Algorítmica

Tema 1. Planteamiento General

Tema 2. La Eficiencia de los Algoritmos

Tema 3. Algoritmos "Divide y vencerás"

Tema 4. Algoritmos Voraces ("Greedy")

Tema 5. Algoritmos basados en Programación Dinámica

Tema 6. Algoritmos para la Exploración de Grafos ("Backtracking", "Branch and Bound")

Tema 7. Otras metodologías algorítmicas

Tema 2: La Eficiencia de los Algoritmos

Bibliografía:

G. BRASSARD, P. BRATLEY. Fundamentos de Algoritmia. Prentice Hall (1997).

Objetivos

- Principio de Invarianza
- Concepto de eficiencia
- Concepto de operación elemental
- Notaciones asintóticas
- Saber calcular la eficiencia de un algoritmo
- Dominar la técnica de la ecuación recurrente

Comparación de algoritmos

- Es frecuente disponer de más de un algoritmo para resolver un problema dado:
- ¿Con cuál nos quedamos?
- Estudio de los recursos:
  - **Tiempo**
  - Espacio

Uso de recursos

- Los recursos empleados se distribuyen en:
  - Diseño
  - Implementación
  - Explotación

Factores que influyen en el uso de los recursos

- Hardware: arquitectura, velocidad, etc.
- Codificador: Calidad del código creado
- Compilador: Calidad del código generado
- Diseñador: algoritmo
- Tamaño de las entradas

Principio de Invarianza

- Dos implementaciones de un mismo algoritmo no diferirán más que en una constante multiplicativa.
- Si $t_1(n)$ y $t_2(n)$ son los tiempos de dos implementaciones de un mismo algoritmo, se verifica:

$$\exists c, d \in \Re, \quad t_1(n) \leq c t_2(n); \quad t_2(n) \leq d t_1(n)$$

Eficiencia

- Medida del uso de los recursos en función del tamaño de las entradas
- T(n): tiempo empleado para una entrada de tamaño n

Eficiencia (II)

- Dos algoritmos para un mismo problema:
- Algoritmo 1: $T(n) = 10^{-4}\ 2^n$ segs. (la constante 10^{-4} representa la velocidad de la máquina.)

  $n=38$, $T(n) = 1$ año
- Algoritmo 2: $T(n) = 10^{-2}\ n^3$ segs. (la constante 10^{-2} representa la velocidad de la máquina.)

  $n=1000$, $T(n) = 1$ año
- Se precisa **análisis asintótico**

- **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...

- **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...

- Factores que influyen en el consumo de recursos:
  - **Factores externos.**
    - El ordenador donde se ejecute.
    - 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. Procesar un fichero de blog: número de mensajes.
  - **Contenido de los datos de entrada.**
    - **Mejor caso (t_m).** El contenido favorece una rápida ejecución.
    - **Peor caso (t_M).** La ejecución más lenta posible.
    - **Caso promedio (t_p).** Media de todos los posibles contenidos.

- Los factores externos no aportan información sobre el algoritmo.
- **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 de búsqueda secuencial.
  - Mejor caso. Se encuentra x en la 1ª posición:

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

    $t_M(N) = b \cdot N + c$
- **Ojo**: El mejor caso no significa tamaño pequeño.

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.

- El proceso básico de análisis de la eficiencia algorítmica es el conocido como **conteo de instrucciones** (**o de memoria**).

- **Conteo de instrucciones:** Seguir la ejecución del algoritmo, sumando las instrucciones que se ejecutan.

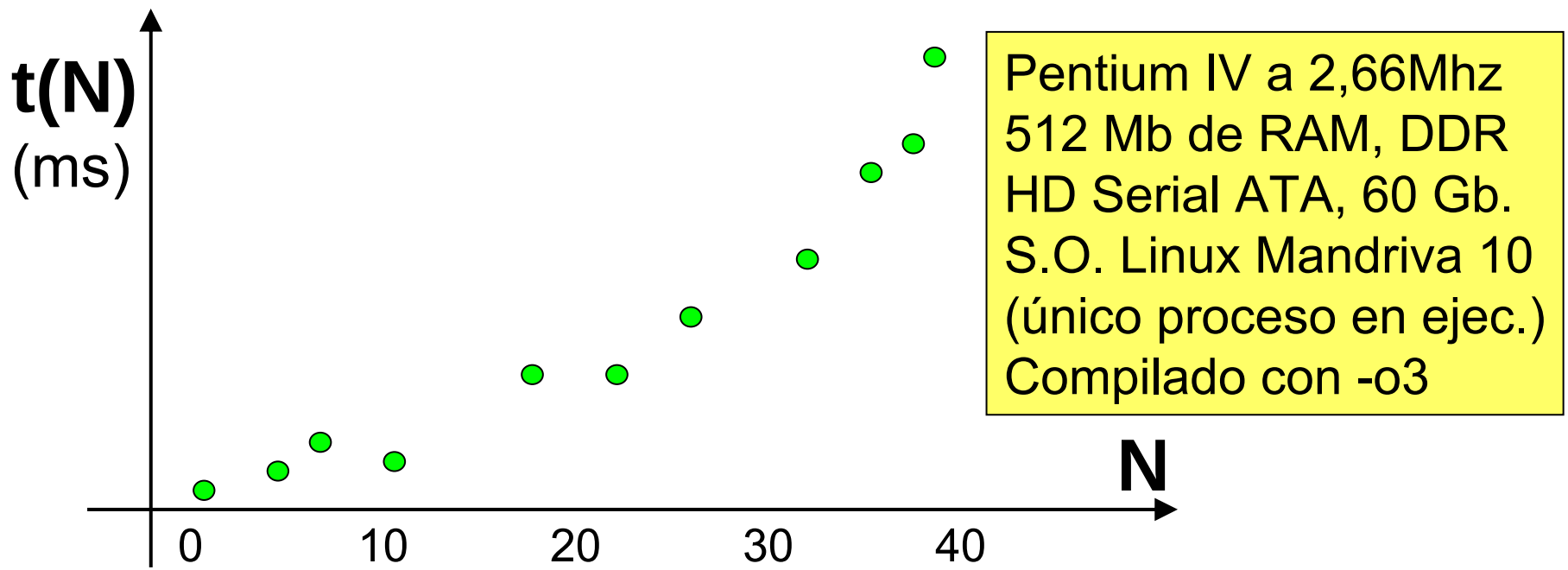
- **Conteo de memoria:** Lo mismo. Normalmente interesa el máximo uso de memoria requerido.

Medidas de la eficiencia

- Enfoques de medida:
  - Enfoque práctico (*a posteriori*)
  - Enfoque teórico (*a priori*)
  - Enfoque híbrido
- Tipos de Análisis:
  - Peor caso
  - Mejor caso
  - Caso promedio
  - Análisis probabilístico
  - Análisis amortizado

Enfoque a posteriori

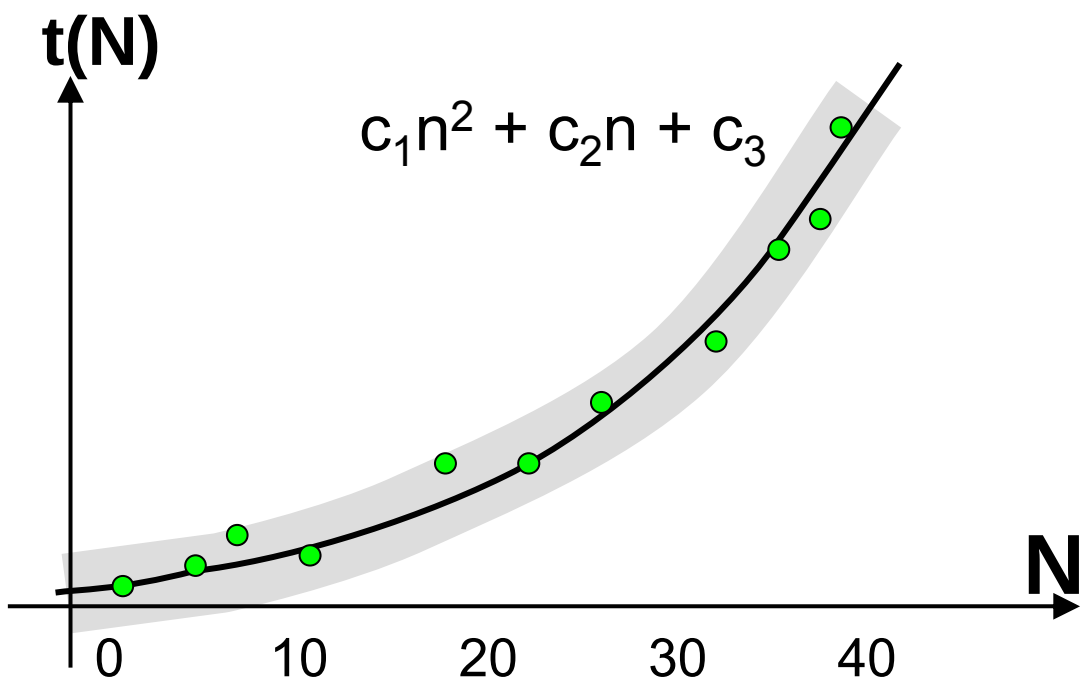
- El análisis de algoritmos puede hacerse **a posteriori:** implementar el algoritmo y contar lo que tarda para distintas entradas.
- En este caso, cobran especial importancia las **herramientas** de la **estadística**: representaciones gráficas, técnicas de muestreo, regresiones, tests de hipótesis, etc.
- Hay que ser muy **específicos**, indicar: ordenador, S.O., condiciones de ejecución, opciones de compilación, etc.



Enfoque a posteriori

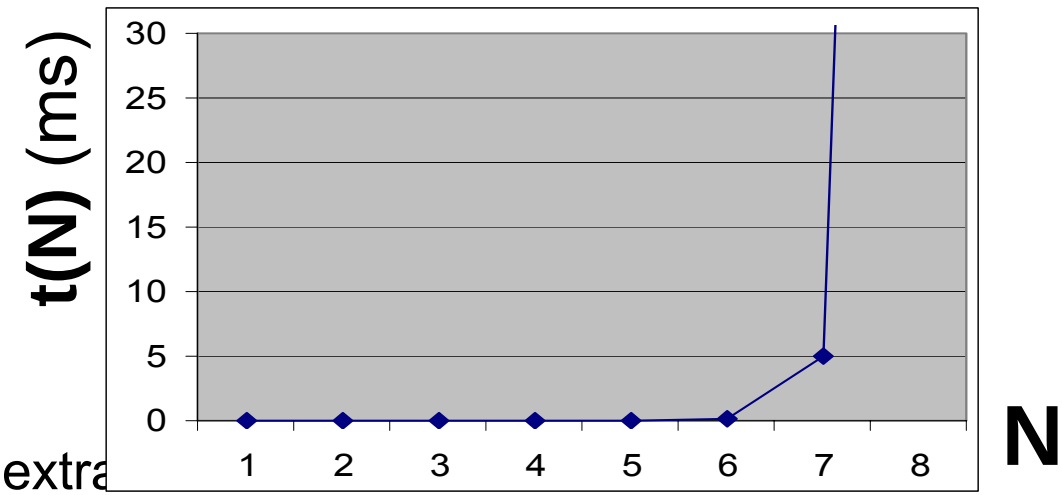
- Indicamos los **factores externos**, porque influyen en los tiempos (multiplicativamente), y son útiles para comparar tiempos tomados bajo condiciones distintas.
- La medición de los tiempos es un **estudio experimental**.
- El análisis a posteriori suele complementarse con un **estudio teórico** y un **contraste teórico/experimental**.

- **Ejemplo**. Haciendo el estudio teórico de un 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?



Enfoque a posteriori

- El 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.** Para un cierto programa:
  - 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 extra
- 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.

Notación asintótica

- Estudia el comportamiento del algoritmo cuando el tamaño de las entradas, n, es los suficientemente grande, sin tener en cuenta lo que ocurre para entradas pequeñas y obviando factores constantes
- Notaciones: O, Ω, Θ (o, omega y tita)

Órdenes de eficiencia

Un algoritmo tiene un *tiempo de ejecución de orden* T(n), para una función dada T, si existe una constante positiva c, y una implementación del algoritmo capaz de resolver cada caso del problema en un tiempo acotado superiormente por cT(n), donde n es el tamaño del problema considerado

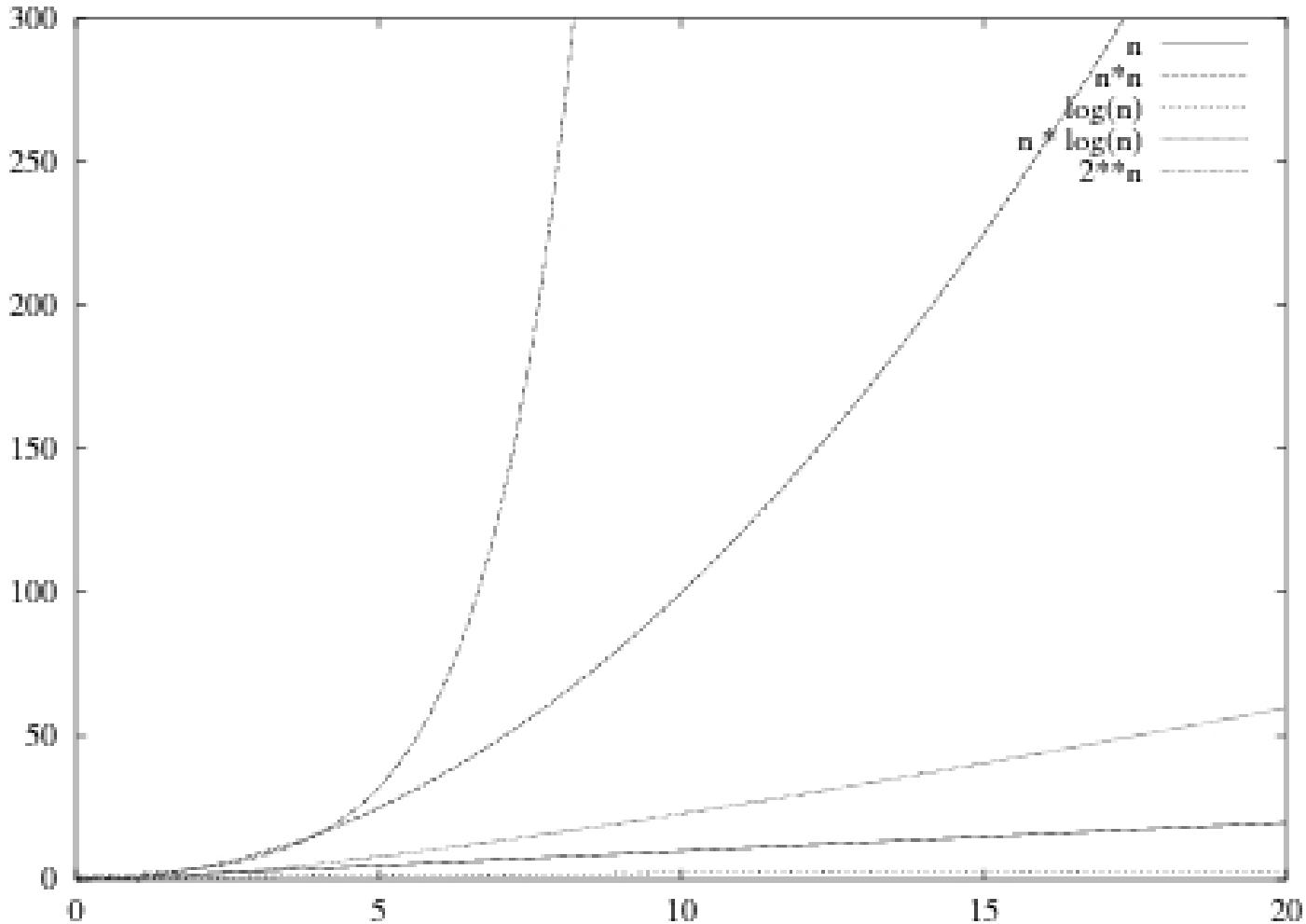
Órdenes más habituales

- Lineal: n
- Cuadrático: n^2
- Polinómico: n^k, con $k \in \mathbb{N}$
- Logarítimo: log(n)
- Exponencial: c^n

log(n)	n	n^2	n^5	2^n
1	2	4	32	4
2	4	16	1024	16
3	8	64	32768	256
4	16	256	1048576	65536
5	32	1024	33554432	4.29E+09
6	64	4096	1.07E+09	1.84E+19
7	128	16384	3.44E+10	3.4E+38
8	256	65536	1.1E+12	1.16E+77
9	512	262144	3.52E+13	1.3E+154
10	1024	1048576	1.13E+15	#NUM!

$$1 < log(n) < log^2(n) < \sqrt{n} < n < n*log(n) < n^2 < 2^n$$

Órdenes más habituales (gráfico)



- 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.

Notación O-Mayúscula

- Decimos que una función T(n) es O(f(n)) si existen constantes n_0 y c tales que $T(n) \leq cf(n)$ para $n \geq n_0$:

- T(n) es O(f(n)) $\Leftrightarrow$

  $\exists c \in R$, $\exists n_0 \in N$, tal que $\forall n \geq n_0$, $T(n) \leq cf(n)$

Ejemplos en O(.)

- $T(n) = (n + 1)^2$ es $O(n^2)$
- $T(n) = 3n^3 + 2n^2$ es $O(n^3)$
- $T(n) = 3^n$ no es $O(2^n)$

Flexiblidad en la notación: Emplearemos la notación O(f(n)) aun cuando en un número finito de valores de n, f(n) sea negativa o no esté definida. Ej.: n/log(n)

Interpretación de la definición de O

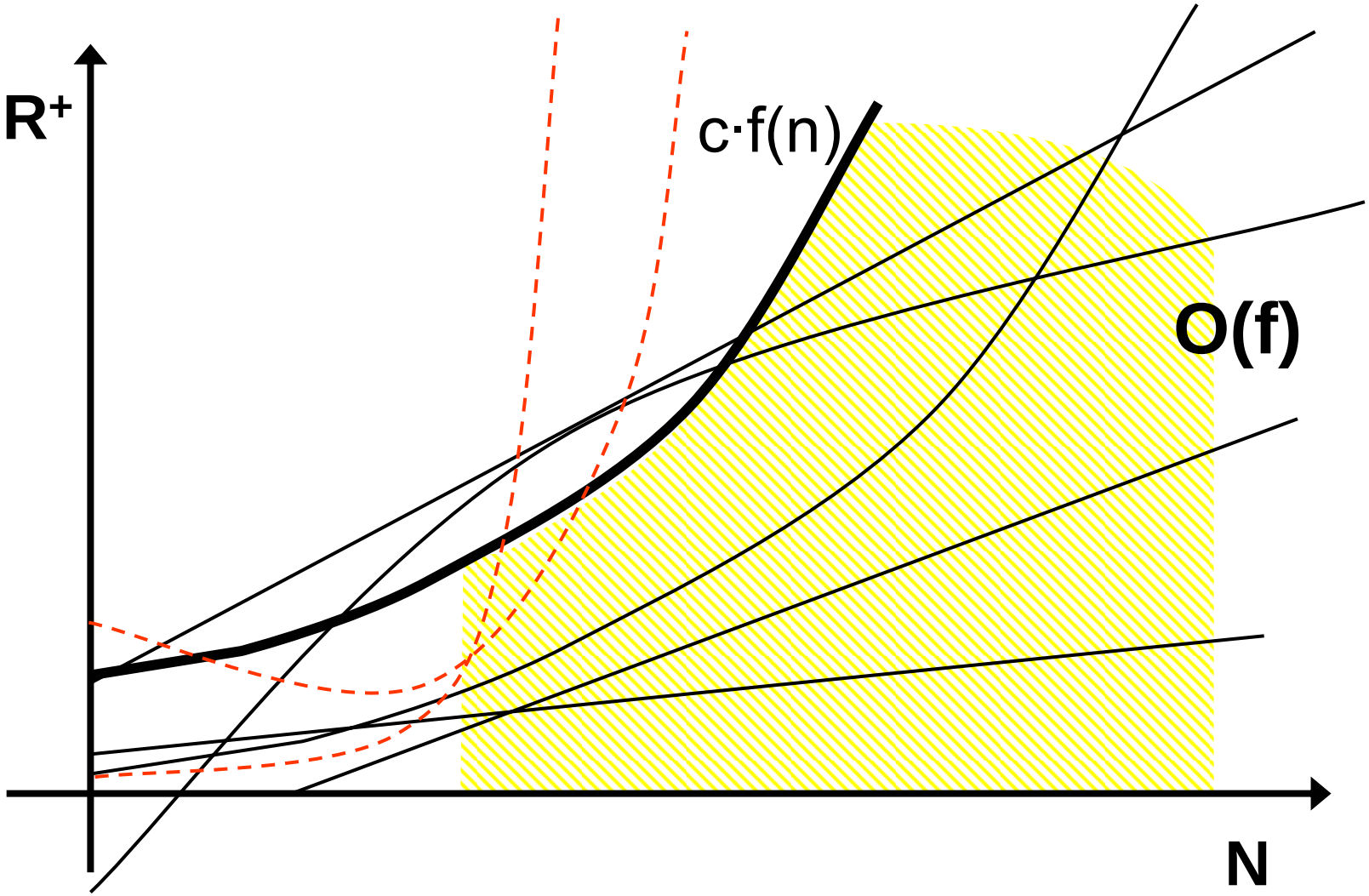
Orden de complejidad de f(n): O(f)

- 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.

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

Observaciones:

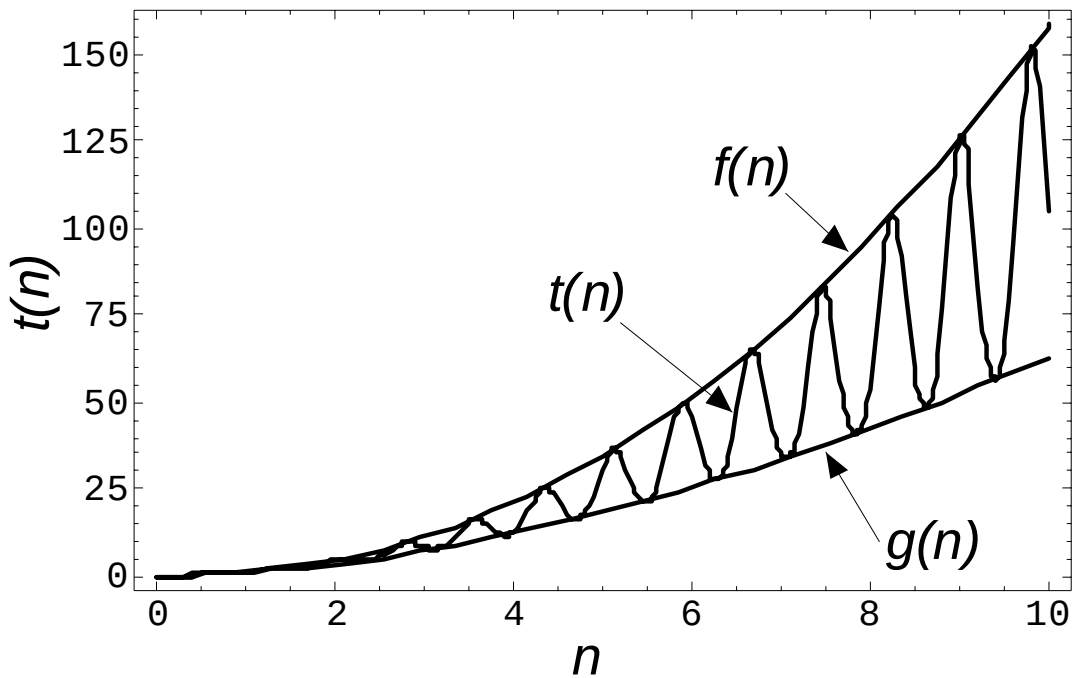
- **O(f)** es un **conjunto de funciones**, no una función.
- “Valores de n suficientemente grandes...”: no nos importa lo que pase para valores pequeños.
- “Funciones acotadas superiormente por un múltiplo de f...”: nos quitamos las constantes multiplicativas.
- La definición es aplicable a cualquier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


R+
c·f(n)
O(f)
N

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**. $t(n) = 2n^2/5 + 6n + 3\pi \cdot \log_2 n + 2 \Rightarrow t(n) \in O(n^2)$

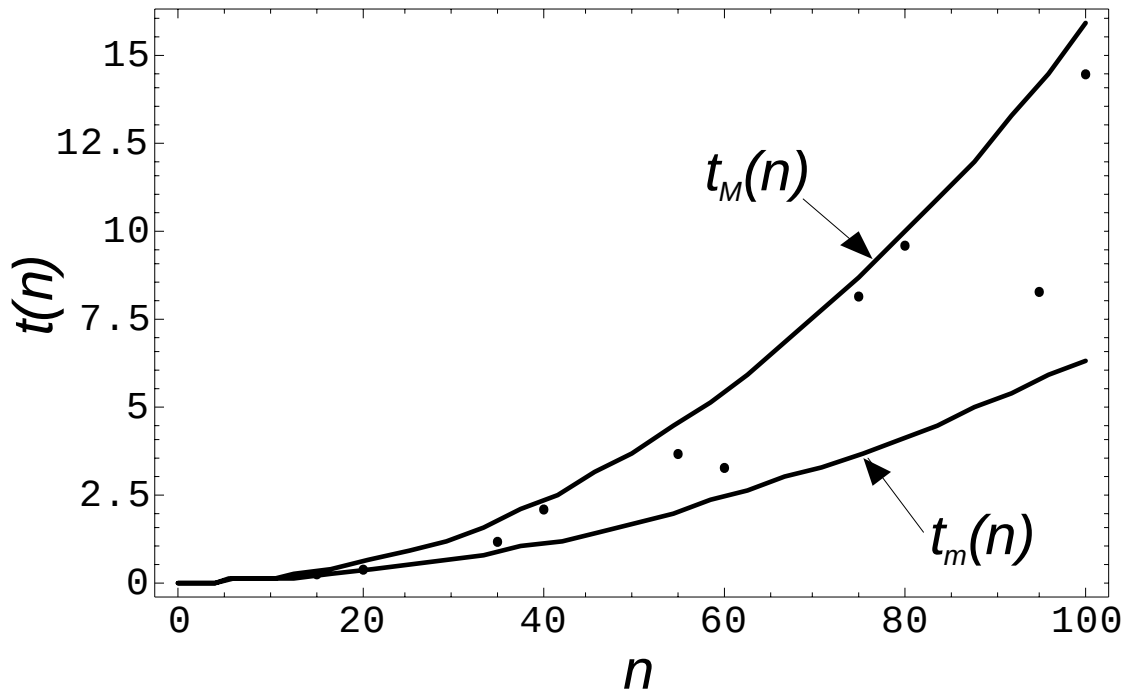
- 2) Acotar una función difícil de calcular con precisión.
- **Ejemplo.** $t(n) \in O(f(n))$

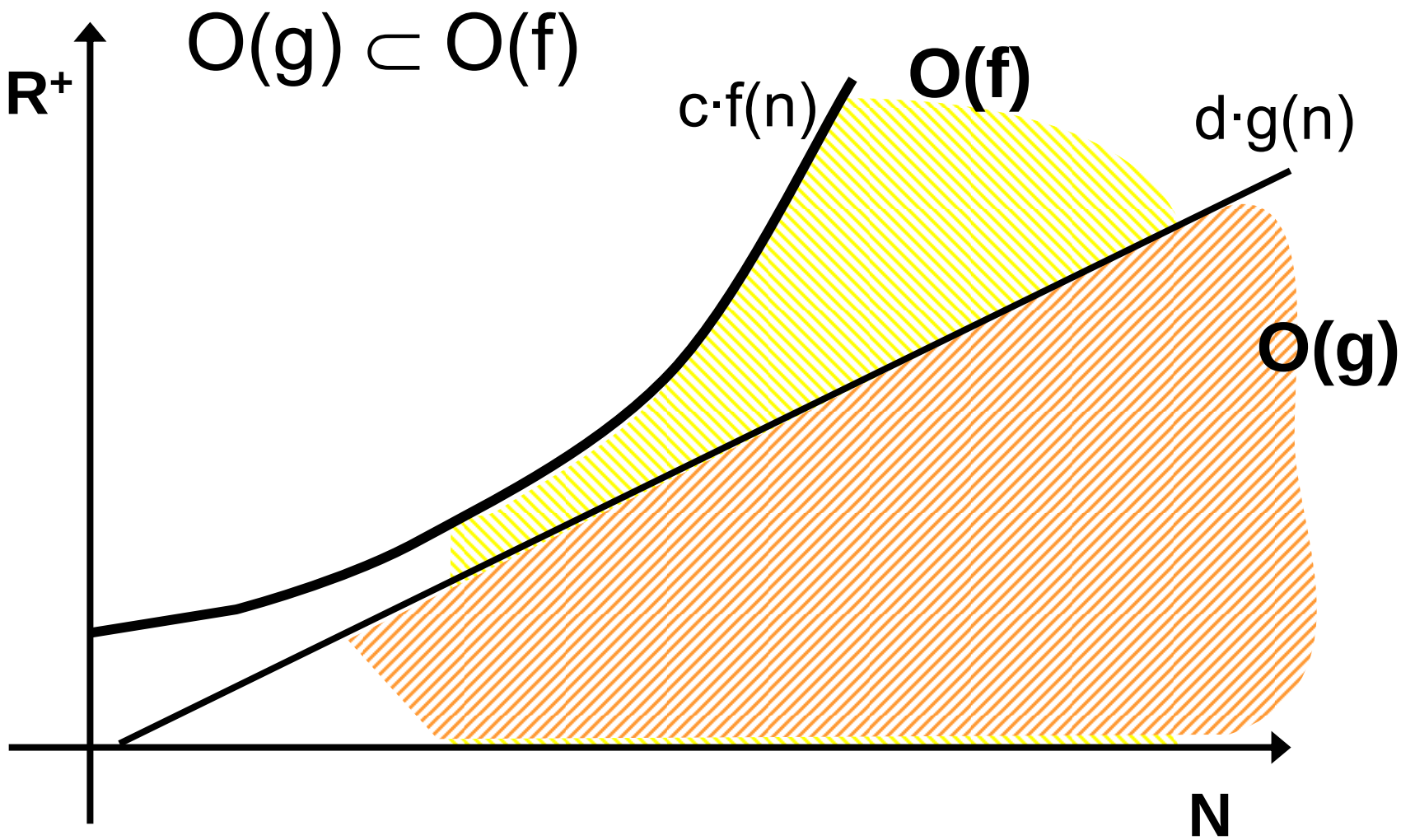


Uso de los órdenes de complejidad

- 3) Acotar una función que no tarda lo mismo para el mismo tamaño de entrada (distintos casos, mejor y peor).

- **Ejemplo.**
  $t(n) \in O(t_M(n))$





Relación de orden entre O(..) = Relación de inclusión entre conjuntos.

$O(g) \leq O(f) \Leftrightarrow O(g) \subseteq O(f) \Leftrightarrow$ Para toda $t \in O(g)$, $t \in O(f)$

Notaciones Ω y Θ

Notación Ω: cota inferior:

T(n) es $\Omega(f(n))$ cuando $\exists c \in R^+, \exists n_0 \in N: \forall n \geq n_0 \Rightarrow T(n) \geq cf(n)$

Notación Θ: Orden exacto

T(n) es $\Theta(f(n))$ cuando

T(n) es O(f(n)) y T(n) es $\Omega(f(n))$

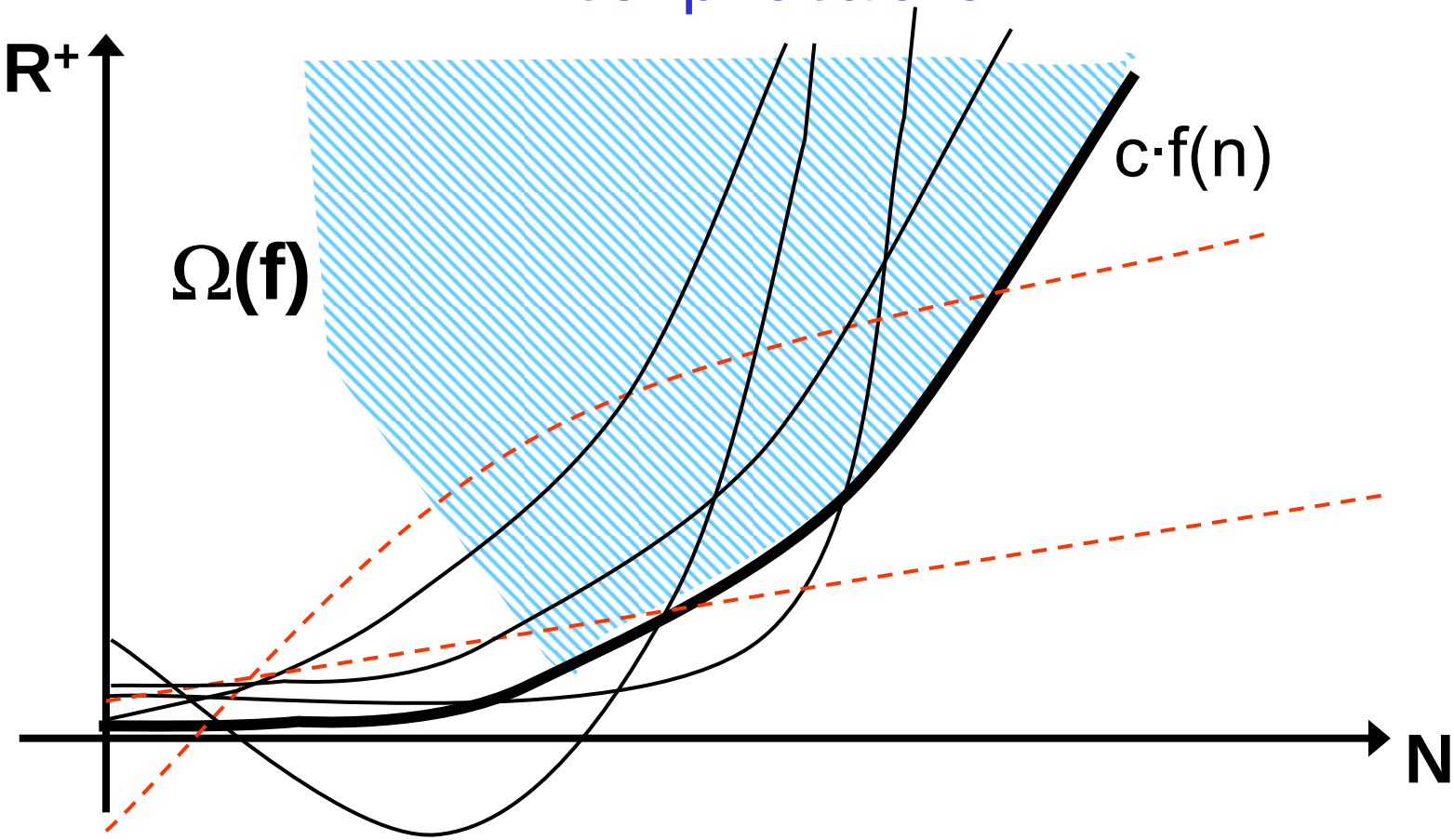
Interpretación

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.

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

Interpretación



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

Interpretación

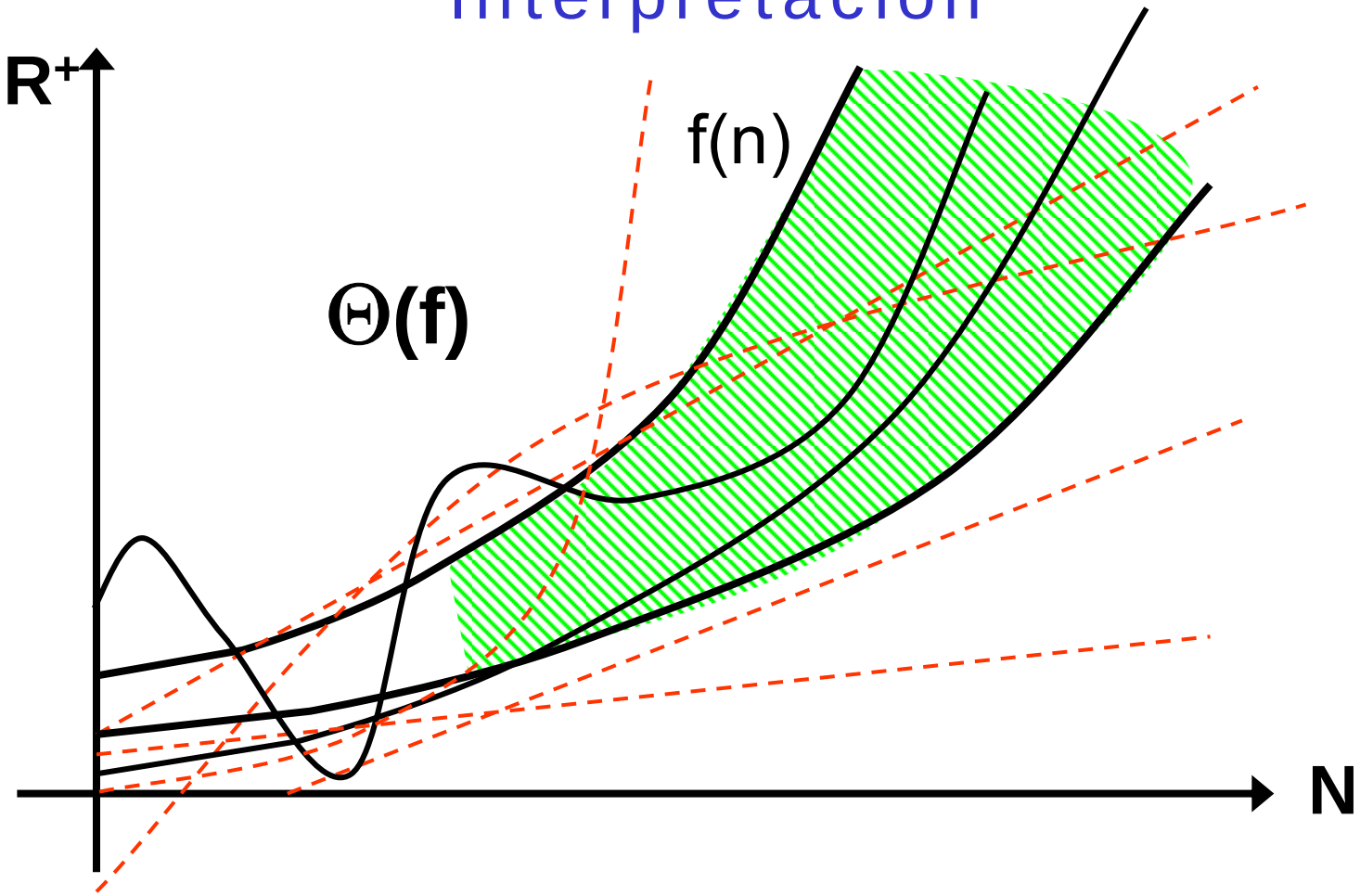
Orden exacto de f(n): $\Theta(f)$

- 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.

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

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

Interpretación



- Si un algoritmo tiene un **t** tal que t ∈ O(f) y t ∈ Ω(f), entonces **t** ∈ **Θ(f)**.

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

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

Propiedades de las notaciones asintóticas.

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

  – Si $f \in \Omega(g)$ 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 $O(f) \subseteq O(g)$.

- **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) $O(f) \subseteq O(g) \Leftrightarrow f \in O(g)$

Propiedades de las notaciones asintóticas.

- **P4. Propiedad del máximo.**
  Dadas f y g, de N en R^+, $O(f+g) = O(\max(f, g))$.
  - Con omegas: $\Omega(f+g) = \Omega(\max(f, g))$

- **P5. 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) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty \Rightarrow O(f) \supset O(g)$

Notaciones con varios parámetros.

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

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.

- **Ejemplo.** Tiempo de ejecución de la BPP con listas de adyacencia: **O(n+a)**.
  Memoria usada en una tabla hash: depende del número de cubetas, elementos, tamaño de celda...

- Podemos extender los conceptos de $\Omega(f)$ y $\Theta(f)$, para funciones con varios parámetros.
- Las propiedades se siguen cumpliendo
- Ejemplos:

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

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:

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

Notaciones condicionales.

- De igual forma, tenemos $\Omega(f \mid P)$ y $\Theta(f \mid P)$.
- **Ejemplo.**
  - Si estudiamos el tiempo para tamaños de entrada que sean potencia de 2:
    $t(n) \in O(f \mid n = 2^k)$
  - Para tamaños que sean múltiplos de 2:
    $t(n) \in O(f \mid n = 2k)$

Cotas de complejidad frecuentes.

- **Algunas relaciones entre órdenes frecuentes.**

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

- Se establece una jerarquía de órdenes

Cotas de complejidad frecuentes.

- El orden de un polinomio $a_n x^n + \ldots + a_1 x + a_0$ 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.

Cálculo del Orden de Eficiencia

Operación elemental

- Operación de un algoritmo cuyo tiempo de ejecución se puede acotar superiormente por una constante.

  En nuestro análisis sólo contará el número de operaciones elementales y no el tiempo exacto necesario para cada una de ellas.

Consideraciones sobre operaciones elementales

- En la descripción de un algoritmo puede ocurrir que una línea de código corresponda a un número variable de operaciones elementales.

  Por ejemplo, si A es un vector con n elementos, y queremos calcular

  x = max{A[k], 0 <= k < n}

  el tiempo para hacerlo depende de n, no es constante

Consideraciones sobre operaciones elementales

- Algunas operaciones matemáticas no deben ser tratadas como tales operaciones elementales. Por ejemplo, el tiempo necesario para realizar sumas y productos crece con la longitud de los operandos. Sin embargo, en la práctica se consideran elementales siempre que los datos que se usen tengan un tamaño razonable.

- No obstante, consideraremos las operaciones suma, diferencia, producto, cociente, módulo, operaciones booleanas, comparaciones y asignaciones como elementales, salvo que explícitamente se establezca otra cosa.

Reglas del cálculo del tiempo de ejecución

1. Sentencias simples
2. Bucles
3. Sentencias condicionales
4. Bloques de sentencias
5. Llamadas a funciones
6. Funciones recursivas

Sentencias simples

- Consideraremos que cualquier sentencia simple (lectura, escritura, asignación, etc.) va a consumir un tiempo constante, O(1), salvo que contenga una llamada a función

- Número de instrucciones t(n) → sumar 1 por cada instrucción o línea de código de ejecución constante.
- Tiempo de ejecución t(n) → sumar una constante (c_1, c_2, ...) por cada tipo de instrucción o grupo de instrucciones secuenciales.

Bucles

- El tiempo invertido en un bucle es la suma del tiempo invertido en cada iteración. Este tiempo debería incluir: el tiempo del cuerpo y el asociado a la evaluación de la condición y actualización.
- En un bucle donde todas las iteraciones son iguales, el tiempo total será el producto del número de iteraciones por el tiempo que requiere cada una.

Introducción.

- **Bucles FOR**: Se pueden expresar como un sumatorio, con los límites del FOR como límites del sumatorio.

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

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

- **Bucles WHILE y REPEAT**: cota inferior y superior del número de ejecuciones ¿Se puede convertir en un FOR?

Sentencias condicionales

- El tiempo de ejecución es el máximo tiempo de la parte if y de la parte else, de forma que si son, respectivamente, O(f(n)) y O(g(n)), será O(max(f(n), g(n))).

Bloques de sentencias

- Se aplica la regla de la suma, de forma que se calcula el tiempo de ejecución tomando el máximo de los tiempos de ejecución de cada una de las partes (sentencias individuales, bucles o condicionales) en que puede dividirse.

Llamadas a funciones

- Si una determinada función P tiene una eficiencia de O(f(n)) con n la medida del tamaño de los argumentos, cualquier función que llame a P tiene en la llamada una cota superior de eficiencia de O(f(n))
  - Las asignaciones con diversas llamadas a función deben sumar las cotas de tiempo de ejecución de cada llamada
  - La misma consideración es válida para las condiciones de bucles y sentencias condicionales

Ejemplos

- **Ejemplos**. Estudiar t(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
```

Introducción.

- **Ejemplos**. Estudiar t(n).

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

Funciones recursivas

- Las funciones de tiempo asociadas son también recursivas.
- Ejemplo:

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

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)
```

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:

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

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

Solución de ecuaciones recursivas
Resolución de recurrencias asintóticas

- Ecuación característica
- Recurrencias homogéneas
- Recurrencias no homogéneas
- Cambios de variable
- Transformación de rangos

Funciones recursivas (I)

- Para analizar el tiempo de ejecución de un procedimiento recursivo, le asociamos una función de eficiencia desconocida, T(n), y la estimamos a partir de T(k) para distintos valores de k.

Función factorial

```
1: int fact(int n) {
2:   if (n <= 1)
3:     return 1;
4:   else
5:     return (n * fact(n – 1));
6: }
```

Llamamos T(n) al tiempo de ejecución de fact(n).

Las líneas 2 y 3 son operaciones elementales, sean c y d sus tiempos de ejecución

Función factorial (II)

$$
\begin{aligned}
T(n) &= c + T(n - 1) \\
&= c + (c + T(n-2)) = 2c + T(n-2) \\
&= 2c + (c + T(n-3)) = 3c + T(n-3) \\
&\ldots \\
&= ic + T(n-i) \\
&\ldots \\
&= (n-1)c + T(n - (n-1)) = (n-1)c + c + d \\
&= nc + d
\end{aligned}
$$

De donde T(n) es O(n)

Ejemplo

```
1:      int E(int n) {
2:      if (n == 1)
3:              return 0;
4:      else
5:              return E(n/2) + 1;
6:      }
```

$$T(n) = \begin{cases} 1, & n = 1 \\ 1 + T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

T(n) es $O(\log_2(n))$

Ejemplo de recurrencia

$$T(n) = T\left(\frac{n}{2}\right) + n^2, \quad n \geq 2, \quad T(1) = 1$$

- Cambio de variable: $n = 2^m$; $n^2 = (2^m)^2 = (2^2)^m = 4^m$

  $T(2^m) = T(2^{m-1}) + 4^m =$

  $= T(2^{m-2}) + 4^{m-1} + 4^m$

  ...

  $= T(2^{m-i}) + [4^{m-(i-1)} + \ldots + 4^{m-1} + 4^m]$

Ejemplo

$$T(2^m) = T(1) + [4^1 + \cdots + 4^{m-2} + 4^{m-1} + 4^m]$$

$$= \sum_{i=0}^{m} 4^i = \frac{4^{m+1} - 1}{4 - 1} = \frac{4}{3} 4^m - \frac{1}{3}$$

$$= [4^m = n^2] = \frac{4}{3} n^2 - \frac{1}{3}$$

T(n) es $O(n^2)$

Resolución de recurrencias

- Desarrollar; intentar encontrar la expresión general; resolver.
- Método de la ecuación característica

Resolución de recurrencias

- 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 homegéneas:

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

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

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

Recurrencias homogéneas

- Consideramos recurrencias *homogéneas lineales con coeficientes constantes*:

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = 0$$

- Lineales: no hay términos $t_{n-1} t_{n-j}, t^2_{n-i}$
- Homogénea: igualada a 0
- Con coeficientes constantes: a_i son ctes
- Ejemplo (sucesión de Fibonacci)

$$f_n = f_{n-1} + f_{n-2} \quad \Rightarrow \quad f_n - f_{n-1} - f_{n-2} = 0$$

- *Observación*: las combinaciones lineales de las soluciones de la recurrencia también son soluciones

Ecuaciones lineales homogéneas.

- La ecuación de recurrencia es de la forma:

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

- **Caso sencillo:**

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

- **Solución:** $t(n) = x^n$

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

Ecuaciones lineales homogéneas.

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

Ecuación característica de la ecuación recurrente lineal homogénea

- **k**: conocida. $\mathbf{a_i}$: 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 los casos base (condiciones iniciales).

- Son constantes que añadimos nosotros. Debemos resolverlas, usando los casos base de la ecuación recurrente.

En resumen:

Suposición $t_n = x^n$

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

satisfecha si x=0 o (ecuación característica)

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

Polinomio característico

$$p(x) = a_0 x^k + a_1 x^{k-1} + \cdots + a_k$$

En resumen:

(Teorema fundamental del Álgebra)

$$p(x) = \prod_{i=1}^{k} (x - r_i)$$

Consideremos una raíz del polinomio característico, r_i, $p(r_i) = 0$, r_i^n es solución de la recurrencia.

Además,

$$t_n = \sum_{i=1}^{k} c_i r_i^n$$

Cuando todos los r_i son distintos, éstas son las únicas soluciones

Ejemplo Fibonacci

$$f_n = \begin{cases} n, & \text{si } n = 0 \text{ ó } n = 1 \\ f_{n-1} + f_{n-2} & \text{en otro caso} \end{cases}$$

$$f_n - f_{n-1} - f_{n-2} = 0$$

$$p(x) = x^2 - x - 1$$

$$r_1 = \frac{1+\sqrt{5}}{2} \quad y \quad r_2 = \frac{1-\sqrt{5}}{2}$$

Solución Fibonacci

Solución general:

$$f_n = c_1 r_1^n + c_2 r_2^n$$

$$\begin{aligned} c_1 &+ c_2 &= 0 \\ r_1 c_1 &+ r_2 c_2 &= 1 \end{aligned}$$

$$c_1 = \frac{1}{\sqrt{5}} \quad y \quad c_2 = -\frac{1}{\sqrt{5}}$$

$$f_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

Ejemplo

$$t_n = \begin{cases} 0, & n = 0 \\ 5, & n = 1 \\ 3t_{n-1} + 4t_{n-2}, & \text{en otro caso} \end{cases}$$

$$t_n - 3t_{n-1} - 4t_{n-2} = 0$$

$$x^2 - 3x - 4 = (x+1)(x-4)$$

A partir de las condiciones iniciales: $c_1 = -1$ y $c_2 = 1$

$$t_n = c_1(-1)^n + c_2 4^n$$

Ecuaciones lineales homogéneas.

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

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

- ¿Cuál es la solución para **t(n)**?

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

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

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

Ecuaciones lineales homogéneas.

- Las derivadas valen 0 en s_1, hasta la m-1-ésima.
- **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...
- Si **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 **s_k** de multiplicidad **m**, la solución será:

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

En resumen: Si hay raíces múltiples....

Si las raíces del polinomio característico NO son todas distintas. Sean r_i con multiplicidad m_i, i=1,...,l, las soluciones de p(x). Entonces

$$t_n = \sum_{i=1}^{l} \sum_{j=0}^{m_i - 1} c_{ij} n^j r_i^n$$

Ejemplo

$$t_n = \begin{cases} n, & \text{si } n = 0,1 \text{ ó } 2 \\ 5t_{n-1} - 8t_{n-2} + 4t_{n-3} & \text{en otro caso} \end{cases}$$

$$t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0$$

$$x^3 - 5x^2 + 8x - 4 = (x-1)(x-2)^2$$

$$t_n = c_1(1)^n + c_2 2^n + c_3 n 2^n$$

A partir de las condiciones iniciales:

c1 = -2, c2 = 2, c3 = -1/2

$$t_n = 2^{n+1} - n2^{n-1} - 2$$

Recurrencias no homogéneas

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = b^n p(n)$$

- b es una constante
- p(n) es un polinomio de n de grado d

- Ejemplo: $t_n - 2t_{n-1} = 3^n$

- El polinomio característico es:

$$p(x) = (a_0 x^k + a_1 x^{k-1} + \cdots + a_k)(x-b)^{d+1}$$

- Se procede igual que en el caso homogéneo, salvo que las condiciones iniciales se obtienen de la propia recurrencia

Ejemplo

$$t_n = \begin{cases} 0, & \text{si } n = 0 \\ 2t_{n-1} + 1 & \text{en otro caso} \end{cases}$$

$$t_n - 2t_{n-1} = 1 \qquad p(x) = (x-2)(x-1)$$

Solución general : $t_n = c_1 2^n + c_2$

$$\begin{array}{rcrcl} c_1 & + & c_2 & = & 0 \\ 2c_1 & + & c_2 & = & 1 \end{array} \Rightarrow \begin{array}{c} c_1 = 1 \\ c_2 = -1 \end{array}$$

$$t_n = 2^n - 1$$

Ejemplo

$$t_n = \begin{cases} 0, & \text{si } n = 0 \\ 2t_{n-1} + n & \text{en otro caso} \end{cases}$$

$$p(x) = (x-2)(x-1)^2$$

Solución general : $t_n = c_1 2^n + c_2 1^n + c_3 n 1^n$

$$\begin{array}{rcrcrcl} c_1 & + & c_2 & & & = & 0 \\ 2c_1 & + & c_2 & + & c_3 & = & 1 \\ 4c_1 & + & c_2 & + & 2c_3 & = & 4 \end{array} \quad \Rightarrow \quad \begin{array}{l} c_1 = 2 \\ c_2 = -2 \\ c_3 = -1 \end{array}$$

$$t_n = 2^{n+1} - n - 2$$

Cambio de variable

- **Calcular el orden de T(n) si n es potencia de 2, y**

$$T(n) = 4T(n/2) + n,\ n > 1$$

- Reemplazamos n por 2^k (de modo que $k = \lg n$) para obtener $T(2^k) = 4T(2^{k-1}) + 2^k$. Esto puede escribirse,

$$t_k = 4t_{k-1} + 2^k$$

- si $t_k = T(2^k) = T(n)$.
- La ecuación característica es $(x-4)(x-2) = 0$

  y entonces $t_k = c_1\, 4^k + c_2\, 2^k$.
- Poniendo n en lugar de k, tenemos $T(n) = c_1\, n^2 + c_2\, n$
- y T(n) es por tanto $O(n^2$ tq. n es una potencia de 2 y suponiendo que c_1 es positivo)

Cambio de variable

- Encontrar el orden de T(n) si n es una potencia de 2 y si

$$T(n) = 4T(n/2) + n^2 , n > 1$$

- Obtenemos sucesivamente

$$T(2^k) = 4T(2^{k-1}) + 4^k, \text{ y}$$

$$t_k = 4t_{k-1} + 4^k$$

- Ecuación característica: $(x-4)^2 = 0$, y asi

  $t_k = c_1 4^k + c_2 k4^k$, $\mathbf{T(n) = c_1 n^2 + c_2 n^2 \lg n}$

- y T(n) es $O(n^2 \log n$ tq. n es potencia de 2).

Cambio de variable

- Calcular el orden de T(n) si n es una potencia de 2

$$T(n) = 2T(n/2) + n\lg n,\ n > 1$$

- Obtenemos

$$T(2^k) = 2T(2^{k-1}) + k2^k$$

$$t_k = 2t_{k-1} + k2^k$$

- La ecuación característica es $(x-2)^3 = 0$, y asi,

$$t_k = c_1 2^k + c_2 k2^k + c_3 k^2 2^k$$

$$T(n) = c_1 n + c_2 n\lg n + c_3 n\lg^2 n$$

- Asi T(n) es $O(n\log^2 n$ tq. n es potencia de 2).

Cambio de variable

- Calcular el orden de T(n) si n es potencia de 2 y $T(n) = 3T(n/2) + cn$ (c es constante, $n \geq 1$).

- Obtenemos sucesivamente,

$$T(2^k) = 3T(2^{k-1}) + c2^k$$
$$t_k = 3t_{k-1} + c2^k$$

- Ecuación característica: $(x-3)(x-2) = 0$, y asi,

$$t_k = c_1 3^k + c_2 2^k$$
$$T(n) = c_1 3^{\lg n} + c_2 n$$

- y como $a^{\lg b} = b^{\lg a}$, $T(n) = c_1 n^{\lg 3} + c_2 n$

y finalmente T(n) es $O(n^{\lg 3}$ tq. n es potencia de 2).

Transformaciones del rango

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

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

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

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

- Tomando logaritmos (en base 2):

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

- Se hace una transformación de la imagen:

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

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

Transformaciones del rango

- 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:

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

  $= (2^{3n-2} \cdot 3^n)/n = 24^n/4n$

- Quitar la condición de que n sea potencia de 2.

Transformaciones del rango

- $T(n) = nT^2(n/2)$, $n > 1$, $T(1) = 6$ y n potencia de 2.

- Cambiamos la variable: $t_k = T(2^k)$, y asi

$$t_k = 2^k t^2_{k-1} , k > 0; t_0 = 6.$$

- Esta recurrencia no es lineal, y uno de los coeficientes no es constante.

- Para transformar el rango, creamos una nueva recurrencia tomando $V_k = \lg t_k$, lo que da,

$$V_k = k + 2 V_{k-1} , k > 0; V_0 = \lg 6.$$

- Ecuación característica: $(x-2)(x-1)^2 = 0$ y asi,

$$V_k = c_1 2^k + c_2 1^k + c_3 k1^k$$

Algo más sobre las 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.

Algo más sobre las 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 las condiciones iniciales para despejar c_1, c_2, c_3.

Algo más sobre las condiciones iniciales

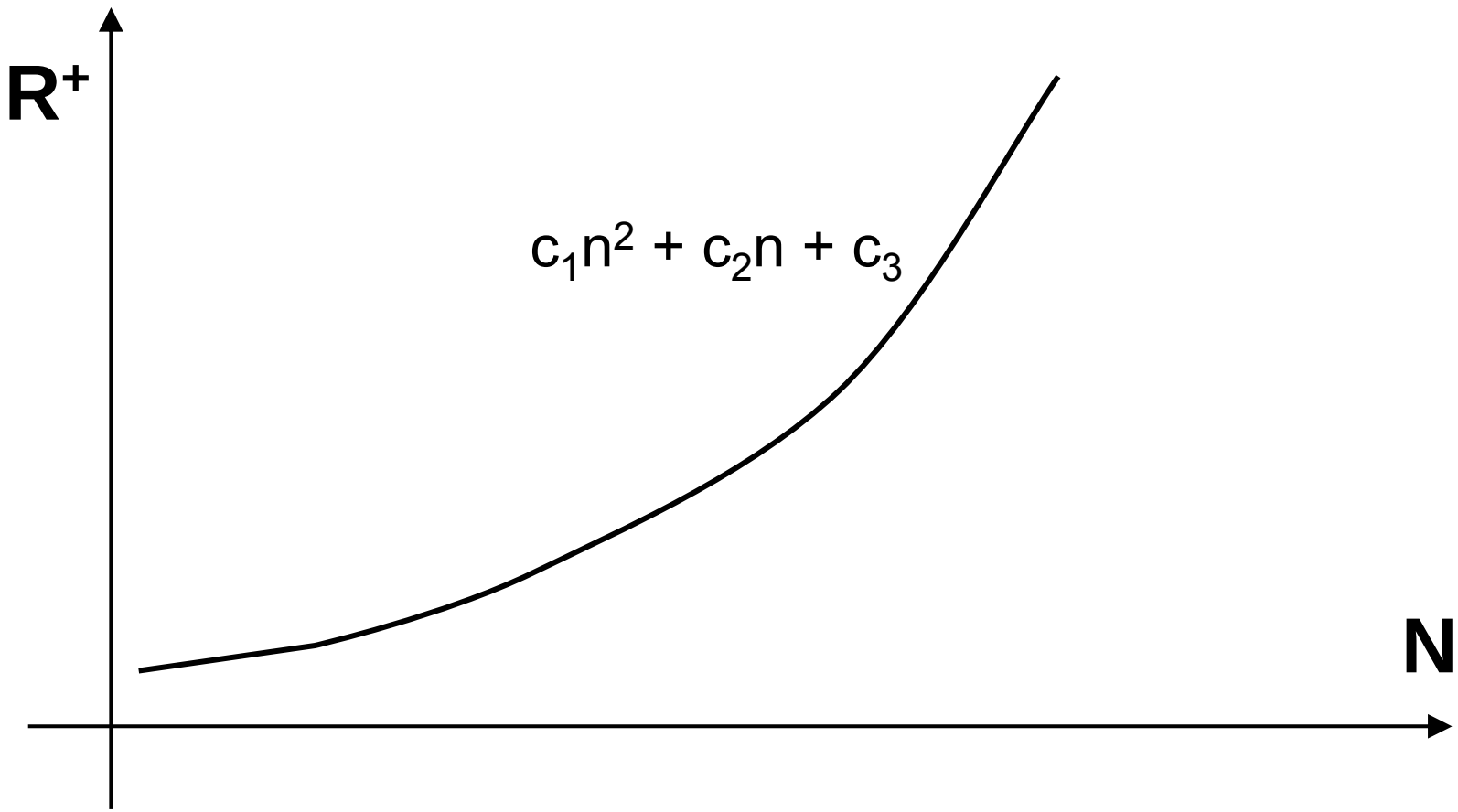
- El cálculo de constantes también se puede aplicar en el estudio experimental de algoritmos.

- **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.

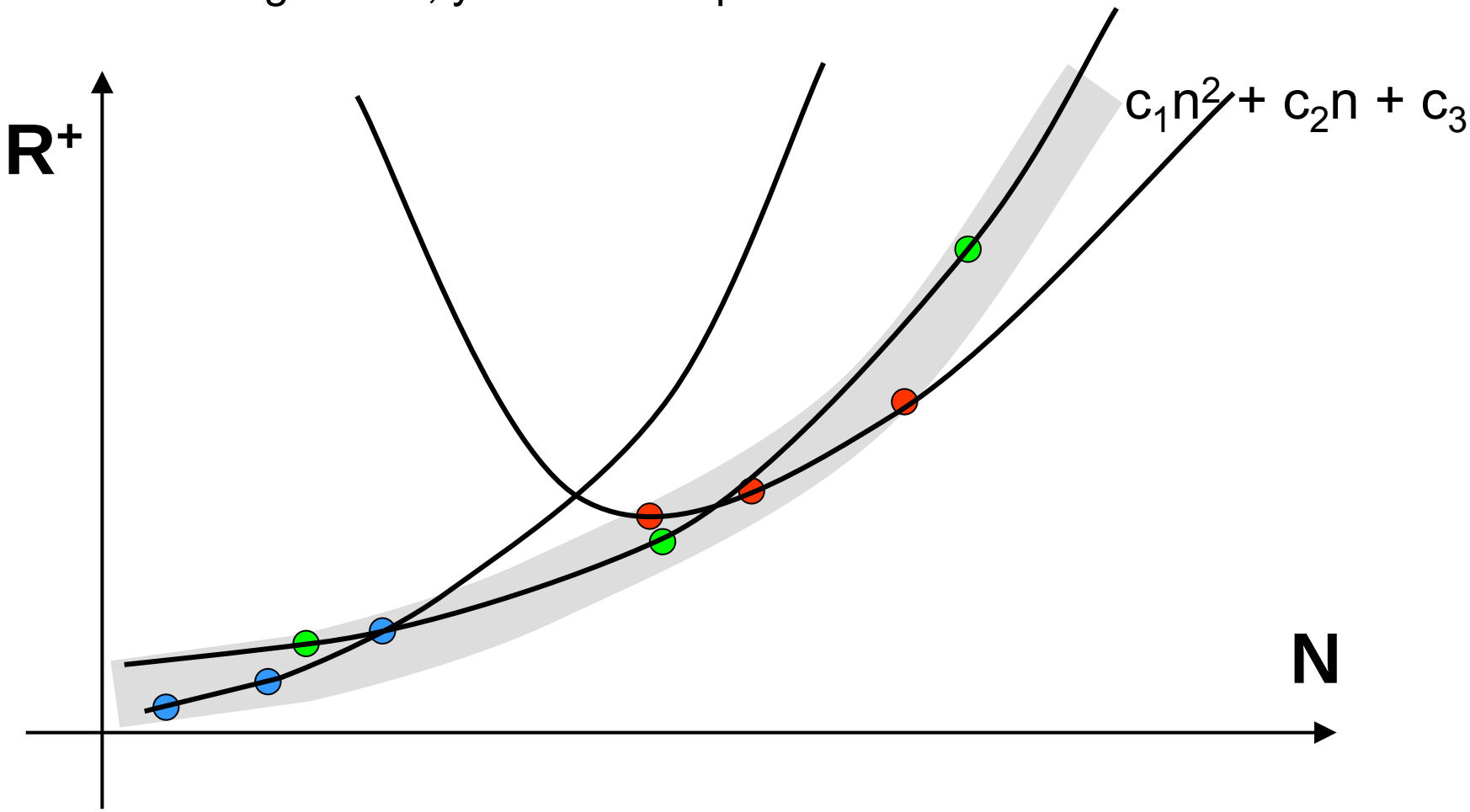
Algo más sobre las condiciones iniciales

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



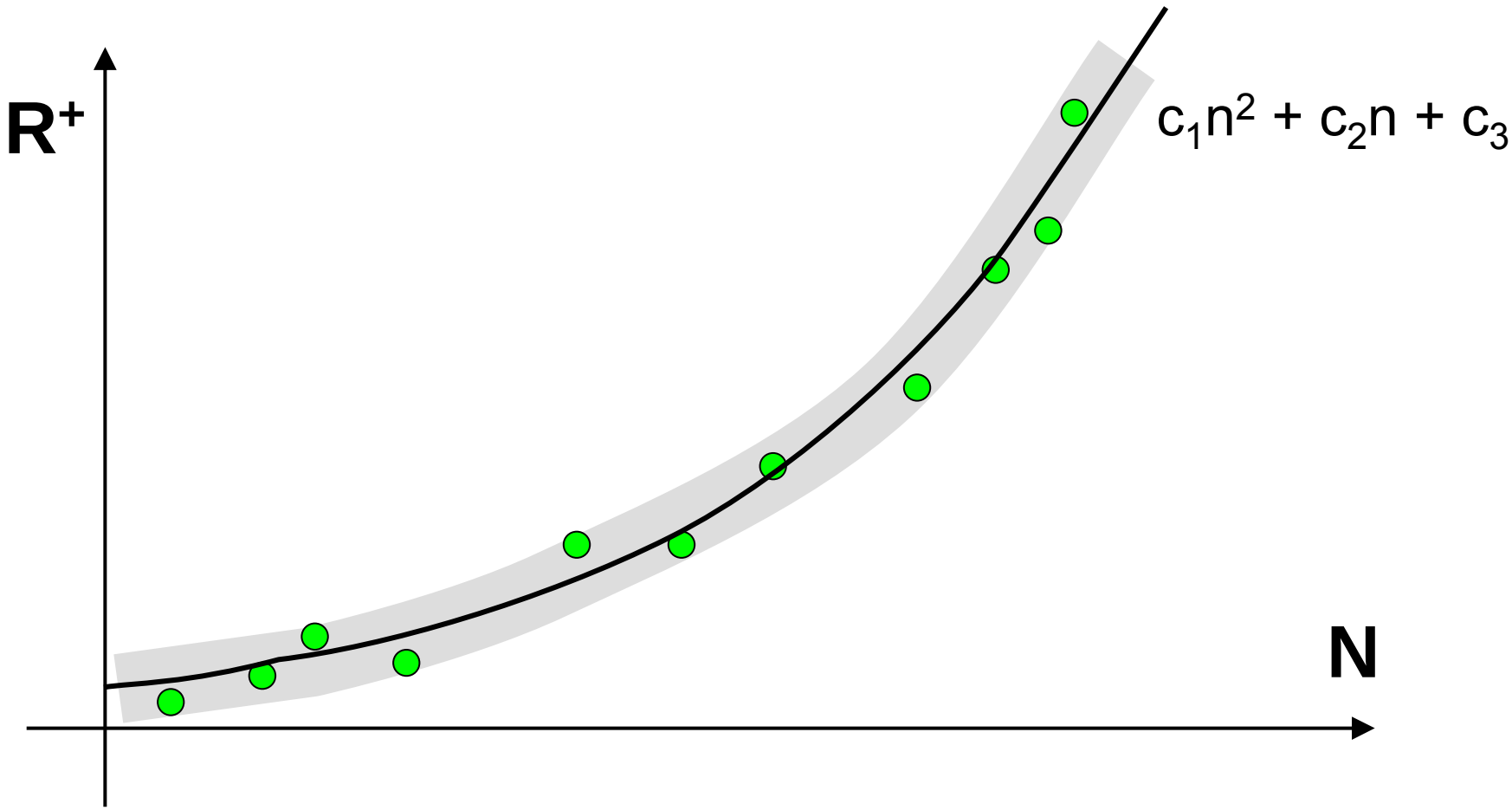
Algo más sobre las condiciones iniciales

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



Algo más sobre las condiciones iniciales

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



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: $O(t)$, $\Omega(t)$, $\Theta(t)$, $o(t)$.

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).

Algorítmica

Tema 1. Planteamiento General

Tema 2. La Eficiencia de los Algoritmos

Tema 3. Algoritmos "Divide y vencerás"

Tema 4. Algoritmos Voraces ("Greedy")

Tema 5. Algoritmos basados en Programación Dinámica

Tema 6. Algoritmos para la Exploración de Grafos ("Backtracking", "Branch and Bound")

Tema 7. Otras metodologías algorítmicas