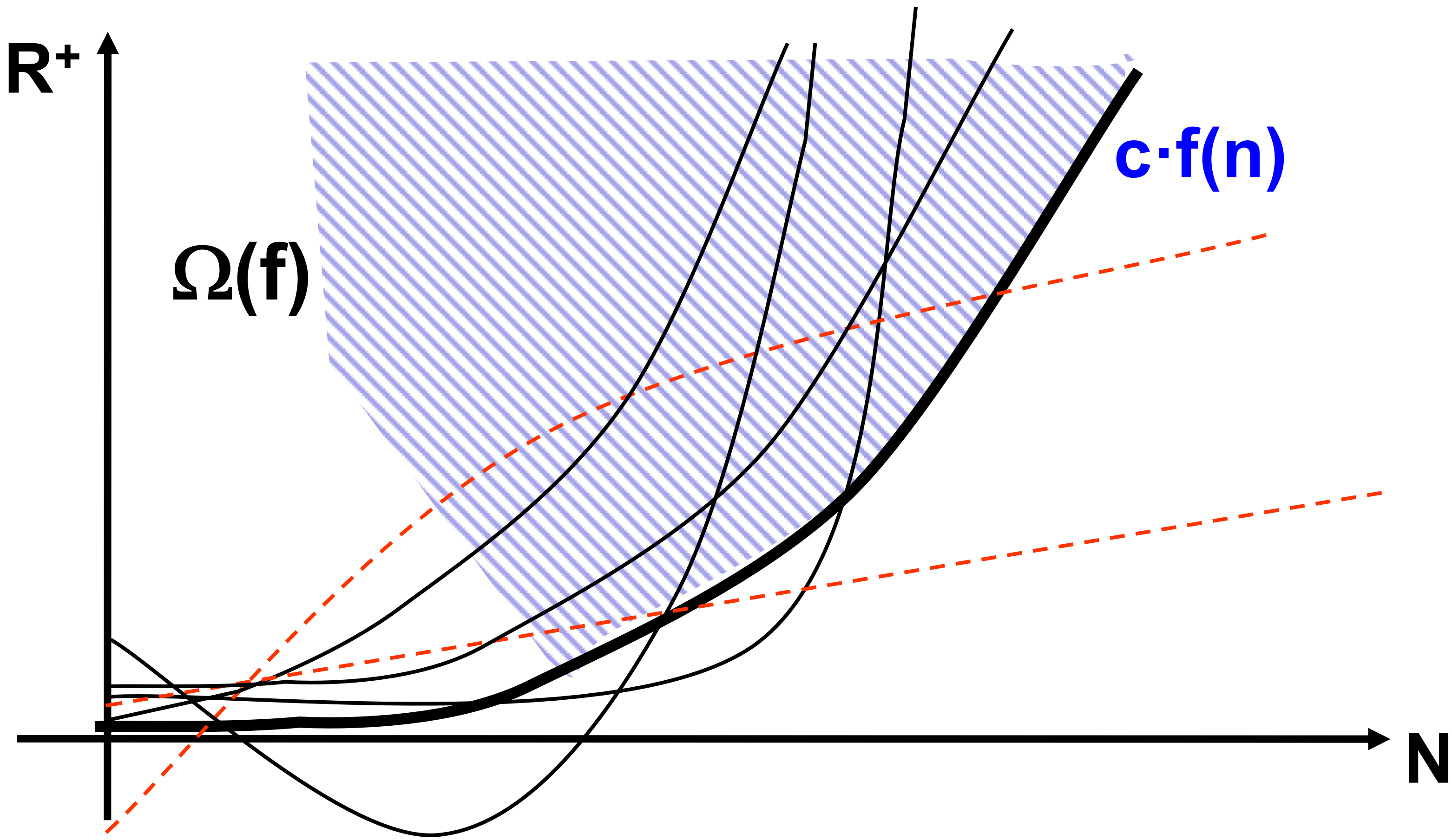
1.2.1. Definiciones.

$O(g) \subset O(f)$



1.2.1. Definiciones.

b) **Orden inferior u omega de f(n): $\Omega(f)$**



1.2.1. Definiciones.

b) **Orden inferior u omega de f(n): $\Omega(f)$**

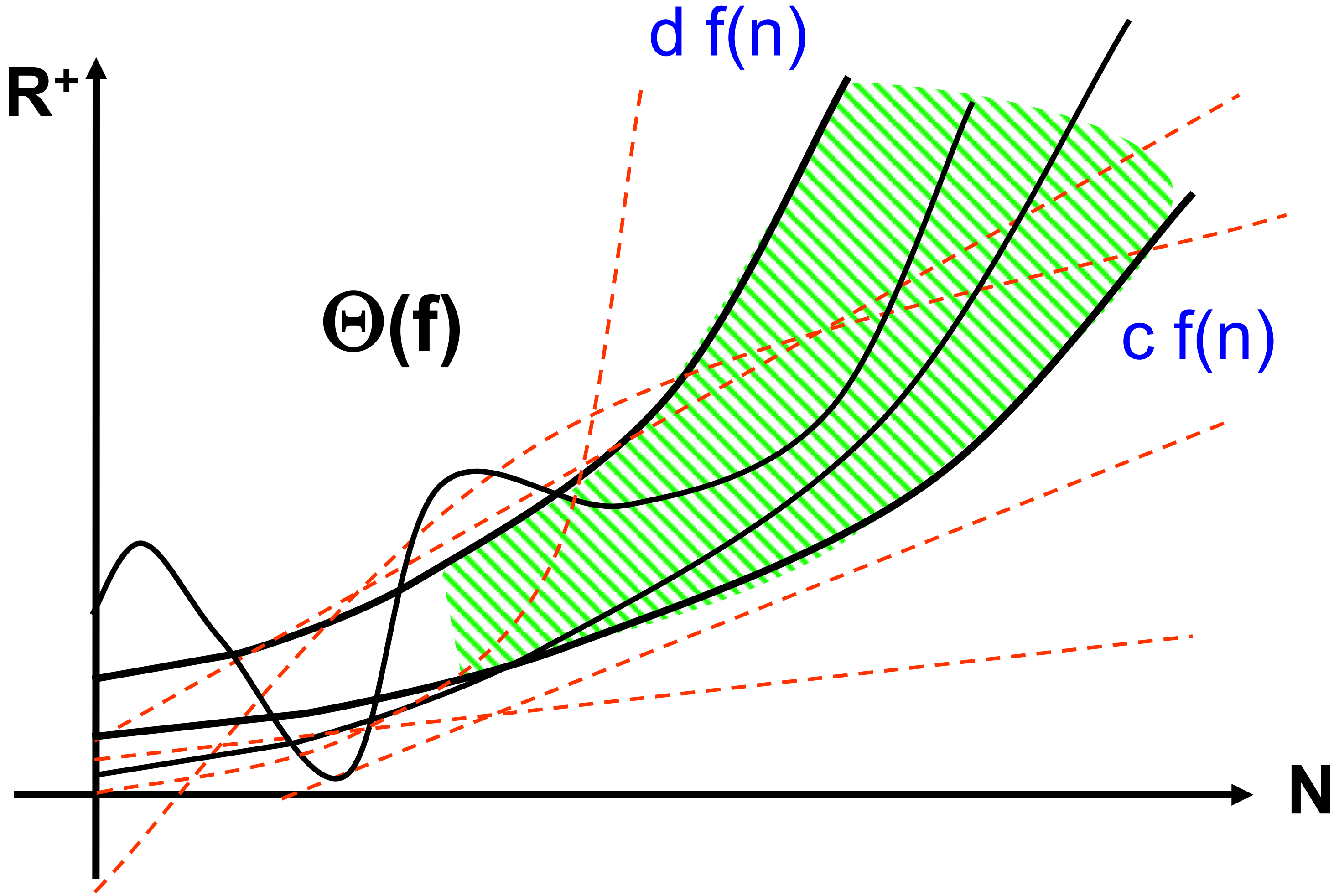
- Dada una función **f: $\mathbf{N} \rightarrow \mathbf{R}^+$**, llamamos **omega de f** al conjunto de todas las funciones de **N** en **$\mathbf{R}^+$** acotadas **inferiormente** por un múltiplo real positivo de **f**, para valores de **n** suficientemente grandes.

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

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

1.2.1. Definiciones.

c) Orden exacto de f(n): Θ(f)



1.2.1. Definiciones.

c) **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.

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

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

- Si un algoritmo tiene un **t** tal que $t \in O(f)$ y $t \in \Omega(f)$, entonces **t** $\in$ **Θ(f)**.

1.2.1. Definiciones.

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

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

1.2.1. Definiciones.

d) **Notación o pequeña de f(n): o(f)**

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

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

- Esta notación conserva las constantes multiplicativas para el término de mayor orden.

1.2.1. Definiciones.

Notación o pequeña de f(n): o(f)

- **Ejemplo**. $t(n) = a_m n^m + a_{m-1} n^{m-1} + ... + a_1 n + a_0$

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

- $t(n) = 3{,}2n^2 + 8n - 9 \in o(¿?)$
- $t(n) = 82\ n^4 + 3 \cdot 2^n + 91 \log_2 n \in o(¿?)$
- $t(n) = 4n^3 + 3n^3 \log_2 n - 7n^2 + 8 \in o(¿?)$

1.2.2. Propiedades de las notaciones asintóticas.

- **P1. Transitividad.**
  Si f $\in$ O(g) y g $\in$ O(h) entonces f $\in$ O(h).
  - Si f $\in$ Ω(g) y g $\in$ Ω(h) entonces f $\in$ Ω(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?

1.2.2. Propiedades de las notaciones asintóticas.

- **P3. Relación pertenencia/contenido.** 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) $O(f) \subseteq O(g) \Leftrightarrow f \in O(g)$

- **P4. Propiedad del máximo.** Dadas f y g, de N en R^+, $O(f+g) = O(\max(f, g))$.
  - Con omegas: $\Omega(f+g) = \Omega(\max(f, g))$
  - ¿Y para los $\Theta(f+g)$?
  - Ejemplo: $O(2^n + n^6 + n!) = ...$

1.2.2. Propiedades de las notaciones asintóticas.

- **P5. Equivalencia entre notaciones.**
  Dadas f y g de N en R^+, $O(f)=O(g) \Leftrightarrow \Theta(f)=\Theta(g) \Leftrightarrow f \in \Theta(g) \Leftrightarrow \Omega(f)=\Omega(g)$

- **P6. Relación límites/órdenes.**
  Dadas f y g de N en R^+, se cumple:
  - i) $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in R^+ \Rightarrow O(f) = O(g)$
  - ii) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow O(f) \subset O(g)$
  - iii) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty \Rightarrow O(f) \supset O(g)$

1.2.3. Notaciones con varios parámetros.

- En general, tiempo/memoria consumidos pueden depender de muchos parámetros: **f: $\mathbf{N^m \rightarrow R^+}$** (f: Nx...^{m}..xN $\rightarrow$ R^+)

  >> Ejemplo: $t(n,m) = m\, n^2$

- Podemos extender los conceptos de O(f), Ω(f) y Θ(f)…

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

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

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

1.2.3. Notaciones con varios parámetros.

- Las propiedades se siguen cumpliendo
- Para calcular órdenes, la clave es:
  - ver qué términos acotan a otros,
  - considerando que **$O(n) \neq O(m)$**.

  Ejemplo,

$$t(n,m,r)=nmr+mr^2+nr+n^3+n^2r \in O(?)$$

- ¿Qué relación hay entre los siguientes órdenes?

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

1.2.4. 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 $\mathbf{f: N \rightarrow R^+}$, y $\mathbf{P: N \rightarrow B}$, llamamos **orden de f según P** (o condicionado a P) al conjunto:

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

1.2.4. Notaciones condicionales.

- De igual forma, tenemos $\Omega(f \mid P)$ y $\Theta(f \mid P)$.
- **Ejemplo.**
  - Tiempo para tamaños de entrada potencia de 2:
    $t(n) \in O(f \mid n = 2^k)$
  - Para tamaños múltiplos de 2:
    $t(n) \in O(f \mid n = 2k)$
- $O(f) = O(f \mid \text{true})$.
- Para cualquier f y g, $f \in O(g \mid \text{false})$.
- ¿$O(f) \leftrightarrow O(f \mid P)$?

1.2.5. Cotas de complejidad frecuentes.

- **Algunas relaciones entre órdenes frecuentes.**

  $O(1) \subset O(\log n) \subset O(n) \subset O(n \cdot \log n) \subset$

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

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

- ¿Dónde va $O(3^n)$? ¿Y $O(n^3\,2^n)$?
- ¿Qué pasa con las omegas? ¿Y con los órdenes exactos?

1.2.5. Cotas de complejidad frecuentes.

- El orden de un polinomio $a_n x^n + ... + a_1 x + a_0$ es $O(x^n)$.
- $\sum_{i=1}^{n} 1 \in O(n)$; $\sum_{i=1}^{n} i \in O(n^2)$; $\sum_{i=1}^{n} i^m \in O(n^{m+1})$
- 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. Por eso, se omite normalmente.
- **Sumatorios:** se pueden aproximar con integrales, una acotando superior y otra inferiormente.
- Casos **promedios**: usar probabilidades.


Fondos7.net

1.3. 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)...$\rightarrow$ **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)
```

1.3. 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

- **Tipos de ecuaciones de recurrencia:**
  - Lineales y homogéneas:

    $a_0t(n) + a_1t(n-1) + ... + a_kt(n-k) = 0$
  - Lineales y no homegéneas:

    $a_0t(n) + a_1t(n-1) + ... + a_kt(n-k) = p(n) + ...$
  - No lineales:

    Ejemplo: $a_0t^2(n) + t(n-1)*t(n-k) + sqrt(t(n-2) + 1) = p(n)$

1.3. Ecuaciones de recurrencia.

- Ejemplos… ¿qué ecuación sale? ¿de qué tipo? ¿casos?

```
algoritmoRec1(int a[], int n)
    si n<=1
        return 1
    si no
        return 1+algoritmoRec1(a,n-1)

algoritmoRec2(int a[], int pos)
    si pos>1
        si par(a[pos])
            algoritmoRec2(a,pos/2)
        en otro caso
            algoritmoRec2(a,pos/4)
```

1.3.1. Ecuaciones lineales homogéneas.

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

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

- Se transforma en:

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

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

(Ecuación característica)

1.3.1. Ecuaciones lineales homogéneas.

- Suponiendo soluciones con 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:

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

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

(Ecuación característica)

- **k**: conocida. $\mathbf{a_i}$: conocidas. **x**: desconocida.
- Resolver la ecuación para la incógnita **x**… $x=s$
- El resultado es: $t(n) = s^n$
- Pero... polinomio de grado **k** tiene **k** soluciones...

1.3.1. Ecuaciones lineales homogéneas.

- Sean las soluciones de la ecuación carácterística:

  $x = (s_1, s_2, ..., s_k)$ (por ahora todas distintas)

- Soluciones de la ecuación recurrente:

  $s_1^n, s_2^n, ..., s_k^n,$

  pero también son solución sus **combinaciones lineales:**

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

- Siendo c_i constantes, cuyos valores dependen de las condiciones iniciales (casos base y otros – ver 1.3.5).
- Son constantes que añadimos. Debemos resolverlas, usando los casos base de la ecuación recurrente.

1.3.1. Ecuaciones lineales homogéneas.

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

$$t(n) = \begin{cases} 0 & \text{Si } n = 0 \\ 1 & \text{Si } n = 1 \\ 3 \cdot t(n-1) + 4 \cdot t(n-2) & \text{Si } n > 1 \end{cases}$$

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

1.3.1. Ecuaciones lineales homogéneas.

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

$$t(n) = \begin{cases} 0 & \text{Si } n = 0 \\ 1 & \text{Si } n = 1 \\ 3 \cdot t(n-1) + 4 \cdot t(n-2) & \text{Si } n > 1 \end{cases}$$

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

- ¿Qué pasa si no todas las soluciones son distintas?

1.3.1. Ecuaciones lineales homogéneas.

- 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 1.** Calcular t(n) y el orden para:

  $t(n) = 2t(n-1)-t(n-2)$; $t(0)=t(1)=1$

- **Ejemplo 2.** Calcular t(n) y orden para:

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

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

PISTA: "teorema de las raíces racionales": raíces son de la forma **p/q**, con p divisor de a_k (término independiente) y q divisor de a_0

1.3.2. Recurrencias no homogéneas.

- **Regla:** Si en la ecuación de recurrencia aparece un término de la forma $\mathbf{b^n \cdot p(n)}$ (p(n) polinomio de n), entonces en la ecuación característica habrá un factor:

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

- **OJO:**
  - $1=1^n$ puede estar implícito, por ej., $t(n)-t(n-1) = 3$
  - Agrupar términos con misma base b

- **Ejemplo:** $t(n) - t(n-3) = 2 + n^3 + n^2 \cdot 3^n + 2^{(n+1)} + 8n^2$
- ¿Cuál es la ecuación característica?

1.3.2. 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)).

  $t(n) = 1 + n$ $\qquad$ $n = 0, 1$

  $t(n) = 4t(n-2) + (n+5)3^n + n^2$ $\qquad$ Si $n>1$

1.3.3. Cambio de variable.

$$t(n) = a\cdot t(n/2) + b\cdot t(n/4) + ...$$

Cambio de variable:

- Convertir las ecuaciones anteriores en algo de la forma $t'(k) = a\cdot t'(k-1) + b\cdot t'(k-2) + ...$
- Resolver el sistema en **k**.
- Deshacer el cambio, y obtener el resultado en **n**

Cambios típicos:

- $n = 2^k$; $k = \log_2 n$
- $n = 3^k$, $k = \log_3 n$
- $n = 5k$; $k = n/5$

1.3.3. Cambio de variable.

- **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

1.3.3. Cambio de variable.

- Los órdenes que obtenemos son condicionados a que se cumplan las condiciones del cambio:

$$t(n) \in \Theta(f \mid P(n))$$

- ¿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 $f(bn) \in \Theta(f(n))$ (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 $t(n) \in \Theta(f(n))$.

1.3.4. 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:

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

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

1.3.4. Otras técnicas.

Expansión de recurrencias

- Puede ser adecuada cuando sólo hay un término recurrente o cuando la ecuación es no lineal.

- **Ejemplo.**

  $t(0) = 1$

  $t(n) = n\ t(n-1) + 1$

- No aplicar si aparecen varios términos recurrentes:

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

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

1.3.4. 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.
  - **Caso base.** Para algún valor pequeño, $t(n) \leq c_1 \cdot f(n)$
  - **Caso general.** Suponiendo que $t(n-1) \leq c_1 \cdot f(n-1)$, entonces se demuestra que $t(n) \leq c_1 \cdot f(n)$

1.3.4. Otras técnicas.

Inducción constructiva

- **Ejemplo**. Dada la siguiente ecuación recurrente, 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 $t(n) \in O(n!)$.

1.3.5. Condiciones iniciales.

- ¿Cuál es el significado de las condiciones iniciales?
- **Condición inicial:** caso base de una ecuación recurrente.
- ¿Cuántas aplicar?
  - Tantas como constantes indeterminadas.
  - **n** incógnitas, **n** ecuaciones: sistema determinado. Aplicamos el método de Cramer.
- ¿Cuáles aplicar?
  - Las condiciones aplicadas se deben poder alcanzar desde el caso general.
  - Si se ha aplicado un cambio de variable, deben cumplir las condiciones del cambio.

1.3.5. Condiciones iniciales.

- **Ejemplo.**

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

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

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

- Aplicar condiciones iniciales para despejar c_1, c_2, c_3.
- ¿Cuántas aplicar? ¿Cuáles?
- ¿Y si cambiamos "$n > 10$" por "$n > 4$"?

1.3.5. Condiciones iniciales.

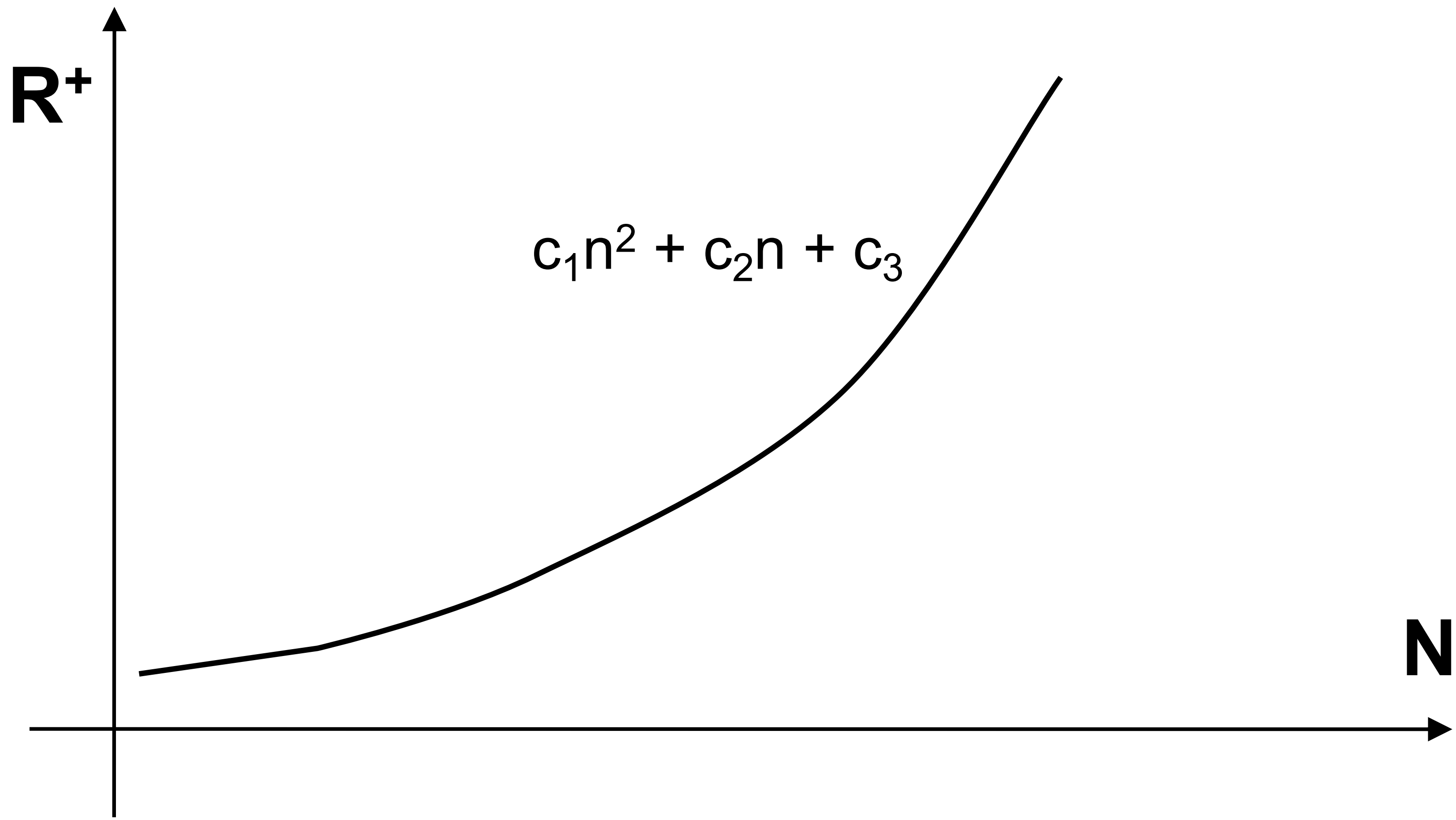
- El cálculo de constantes también se puede aplicar en el estudio experimental de algoritmos.

- **Proceso**

1. Hacer una estimación teórica del tiempo de ejecución.
2. Expresar el tiempo en función de constantes indefinidas.
3. Tomar medidas del tiempo de ejecución para distintos tamaños de entrada.
4. Resolver las constantes.

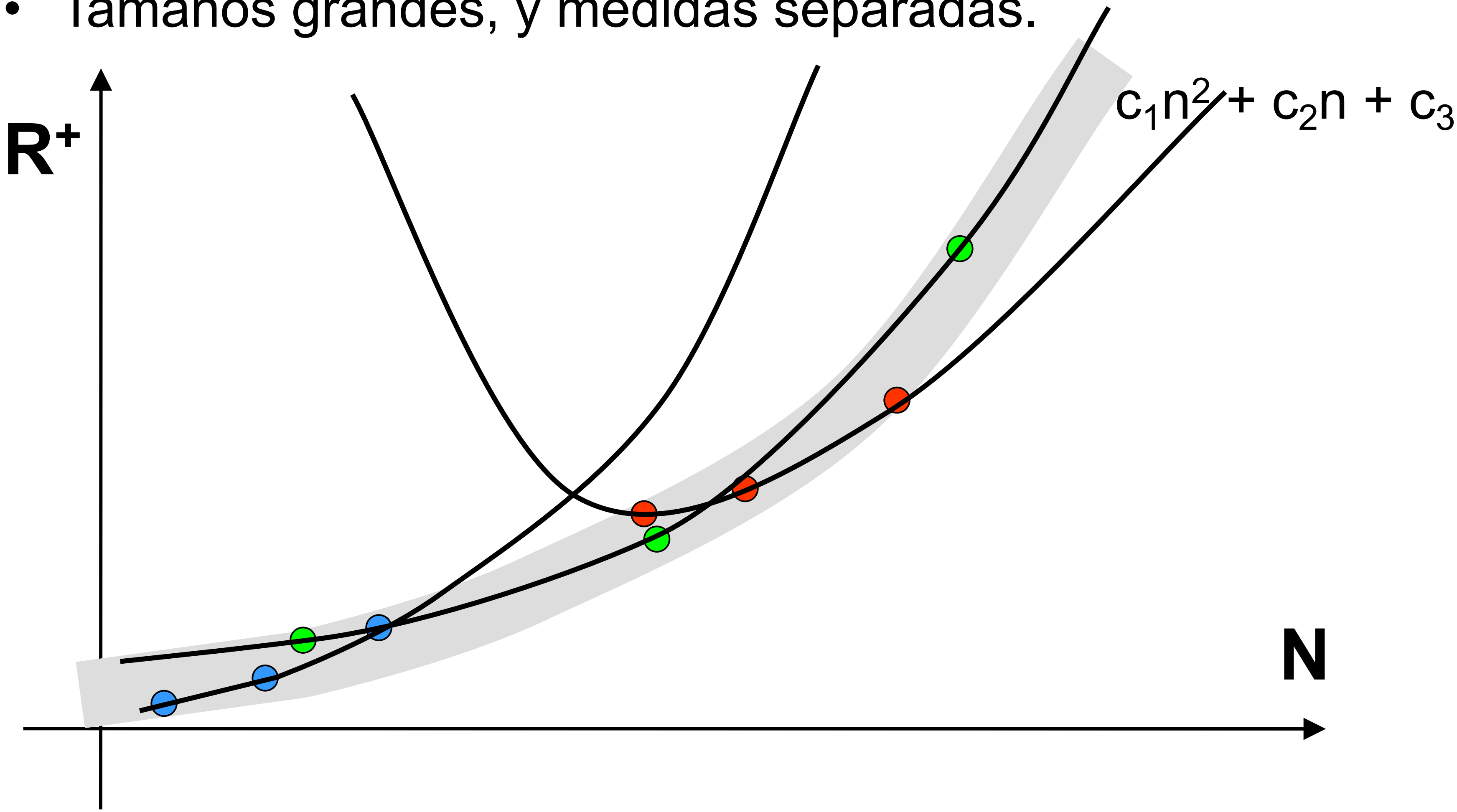
1.3.5. Condiciones iniciales.

- **Ejemplo:** $t(n) = a(n+1)^2 + (b+c)n + d$
- Simplificamos constantes: $t(n) = c_1n^2 + c_2n + c_3$



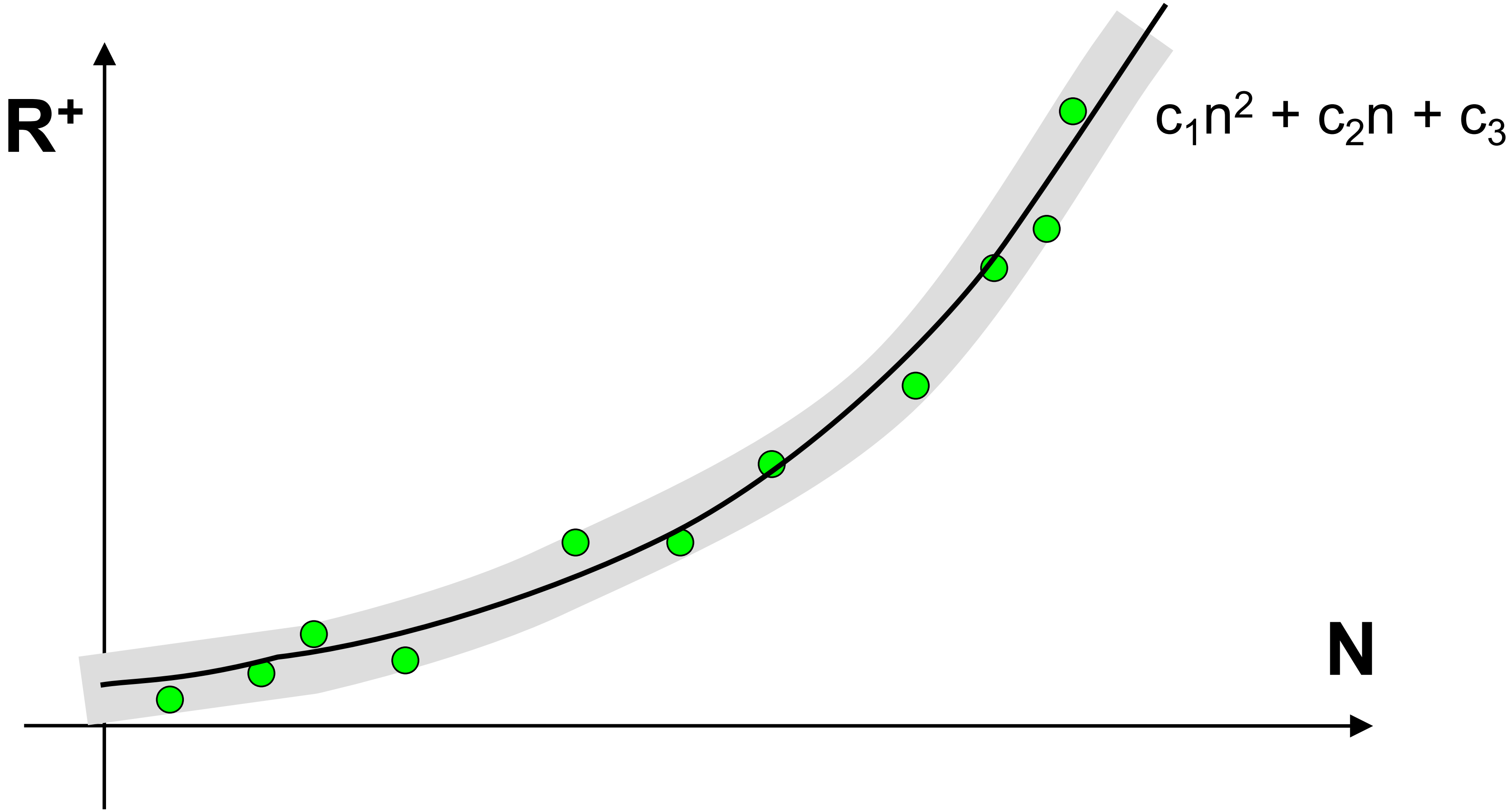
1.3.5. Condiciones iniciales.

- **Ajuste sencillo:** Tomar 3 medidas de tiempo.
  3 incógnitas y 3 ecuaciones: resolvemos c_1,c_2,c_3.
- Tamaños grandes, y medidas separadas.



1.3.5. Condiciones iniciales.

- **Ajuste preciso:** Tomar muchas medidas de tiempo.
- Hacer un ajuste de regresión.




LANDSCAPEHDWALLS.COM

1. Análisis de algoritmos.

Conclusiones:

- **Eficiencia:** consumo de recursos en función de los resultados obtenidos.
- **Recursos consumidos** por un algoritmo: fundamentalmente tiempo de ejecución y memoria.
- La estimación del tiempo, **t(n)**, es aproximada, parametrizada según el tamaño y el caso (t_m, t_M, t_p).
- **Conteo de instrucciones:** obtenemos como resultado la función **t(n)**
- Para simplificar se usan las notaciones asintóticas: $O(t)$, $\Omega(t)$, $\Theta(t)$, $o(t)$.

1. Análisis de algoritmos.

Conclusiones:

- **Ecuaciones recurrentes:** surgen normalmente del conteo (de tiempo o memoria) de algoritmos recursivos.
- Tipos de ecuaciones recurrentes:
  - Lineales, homogéneas o no homogéneas.
  - No lineales (menos comunes en el análisis de algoritmos).
- **Resolución de ecuaciones recurrentes**:
  - Método de expansión de recurrencias (el más sencillo).
  - Método de la ecuación característica (lineales).
  - Cambio de variable (previo a la ec. característica) o transformación de la imagen.
  - Inducción constructiva (general pero difícil de aplicar).