Análisis de la eficiencia de los algoritmos

Yolanda Ortega Mallén

Dpto. de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

Objetivos

1. Obtener una medida del tiempo de ejecución de un algoritmo, que permita:
   - Comparar la eficiencia de distintos algoritmos para un mismo problema.
   - Identificar algoritmos con un tiempo de ejecución inaceptable para un tamaño de entrada.
2. Obtener una medida de la memoria requerida por un algoritmo.

Sumario

- Conceptos básicos.
- Medidas del comportamiento asintótico.
- Cálculo del coste en tiempo de ejecución.
- Coste en espacio de memoria.

- R. Peña. *Diseño de programas. Formalismo y abstracción*. Tercera edición. Prentice Hall, 2005. Capítulo 1.
- G. Brassard y P. Bratley. *Fundamentos de Algoritmia*, Prentice Hall, 1997. Capítulos 2, 3 y 4.
- N. Martí Oliet, C. Segura Díaz y J. A. Verdejo López. *Algoritmos correctos y eficientes: diseño razonado ilustrado con ejercicios*. Garceta Grupo Editorial, 2012. Capítulo 3.

Algoritmos correctos, pero ¿cuánto de costosos?

Recursos utilizados:

- espacio de memoria,
- tiempo de ejecución.

¿Por qué preocuparse? Creciente potencia de cálculo y mayor capacidad de almacenamiento de los computadores.

Ejemplo: Escoger entre n valores para sumar exactamente S

Requiere examinar 2^n posibilidades.

1. Supongamos que tardamos *1ns* (10^{-9}) en examinar cada caso:

n	10	20	30	40	50	60
2^n	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
$t(n)$	$1\ \mu s$	$1\ ms$	$1\ s$	$+15\ m$	$+10\ d$	$+30\ a$

2. Con una máquina que resolviera 10^{18} casos por segundo:

n	10	20	40	80	90
2^n	10^3	10^6	10^{12}	10^{24}	10^{27}
$t(n)$	$1\ fs$	$1\ ps$	$1\ \mu s$	$+10\ d$	$+30\ a$

3. Algoritmo solo examina n^3 casos + máquina resuelve 100 casos/sg:
   - En un día puede resolver una instancia de tamaño $n > 200$.
   - En un año puede resolver una instancia de tamaño cercano a 1500.

Un algoritmo delimita lo que es soluble en un tiempo/espacio razonable, independientemente de la implementación.

¿Cómo calcular el coste de un algoritmo?

Estrategia empírica (A posteriori) Programar y ejecutar en un computador sobre instancias de prueba.

Estrategia teórica (A priori) Determinar matemáticamente los recursos a utilizar para resolver cualquier instancia.

Factores de los que depende el tiempo de ejecución de un programa:

Tamaño de los datos de entrada: longitud del vector (N).

Contenido de los datos de entrada: diferencia entre T_{min} y T_{max}.

Código generado por el compilador y computador concretos: tiempos elementales $(t_a, t_c, t_i \ldots)$.

Buscamos una medida de la eficiencia de un algoritmo que sea independiente del computador y lenguaje concretos que vayamos a utilizar.

Principio de invarianza

Diferentes implementaciones de un mismo algoritmo diferirán en sus tiempos de ejecución a lo sumo en una constante multiplicativa positiva, para tamaños del problema suficientemente grandes.

Podemos ignorar las constantes multiplicativas;
y contar el número de operaciones elementales (de coste unitario).

Eficiencia en tiempo

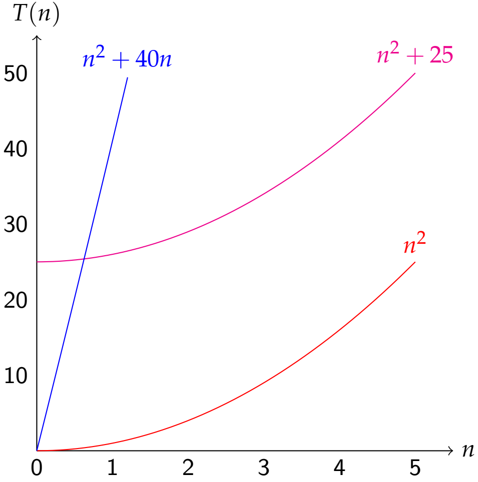
El tiempo de ejecución de un algoritmo puede variar mucho para diferentes datos de entrada de un mismo tamaño.

Caso peor: fijado un tamaño, coste para aquellas instancias en las que se emplea más tiempo de ejecución.

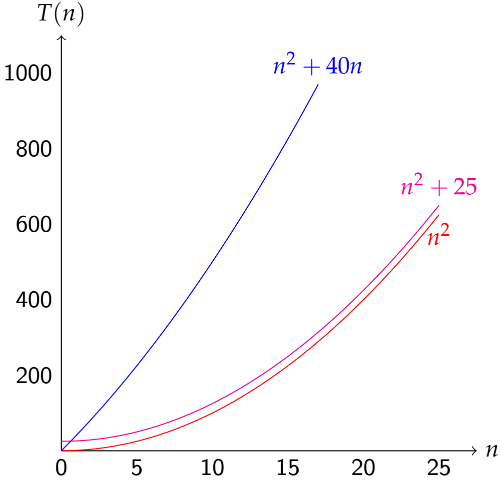
Caso promedio: fijado un tamaño, considerar el coste de cada posible instancia y la frecuencia con que se presenta cada caso.

Supongamos que el tiempo de proceso para una instancia de tamaño 1 es *1ms*.

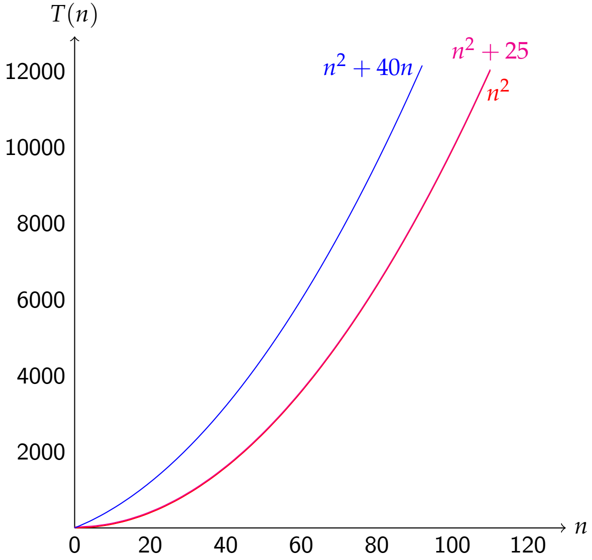
n	$\log_{10} n$	n	n^2	n^3	2^n
10	1 *ms*	10 *ms*	0,1 *s*	1 *s*	1,02 *s*
10^3	3 *ms*	1 *s*	16,67 *m*	11,57 *d*	$3{,}4 * 10^{291}$ *sig*
10^6	6 *ms*	1,67 *h*	31,71 *a*	317 097,9 *sig*	$3{,}1 * 10^{301020}$ *sig*


$T(n)$
$n^2 + 40n$
$n^2 + 25$
n^2
50
40
30
20
10
0
1
2
3
4
5
n

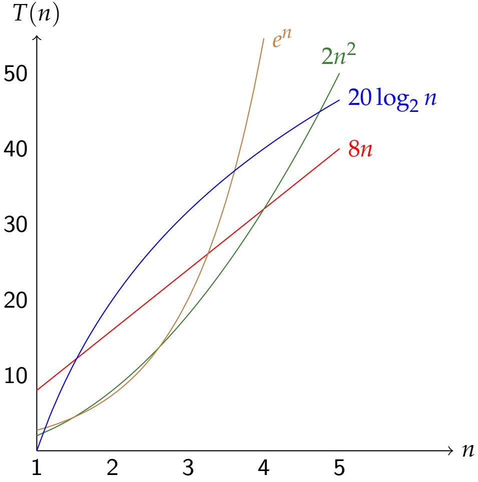
Comportamiento asintótico

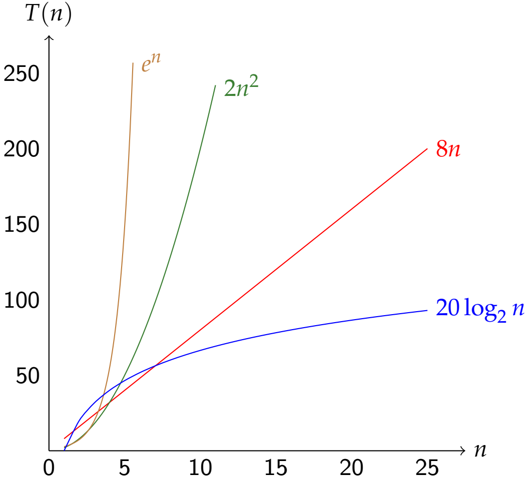


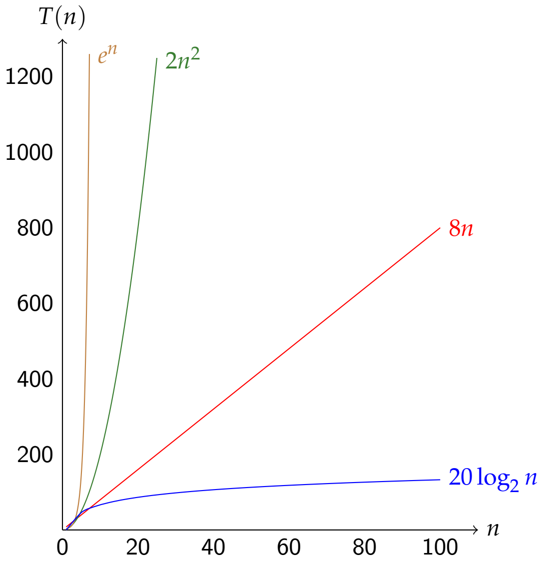
Comportamiento asintótico



Comportamiento asintótico




$T(n)$
250
200
150
100
50
0
5
10
15
20
25
n
e^n
$2n^2$
$8n$
$20 \log_2 n$


$T(n)$
e^n
$2n^2$
1200
1000
800
$8n$
600
400
200
$20\log_2 n$
0
20
40
60
80
100
n

Criterio asintótico para medir la eficiencia de los algoritmos:

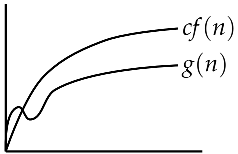
1. Función de coste que solo depende del tamaño de los datos de entrada.
   ¿Qué se entiende por tamaño de un problema?
2. Las constantes multiplicativas o aditivas no se tienen en cuenta.
   Ejemplo: $f(n) = n^2$ y $g(n) = 3n^2 + 27$ se consideran equivalentes.
3. Los costes para tamaños pequeños se consideran irrelevantes;
   la comparación entre funciones de coste se hace para tamaños suficientemente grandes.

Definición 1.1

Sea $f : \mathbb{N} \longrightarrow \mathbb{R}^+ \cup \{0\}$. *El conjunto de las funciones del* **orden de** $f(n)$, *denotado por* $O(f(n))$, *se define como:*

$$O(f(n)) = \{g : \mathbb{N} \longrightarrow \mathbb{R}^+ \cup \{0\} \mid \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} \,.\, \forall n \geq n_0 \,.\, g(n) \leq cf(n)\}$$

Una función g **es del orden de** $f(n)$ *si* $g \in O(f(n))$.



Si $T(n) \in O(f(n))$ tenemos una cota superior al tiempo de ejecución de un algoritmo.

- Se admiten funciones negativas o indefinidas para un número finito de valores de n si eligiendo n_0 suficientemente grande, satisface la definición.
- Las unidades de medida del coste en tiempo (horas, segundos, milisegundos, etc.), o en memoria (octetos, palabras, celdas de longitud fija, etc.) no son relevantes en la complejidad asintótica.
- Implementaciones del mismo algoritmo que difieran en el lenguaje, el compilador, y/o/ el computador, son del mismo orden.
- Se puede aplicar tanto a un análisis en el caso peor, como a un análisis en el caso promedio.
  El orden puede no coincidir.

Representante de $O(f(n))$: la función $f(n)$ más sencilla posible.

1. $O(1)$: **constantes**
2. $O(\log n)$: **logarítmico** (no depende de la base)
3. $O(n)$: **lineal**
4. $O(n^2)$: **cuadrático**
5. $O(n^k)$: **polinomial**
6. $O(2^n)$: **exponencial**
7. $O(n!)$: **factorial**

Supongamos que el tiempo de proceso para una instancia de tamaño 1 es $1\mu s$.

n	$\log n$	n	$n \log n$	n^2	n^3	2^n	$n!$
10	$3,32 \cdot 10^{-6}$	10^{-5}	$3,32 \cdot 10^{-5}$	10^{-4}	$0,001$	$0,001024$	$3,6288$
50	$5,64 \cdot 10^{-6}$	$5 \cdot 10^{-5}$	$2,82 \cdot 10^{-4}$	0,0025	$0,125$	intratable	intratable
100	$6,64 \cdot 10^{-6}$	10^{-4}	$6,64 \cdot 10^{-4}$	0,01	1	intratable	intratable
10^3	10^{-5}	$0,001$	$0,01$	1	10^3	intratable	intratable
10^4	$1,33 \cdot 10^{-5}$	0,01	$0,133$	100	10^6	intratable	intratable
10^5	$1,66 \cdot 10^{-5}$	0,1	$1,66$	10^4	intratable	intratable	intratable
10^6	$2 \cdot 10^{-5}$	1	$19,93$	10^6	intratable	intratable	intratable

Duplicar el tamaño del problema

$T(n)$	$n = 100$	$n = 200$
$O(\log n)$	1 h	1,15 h
$O(n)$	1 h.	2 h
$O(n \log n)$	1 h	2,30 h
$O(n^2)$	1 h	4 h
$O(n^3)$	1 h	8 h
$O(2^n)$	1 h	$1,27 \times 10^{30}$ h

Duplicar el tiempo disponible

$T(n)$	t=1 h	t=2 h
$O(\log n)$	$n = 100$	$n = 10{,}000$
$O(n)$	$n = 100$	$n = 200$
$O(n \log n)$	$n = 100$	$n = 178$
$O(n^2)$	$n = 100$	$n = 141$
$O(n^3)$	$n = 100$	$n = 126$
$O(2^n)$	$n = 100$	$n = 101$

Jerarquía de órdenes de complejidad

$$\underbrace{\underbrace{O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2)}_{\text{razonables en la práctica}} \subset \ldots \subset O(n^k)}_{\text{tratables}} \subset$$

$$\underbrace{\ldots \subset O(2^n) \subset O(n!)}_{\text{intratables}}$$

Reflexivo $f(n) \in O(f(n))$

Transitivo $f(n) \in O(g(n)) \ \wedge \ g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$

Antisimétrico $f(n) \in O(g(n)) \ \wedge \ g(n) \in O(f(n)) \Leftrightarrow O(f(n)) = O(g(n))$

Estricto $f(n) \in O(g(n)) \ \wedge \ g(n) \notin O(f(n)) \Leftrightarrow O(f(n)) \subset O(g(n))$

Escalable $O(a \cdot f(n)) = O(f(n))$ con $a \in \mathbb{R}^+$

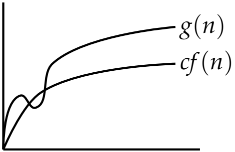
Relación de orden parcial sobre las funciones.

Determinar la mayor función que sea una cota inferior del tiempo de ejecución.

Definición 1.2

Sea $f : \mathbb{N} \longrightarrow \mathbb{R}^+ \cup \{0\}$*. El conjunto* **omega de** $f(n)$*, denotado por* $\Omega(f(n))$*, se define como:*

$$\Omega(f(n)) = \{g : \mathbb{N} \longrightarrow \mathbb{R}^+ \cup \{0\} \mid \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} \,.\, \forall n \geq n_0 \,.\, g(n) \geq cf(n)\}$$

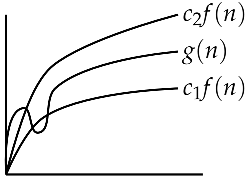


- Principio de dualidad: $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$
- NO confundir la medida $O(f(n))$ como aplicable al caso peor y la medida $\Omega(f(n))$ como aplicable al caso mejor.
- Supongamos para un análisis en el caso peor que
  $T(n) \in O(f(n)) \cap \Omega(g(n))$,
  ninguna instancia de tamaño n puede tener coste superior a $c.f(n)$,
  pero puede haber instancias de tamaño n con coste inferior a $c'.g(n)$.

Definición 1.3

Sea $f : \mathbb{N} \longrightarrow \mathbb{R}^+ \cup \{0\}$*. El conjunto de funciones* **del orden exacto de** $f(n)$*, denotado por* $\Theta(f(n))$ *se define como:*

$$\Theta(f(n)) = \{g : \mathbb{N} \longrightarrow \mathbb{R}^+ \cup \{0\} \mid \exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N} \,.\, \forall n \geq n_0 \,.\, c_1 f(n) \leq g(n) \leq c_2 f(n)\}$$



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

- Simetría de Θ: $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$
- $O(f(n)) = O(g(n)) \Leftrightarrow \Theta(f(n)) = \Theta(g(n))$
  Θ no es más potente que O para comparar el crecimiento de las funciones.
- Ω y Θ son reflexivas, transitivas y escalables.

Limitaciones del criterio asintótico

- Las constantes multiplicativas *escondidas* pueden hacer impracticable el algoritmo.
- Tamaño de los datos ¿grande o pequeño?
- Frecuencia de utilización: eficiencia vs. simplicidad.
- Menos tiempo de ejecución a costa de mayor coste en espacio. ¿Aceptable? ¿Uso de memoria externa?

Reglas de cálculo

Regla de la suma $O(f(n) + g(n)) = O(\max(f(n), g(n)))$

Regla del producto Si $g_1(n) \in O(f_1(n))$ y $g_2(n) \in O(f_2(n))$, entonces $g_1(n) \cdot g_2(n) \in O(f_1(n) \cdot f_2(n))$.

Regla del límite

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = m \in \mathbb{R}^+ \Rightarrow f(n) \in \Theta(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) \in O(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty \Rightarrow f(n) \in \Omega(g(n))$$

Análisis de las estructuras de control (caso peor)

Análisis desde dentro hacia afuera.

Instrucción simple Asignación, E/S, expresión aritmética/lógica, acceso a array/registro, . . . : $O(1)$.

Composición secuencial $P_1\,;\,P_2$,

con $T_i(n) \in O(f_i(n)), i = 1, 2$:

$$T(n) = T_1(n) + T_2(n) \in O(f_1(n) + f_2(n)) = O(\text{máx}(f_1(n), f_2(n)))$$

Instrucción condicional **si** b **entonces** P_1 **si no** P_2 **fsi**,

con $T_B(n) \in O(f_B(n))$ y $T_i(n) \in O(f_i(n)), i = 1, 2$:

$$T(n) = T_B(n) + \text{máx}(T_1(n), T_2(n)) \in O(\text{máx}(f_B(n), f_1(n), f_2(n)))$$

Instrucción iterativa **mientras** b **hacer** P **fmientras**,

con $T_B(n) \in O(f_B(n))$ y $T_P(n) \in O(f_P(n))$:

Una vuelta: $T_B(n) + T_P(n) \in O(\text{máx}(f_B(n), f_P(n))) = O(f_V(n))$

¿Bucle completo? Número de iteraciones: $O(f_{it}(n))$.

$$T(n) \in O(f_V(n) \cdot f_{it}(n))$$

Instrucción crítica

Instrucción elemental que se ejecuta al menos con tanta frecuencia como las demás.

Definir $f_c(n)$ que calcule el número de veces que se ejecuta la función crítica.

Ejemplos

Producto de matrices cuadradas.

```
fun producto(A[1..N,1..N], B[1..N,1..N] de ent)
    dev C[1..N,1..N] de ent
var i,j,k : nat, s : ent
   para i = 1 hasta N hacer
      para j = 1 hasta N hacer
         s := 0 ;
         para k = 1 hasta N hacer
            s := s + A[i,k] * B[k,j]
         fpara ;
         C[i,j] := s
      fpara
   fpara
ffun
```

$$T(N) \in \Theta(N^3)$$

Ejemplos

Ordenación por selección.

proc `ord-selección`$(V[1..N]$ **de** *ent*$)$
  para $i = 1$ **hasta** $N-1$ **hacer**
    $pmin := i$;
    para $j = i+1$ **hasta** N **hacer**
      si $V[j] < V[pmin]$ **entonces** $pmin := j$ **fsi**
    fpara ;
    `intercambiar`$(V[i], V[pmin])$
  fpara
fproc

$$T(N) \in \Theta(\sum_{i=1}^{N-1}(N-i)) = \Theta(N^2)$$

Ejemplos

Determinar si una matriz cuadrada es simétrica.

fun `simétrica?`$(V[1..N, 1..N]$ **de** *ent*$)$ **dev** b : *bool*
var i, j : *nat*
  $b :=$ `cierto` ; $i := 1$;
  mientras $i \leq N \wedge b$ **hacer**
    $j := i + 1$;
    mientras $j \leq N \wedge b$ **hacer**
      $b := (V[i,j] = V[j,i])$;
      $j := j + 1$
    fmientras ;
    $i := i + 1$
  fmientras
ffun

$$T_{\text{mín}}(N) \in \Theta(1) \qquad T_{\text{máx}}(N) \in \Theta(N^2)$$

Ejemplo

proc `ord-inserción`$(V[1..N]$ **de** *ent*$)$
  para $i = 2$ **hasta** N **hacer**
    $elem := V[i]$;
    $j := i - 1$;
    mientras $j > 0 \wedge_c elem < V[j]$ **hacer**
      $V[j+1] := V[j]$;
      $j := j - 1$
    fmientras ;
    $V[j+1] := elem$
  fpara
fproc

caso peor: $T_{\text{máx}}(N) = \sum_{i=2}^{N} i = \frac{(N+2)(N-1)}{2} \in \Theta(N^2)$

caso mejor: $T_{\text{mín}}(N) = \sum_{i=2}^{N} 1 = N - 1 \in \Theta(N)$

caso promedio: $\Theta(N^2)$

Recurrencias: funciones de coste recursivas.

fun `factorial`$(n : nat)$ **dev** $f : nat$
  casos
    $n = 0 \ \rightarrow \ f := 1$
    □ $n > 0 \ \rightarrow$
      $f := n * \texttt{factorial}(n-1)$
  fcasos
ffun

Recurrencia

$$T(n) = \begin{cases} k_1 & \text{si } n = 0 \\ T(n-1) + k_2 & \text{si } n > 0 \end{cases}$$

Para obtener el orden de complejidad de $T(n)$:

Despliegue obtener fórmula explícita de $T(n)$.

Teoremas disminución del tamaño del problema por sustracción o por división.

El objetivo es conseguir una fórmula explícita (en función de n) de $T(n)$.

Despliegue	Sustituir T por la parte derecha de la ecuación. Repetir hasta encontrar una fórmula que dependa del número de despliegues (o llamadas recursivas) i.
Postulado	Obtener el valor de i que hace desaparecer T (caso básico). En la fórmula, sustituir i por ese valor para obtener la fórmula explícita T^*, que solo depende de n.
Comprobación	Demostrar por inducción que $T = T^*$.

Recurrencia

$$T(n) = \begin{cases} k_1 & \text{si } n = 0 \\ T(n-1) + k_2 & \text{si } n > 0 \end{cases}$$

$$\begin{array}{rcl} T(n) & \stackrel{1}{=} & T(n-1) + k_2 \\ & \stackrel{2}{=} & T(n-2) + k_2 + k_2 \quad = \quad T(n-2) + 2 \cdot k_2 \\ & \stackrel{3}{=} & T(n-3) + k_2 + 2 \cdot k_2 \quad = \quad T(n-3) + 3 \cdot k_2 \\ & \vdots & \\ & \stackrel{i}{=} & T(n-i) + i \cdot k_2 \\ & \vdots & \\ & \stackrel{n}{=} & T(0) + n \cdot k_2 \quad = \quad n \cdot k_2 + k_1 = T^*(n) \in \Theta(n) \end{array}$$

$\forall n \geq 0 \,.\, T(n) = T^*(n)$

Caso base $T(0) = k_1 = T^*(0)$

Paso inductivo H.I.: $T(n) = T^*(n)$

$$T(n+1) = T(n) + k_2 \stackrel{h.i.}{=} n \cdot k_2 + k_1 + k_2 = (n+1) \cdot k_2 + k_1 = T^*(n+1)$$

Ejemplos

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 3T(n-1) + 2 & \text{si } n \geq 2 \end{cases}$$

$$\begin{array}{rcl} T(n) & \stackrel{1}{=} & 3T(n-1) + 2 \\ & \stackrel{2}{=} & 3(3T(n-2) + 2) + 2 = 3^2 T(n-2) + 3 \cdot 2 + 2 \\ & \stackrel{3}{=} & 3^2(3T(n-3) + 2) + 3 \cdot 2 + 2 = 3^3 T(n-3) + 3^2 \cdot 2 + 3 \cdot 2 + 2 \\ & \vdots & \\ & \stackrel{i}{=} & 3^i T(n-i) + \sum_{j=0}^{i-1} 3^j \cdot 2 \\ & \vdots & \\ & \stackrel{n-1}{=} & 3^{n-1} T(1) + \sum_{j=0}^{n-2} 3^j \cdot 2 = 3^{n-1} + 2 \cdot \dfrac{3 \cdot 3^{n-2} - 3^0}{3-1} \\ & = & 2 \cdot 3^{n-1} - 1 \in \Theta(3^n) \end{array}$$

Ejemplos

$$T(n) = \begin{cases} 4 & \text{si } n = 1 \\ 2T(n/2) + 3n + 2 & \text{si } n \geq 2, \text{ potencia de 2} \end{cases}$$

Haciendo sucesivos desplegados obtenemos

$$\begin{aligned}
T(n) &\stackrel{1}{=} 2T(\frac{n}{2}) + 3n + 2 \\
&\stackrel{2}{=} 2(2T(\frac{n/2}{2}) + 3\frac{n}{2} + 2) + 3n + 2 = 2^2T(\frac{n}{2^2}) + 2 \cdot 3n + 2^2 + 2 \\
&\stackrel{3}{=} 2^2(2T(\frac{n/2^2}{2}) + 3\frac{n}{2^2} + 2) + 2 \cdot 3n + 2^2 + 2 \\
&= 2^3T(\frac{n}{2^3}) + 3 \cdot 3n + 2^3 + 2^2 + 2 \\
&\stackrel{4}{=} 2^3(2T(\frac{n/2^3}{2}) + 3\frac{n}{2^3} + 2) + 3 \cdot 3n + 2^3 + 2^2 + 2 \\
&= 2^4T(\frac{n}{2^4}) + 4 \cdot 3n + 2^4 + 2^3 + 2^2 + 2^1 \\
&\vdots
\end{aligned}$$

Ejemplos

$$\begin{aligned} &\stackrel{i}{=} \quad 2^i T(\frac{n}{2^i}) + i \cdot 3n + \sum_{j=1}^{i} 2^j \\ &= \quad 2^i T(\frac{n}{2^i}) + i \cdot 3n + 2^{i+1} - 2. \end{aligned}$$

Al caso básico se llega cuando $\frac{n}{2^i} = 1$, es decir, cuando $i = \log n$.

Por tanto para n potencia de 2,

$$\begin{aligned} T(n) &\stackrel{\log n}{=} 2^{\log n} T(1) + 3n \log n + 2^{1+\log n} - 2 \\ &= 4n + 3n \log n + 2n - 2 \\ &= 3n \log n + 6n - 2 \\ &\in \Theta(n \log n). \end{aligned}$$

- Caso directo: coste constante
- Preparación de llamadas y combinación de resultados: coste polinómico

Teorema de la resta: Descomposición restando una cantidad constante

$$T(n) = \begin{cases} k_0 & \text{si } 0 \leq n < b \\ a \cdot T(n-b) + k_1 \cdot n^k & \text{si } n \geq b \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n^{k+1}) & \text{si } a = 1 \\ \Theta(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

Teorema de la división: Descomposición dividiendo por una cantidad constante $b \geq 2$

$$T(n) = \begin{cases} k_0 & \text{si } 0 \leq n < b \\ a \cdot T(\frac{n}{b}) + k_1 \cdot n^k & \text{si } n \geq b \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < b^k \\ \Theta(n^k \log n) & \text{si } a = b^k \\ \Theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

El coste es menor cuanto más pequeñas son a y k y más grande es b.

Ejemplo: búsqueda binaria

```
fun búsqueda-binaria(V[1..N] de ent, e : ent, c, f : nat) dev ⟨b : bool, p : nat⟩
    si c > f entonces ⟨b, p⟩ := ⟨falso, c⟩
    si no
        m := (c + f) div 2 ;
        casos
                e < V[m] → ⟨b, p⟩ := búsqueda-binaria(V, e, c, m − 1)
              □ e = V[m] → ⟨b, p⟩ := ⟨cierto, m⟩
              □ e > V[m] → ⟨b, p⟩ := búsqueda-binaria(V, e, m + 1, f)
        fcasos
    fsi
ffun
```

Tamaño $n = f - c + 1$.

Recurrencia

$$T(n) = \begin{cases} k_1 & \text{si } n = 0 \\ T(n \operatorname{div} 2) + k_2 & \text{si } n > 0 \end{cases}$$

$$T(n) \in \Theta(\log n)$$

Ejemplo (2)

$\{\ \texttt{cierto}\ \}$
fun $\texttt{fibonacci}(n : \textit{nat})$ **dev** $f : \textit{nat}$
  casos
    $n = 0 \ \rightarrow\ f := 0$
  □ $n = 1 \ \rightarrow\ f := 1$
  □ $n \geq 2 \ \rightarrow\ f := \texttt{fibonacci}(n-1) + \texttt{fibonacci}(n-2)$
  fcasos
ffun
$\{\ f = \mathit{fib}_n\ \}$

$$T(n) = \begin{cases} k_1 & n \leq 1 \\ T(n-1) + T(n-2) + k_2 & n > 1 \end{cases}$$

$T(n)$ es monótona, es decir $T(n) \geq T(n')$ si $n \geq n'$

Ejemplo (2)

Veamos que $T(n) \in O(2^n) \cap \Omega(2^{n \text{ div } 2})$:

- $T(n) \in O(2^n)$. Si $n \geq 2$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + k_2 \\ &\leq 2T(n-1) + k_2 \leq 2^2 T(n-2) + 2k_2 + k_2 \leq \ldots \\ &\leq 2^i T(n-i) + k_2 \textstyle\sum_{j=0}^{i-1} 2^j \end{aligned}$$

  Cuando $n - i = 1$ entonces $i = n - 1$. Luego $T(n) \leq k_1 2^{n-1} + k_2 2^{n-1}$ y por tanto $T(n) \in O(2^n)$.

- $T(n) \in \Omega(2^n)$. Si $n \geq 2$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + k_2 \\ &\geq 2T(n-2) \geq 2^2 T(n-4) \geq \ldots \geq 2^i T(n-2i) \end{aligned}$$

  Si $n - 2i = 0$ (n es par) entonces $i = n/2$. Si $n - 2i = 1$ (n es impar) entonces $i = (n-1)/2$.
  En cualquier caso $T(n) \geq k_1 2^{n \text{ div } 2}$, luego $T(n) \in \Omega(2^{n \text{ div } 2})$.

Recurrencias con historia

$$T(n) = \begin{cases} 1 & n = 1 \\ \sum_{i=1}^{n-1} T(i) + n^2 & n \geq 2. \end{cases}$$

Buscamos primero una recurrencia equivalente en cuyo caso recursivo solamente aparezca el valor inmediatamente anterior $T(n-1)$.

Restamos $T(n-1)$ de $T(n)$. Para $n \geq 3$,

$$\begin{aligned} T(n) - T(n-1) &= \sum_{i=1}^{n-1} T(i) + n^2 - (\sum_{i=1}^{n-2} T(i) + (n-1)^2) \\ &= \sum_{i=1}^{n-2} T(i) + T(n-1) + n^2 - \sum_{i=1}^{n-2} T(i) - (n^2 - 2n + 1) \\ &= T(n-1) + 2n - 1. \end{aligned}$$

Despejando tenemos

$$T(n) = 2T(n-1) + 2n - 1.$$

Recurrencias con historia

Si $n = 2$, la definición original de la recurrencia $T(n)$ da

$$T(2) = \sum_{i=1}^{2-1} T(i) + 2^2 = T(1) + 4 = 1 + 4 = 5,$$

y también

$$2T(n-1) + 2n - 1 = 2T(1) + 2 \cdot 2 - 1 = 2 + 4 - 1 = 5,$$

por lo que la fórmula anterior vale para todo $n \geq 2$.

Aplicando desplegado

$$\begin{array}{rcl}
T(n) & \stackrel{1}{=} & 2T(n-1) + 2n - 1 \\
& \stackrel{2}{=} & 2^2T(n-2) + 2^2(n-1) - 2 + 2n - 1 \\
& \stackrel{3}{=} & 2^3T(n-3) + 2^3(n-2) - 2^2 + 2^2(n-1) - 2 + 2n - 1 \\
& \stackrel{i}{=} & 2^iT(n-i) + \sum_{j=0}^{i-1} 2^{j+1}(n-j) - \sum_{j=0}^{i-1} 2^j \\
& = & 2^iT(n-i) + (2n-1)\sum_{j=0}^{i-1} 2^j - 2\sum_{j=0}^{i-1} j2^j.
\end{array}$$

Recurrencias con historia

Al caso básico se llega cuando $n - i = 1$; entonces $i = n - 1$ y

$$T(n) \stackrel{n=1}{=} 2^{n-1} + (2n-1)\sum_{j=0}^{n-2} 2^j - 2\sum_{j=0}^{n-2} j2^j.$$

Para simplificar las sumas recordamos las igualdades

$$\begin{aligned}\sum_{j=0}^{n-2} 2^j &= 2^{n-1} - 1\\ \sum_{j=0}^{n-2} j2^j &= (n-3)2^{n-1} + 2.\end{aligned}$$

Sustituyendo en la expresión anterior queda

$$\begin{aligned}T(n) &= 2^{n-1} + (2n-1)(2^{n-1} - 1) - 2((n-3)2^{n-1} + 2)\\ &= (1 + 2n - 1 - 2n + 6)2^{n-1} - (2n-1) - 4\\ &= 3 \cdot 2^n - 2n - 3\\ &\in \Theta(2^n).\end{aligned}$$

Coste en espacio de memoria

- Considerar todas las variables utilizadas en el algoritmo y multiplicar cada una por el número de bytes necesarios para su almacenamiento.
- Para un primer análisis emplear medidas de tipo asintótico, tener en cuenta únicamente las variables estructuradas y considerar que todos los elementos en todas las estructuras ocupan el mismo espacio.
- La llamada a un subprograma tiene asociado un coste en espacio: tabla de activación.

  Parámetro por valor Espacio igual a su tamaño;
  Parámetro por referencia Espacio unitario.
- En algoritmos recursivos el coste en espacio depende de la profundidad de la recursión. NO del número total de llamadas.
  Solo importa aquella llamada que provoque la mayor profundidad.