Introducción a la complejidad de algoritmos iterativos

October 4, 2010

Consideremos un programa que ordena una lista de n elementos. Supongamos que el tiempo que tarda viene dada por la función:

$$T(n) = 10^{-6} * 2^n \text{ seg.}$$

Consideremos diversas ordenaciones de listas de varios tamaños:

Tamaño	Tiempo
n=5	0.000032 seg
n=10	0.001024 seg
n=11	0.002048 seg
n=20	1.048576 seg
n=30	1073.74 seg
n=40	305 horas
n=50	35 años

Comentarios:

- Para $n = 30$ o $n = 40$ el procedimiento es inservible.
- Con un computador el doble de rápido solo podríamos ampliar el rango de valores de n para los que podemos aplicar el procedimiento en una o dos unidades.
- Con una lista con un elemento más tardamos el doble $\longrightarrow$ Causa de la intratabilidad.
- Es más rentable invertir en algoritmia que en hardware.
- Los algoritmos no solo son importantes por su *correción* sino por su *eficiencia* (tiempo de ejecución y memoria utilizada).

Recursos necesarios para la ejecución de un algoritmo:

- Espacio de almacenamiento (memoria) $\rightarrow$ Relevante en algoritmos recursivos.
- Tiempo de ejecución $\rightarrow$ Existen 2 posibles enfoques:
  - Empíricos: probar y medir. Características:
    - Exactitud de la medida.
    - Irrelevancia de la medida por depender de la computadora, la implementación del algoritmo y del compilador.
  - Analítico: obtener una aproximación matemática a partir de la estructura de un algoritmo. Características:
    - Solamente muestra un comportamiento general.
    - Poseemos una expresión analítica de la función temporal (una "fórmula").

Principio de invariancia: las implementaciones de un mismo algoritmo solo difieren en una constante multiplicativa:

$$Algoritmo \begin{cases} Implemen1 \rightarrow t_1(n) \\ Implemen2 \rightarrow t_2(n) \end{cases}$$

$$\exists c_0, c_1 \in \mathbb{R}^+, n_0 \in \mathbb{N} \mid t_1(n) \leq c_0 t_2(n) t_2(n) \leq c_1 t_1(n) \forall n > n_0$$

$\Rightarrow$ Ambas funciones tienen la misma "forma" $\leftrightarrow$ "Definición intuitiva" de complejidad!!.

- Objetivo: describir cómo dependen los recursos con respecto al "tamaño" de los datos para cada algoritmo ($\rightarrow$ No para cada programa!).
- El tamaño suele ser el tamaño de alguna estructura (por ejemplo, una lista) o el valor de una variable.
- Describiremos la eficiencia temporal mediante *funciones de coste*.
- De las funciones de coste solamente nos interesa su "forma" y no los valores concretos que pueda dar $\rightarrow$ Podemos considerar la "forma" de la función temporal para valores grandes como una definición intuitiva de "complejidad".

Objetivo: Encontrar conceptos que permitan hablar de los valores de la funciones temporales para grandes valores de n.
Sea $f : \mathbb{N} \rightarrow \mathbb{R}$ una función de coste

- $O(f) \equiv \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N}\ \forall n \geq n_0\ g(n) \leq cf(n)\}$. Son las funciones que crecen más lentamente que f.
- $\Omega(f) \equiv \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N}\ \forall n \geq n_0\ g(n) \geq cf(n)\}$. Son las funciones que crecen más rápidamente que f.
- $\Theta(f) \equiv O(f) \cap \Omega(f)$. Funciones que crecen igual de rápido que f.
- $\Theta(f) \equiv \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c, d \in \mathbb{R}, n_0 \in \mathbb{N}\ \forall n \geq n_0\ df(n) \leq g(n) \leq cf(n)\}$.

Teorema

Teorema del límite. *Sean dos funciones, f y g, y sea $\lim_{n\to\infty} \frac{f(n)}{g(n)} = k$. Entonces:*

- *Si $k = 0$ entonces f crece más lentamente que $g \Rightarrow f \in O(g)$.*
- *Si $k \in \mathbb{R}^+$ entonces f crece igual de rápido que g $\Rightarrow f \in \Theta(g)$.*
- *Si $k = \infty$ entonces f crece más rápidamente que g $\Rightarrow g \in O(f)$.*

Una herramienta que permite estudiar la pertenencia de una función a un orden de complejidad:

Teorema

Regla de L'Hopital: *Sean dos funciones, f(x) y g(x), tales que* $\lim_{x \to c} f(x) = 0$ *y* $\lim_{x \to c} g(x) = 0$, *entonces:*

$$\lim_{x \to c} \frac{f(x)}{g(x)} = \lim_{x \to c} \frac{f'(x)}{g'(x)}$$

"Ranking" de las complejidades más habituales:

$n!$	(intratable)
a^n	
n^a	con $a > 1$
$nlog(n)$	
n	
$log(n)$	
1	(la mejor)

Ejercicio

Justifica si es falsa o cierta las siguiente afirmación: Si $f(n) \in O(n^2)$ y $g(n) \in O(n^3)$, entonces $f(n) + g(n) \in O(n^2)$.

Es falsa. Consideremos el siguiente contraejemplo. Sen $f(n) = n^2$ y $g(n) = n^3$. Obviamente, $f(n) = n^2 \in O(n^2)$ y $g(n) = n^3 \in O(n^3)$. Aplicando el teorema del límite:

$$\lim_{n\to\infty} \frac{n^2+n^3}{n^2} = \lim_{n\to\infty} 1 + n = \infty$$

luego $f(n) + g(n) \notin O(n^2)$.

Ejercicio

Justifica si es falsa o cierta las siguiente afirmación:
$O(2^n + 3^n) = O(3^n)$.

Es cierta.

- Sea $f(n) \in O(2^n + 3^n)$, entonces existen c_0 y n_0 tales que para todo $n \geq n_0$ se cumple:
  $f(n) \leq c_0(2^n + 3^n) \leq c_0(3^n + 3^n) = 2c_0 3^n \Rightarrow f(n) \in O(3^n)$.
- Sea $f(n) \in O(3^n)$, entonces existen c_1 y n_0 tales que para todo $n \geq n_0$ se cumple: $f(n) \leq c_1 3^n \leq c_1(2^n + 3^n) \Rightarrow$ $f(n) \in O(2^n + 3^n)$.

Ejercicio

Demostrar que es cierto: $f \in O(f)$

Basta con tomar $n_0 = 1$ y $c = 1$: $f(n) \leq f(n) \forall n \geq 1$.

Ejercicio

Demostrar que es cierto: $O(f) \subseteq O(g) \Leftrightarrow f \in O(g)$

Consideremos los dos sentidos de la implicación:

- Como $f \in O(f) \Rightarrow f \in O(g)$, luego $O(f) \subseteq O(g) \Rightarrow f \in O(g)$.
- Sea $h(n) \in O(f)$: $\exists n_0, c_1$ tales que: $h(n) \leq c_1 f(n)\ \forall n \geq n_1$. Como $f(n) \in O(g)$: $\exists n_0, c_2$ tales que: $f(n) \leq c_2 g(n)$ $\forall n \geq n_2$. A partir de $n_0 = max(n_1, n_2)$ se cumplen ambas y por tanto se cumple:

  $$h(n) \leq c_1 f(n) \leq c_1(c_2 g(n)) = c_0 g(n)\ \forall n \geq n_0$$

  luego: $O(f) \subseteq O(g) \Leftarrow f \in O(g)$.

Ejercicio

Demostrar que es cierto: $O(f) = O(g) \Leftrightarrow f \in O(g)$ *y* $g \in O(f)$

- Basta ver los apartados anteriores para demostrar que $O(f) = O(g) \Rightarrow f \in O(g)$ y $g \in O(f)$.
- Si $f \in O(g)$, entonces $O(f) \subseteq O(g)$, según el apartado anterior. Aplicando la misma idea para $g \in O(f) \Rightarrow O(g) \subseteq O(f)$. Luego $O(f) = O(g)$.

Ejercicio

Demuestra o refuta la siguiente afirmación:
$O(f+g) = O(min(f,g))$

$O(f+g) = O(min(f,g))$:

- $min(f,g) \leq f+g \Rightarrow O(min(f,g)) \subset O(f+g)$
- $O(f+g) \not\subset O(min(f,g))$ ya que, por ejemplo: $O(n^2+n) \neq O(n) = O(min(n^2,n))$

Ejercicio

Demuestra o refuta la siguiente afirmación:
$O(f+g) = O(max(f,g))$

$O(f+g) = O(max(f,g))$:

- $f+g = max(f,g) + min(f,g) \leq 2max(f,g) \Rightarrow O(f+g) \subset O(2max(f,g)) = O(max(f,g))$
- $max(f,g) \leq max(f,g) + min(f,g) = f+g \Rightarrow O(max(f,g) \subset O(f+g))$

Ejercicio

Demuestra o refuta la siguiente afirmación:
$O(f+g) = O(max(f,g) + min(f,g))$

Es trivialmente cierta ya que $f + g = max(f,g) + min(f,g)$.

Ejercicio

Demuestra o refuta la siguiente afirmación:
$\Omega(f+g) = \Omega(min(f,g))$

- $f+g \geq min(f,g) \Rightarrow \Omega(f+g) \subset \Omega(min(f,g))$.
- $\Omega(min(f,g)) \not\subset \Omega(f+g)$ ya que, por ejemplo:
  $\Omega(n^2+n) \neq \Omega(n) = \Omega(min(n^2,n))$

Ejercicio

Demuestra o refuta la siguiente afirmación:
$\Omega(f+g) = \Omega(max(f,g))$

- $f+g = max(f,g) + min(f,g) \leq 2max(f,g) \Rightarrow \Omega(2max(f,g)) = \Omega(max(f,g)) \subset \Omega(f+g)$
- $max(f,g) \leq max(f,g) + min(f,g) = f+g \Rightarrow \Omega(f+g) \subset \Omega(max(f,g))$

Ejercicio

Demuestra o refuta la siguiente afirmación:
$\Omega(f+g) = \Omega(max(f,g) + min(f,g))$

Trivialmente cierta, por ser una igualdad entre funciones.

Ejercicio

Demuestra o refuta la siguiente afirmación:
$\Theta(f+g) = \Theta(min(f,g))$

Falsa. Ver contraejemplo con $f(n) = n^2$ y $g(n) = n$.

Ejercicio

Demuestra o refuta la siguiente afirmación:
$\Theta(f+g) = \Theta(max(f,g))$

Cierta, ya que se cumple $O(f+g) = O(max(f,g))$ y $\Omega(f+g) = \Omega(max(f,g))$.

Ejercicio

Demuestra o refuta la siguiente afirmación:
$\Theta(f+g) = \Theta(max(f,g) + min(f,g))$

Cierta por tratarse de una igualdad entre funciones.

Ejercicio

Demuestra la siguiente cadena de inclusiones:

$$O(1) \subset O(log(n)) \subset O(n) \subset ... \subset O(n^k) \subset O(2^n) \subset O(n!)$$

- Utilizaremos la regla de límite combinada con el teorema de L'Hopital.
- Es útil tener en cuenta las siguientes igualdades:

$$log_a n = \frac{log_b n}{log_b a} \text{ y } log(n) = log_2 n = \frac{Ln(n)}{Ln2}$$

Cálculo de límites:

- $lim_{n\to\infty}\frac{log(n)}{1} = lim_{n\to\infty}n = \infty.$
- $lim_{n\to\infty}\frac{n}{log(n)} = lim_{n\to\infty}\frac{nLn2}{Ln(n)} =_{L'Hopital} lim_{n\to\infty}\frac{Ln2}{1/n} = lim_{n\to\infty}nLn2 = \infty.$
- $lim_{n\to\infty}\frac{n^{k+1}}{n^k} = lim_{n\to\infty}n = \infty.$
- $lim_{n\to\infty}\frac{2^n}{n^k} =_{L'Hopital} lim_{n\to\infty}\frac{2^nLn2}{kn^{k-1}} =_{L'Hopital} lim_{n\to\infty}\frac{2^n(Ln2)^2}{k(k-1)n^{k-2}} =_{L'Hopital} ...(k-2veces) = lim_{n\to\infty}\frac{2^n(Ln2)^k}{k!n^0} = lim_{n\to\infty}2^n = \infty$
  Nota: $f(x) = a^x \Rightarrow f'(x) = a^xLn2$
- $lim_{n\to\infty}\frac{n!}{2^n} = lim_{n\to\infty}\frac{n}{2}\frac{n-1}{2}...\frac{4}{2}\frac{3}{2}\frac{2}{2}\frac{1}{2} \geq lim_{n\to\infty}\frac{4}{2}\frac{4}{2}...\frac{4}{2}\frac{3}{2}\frac{2}{2}\frac{1}{2} = \frac{3}{4}lim_{n\to\infty}2^{n-3} = \infty.$

Ejercicio

Comparar, respecto de O y Ω, los siguientes pares de funciones:
$2^{n+1}, 2^n$

Como $2^{n+1} = 2 \cdot 2^n \Rightarrow$ Solamente se diferencian en una constante multiplicativa. Luego: $O(2^{n+1}) = O(2^n)$ y $\Omega(2^{n+1}) = \Omega(2^n)$.

Ejercicio

Comparar, respecto de O y Ω, los siguientes pares de funciones:
$(n+1)!, n!$

Aplicando el teorema del límite:

$$lim_{n\to\infty}\frac{(n+1)!}{n!} = lim_{n\to\infty} n+1 = \infty$$

Por lo tanto, $O(n!) \subset O((n+1)!)$ y $\Omega((n+1)!) \subset \Omega(n!)$.

Ejercicio

Comparar, respecto de O y Ω, los siguientes pares de funciones: n+log n y $\sqrt{n}$:

Como $O(n + log(n)) = O(n)$ y $\Omega(n + log(n)) = \Omega(n)$ sustituimos y aplicando el teorema del límite:

$$lim_{n\to\infty}\frac{n}{\sqrt{n}} = lim_{n\to\infty}\sqrt{n} = \infty$$

Por lo tanto, $O(\sqrt{n}) \subset O(n)$ y $\Omega(n) \subset \Omega(\sqrt{n})$.

Ejercicio

Comparar, respecto de O y Ω, los siguientes pares de funciones: $n + 2\sqrt{n}$ *y* n^2

Como $O(n + 2\sqrt{n}) = O(n)$ y $\Omega(n + 2\sqrt{n}) = \Omega(n)$ sustituimos y aplicando el teorema del límite:

$$lim_{n\to\infty}\frac{n^2}{n} = lim_{n\to\infty}n = \infty$$

Por lo tanto, $O(n + 2\sqrt{n}) = O(n) \subset O(n^2)$ y $\Omega(n^2) \subset \Omega(n + 2\sqrt{n})$.

Ejercicio

Comparar, respecto de O y Ω, los siguientes pares de funciones: $2(logn)^2$ *y* $1 + log$ *n*

Teniendo en cuenta que se cumple:

- $O(2(log(n))^2) = O((log(n))^2)$ y $\Omega(2(log(n))^2) = \Omega((log(n))^2)$
- $O(1 + log(n)) = O(log(n))$ y $\Omega(1 + log(n)) = \Omega(log(n))$

podemos aplicar el teorema de límite:

$$lim_{n\to\infty}\frac{(log(n))^2}{log(n)} = lim_{n\to\infty}log(n) = \infty$$

Por lo tanto: $O(1 + log(n)) \subset O(2(log(n))^2)$ y $\Omega(2(log(n))^2) \subset \Omega(1 + log(n))$.

Ejercicio

Comparar, respecto de O y Ω, los siguientes pares de funciones:
Para cualquier a $\in \mathbb{R}^+$, log n y n^a

Aplicando el teorema del límite y el teorema de L'Hopital:

$$lim_{n\to\infty}\frac{n^a}{log(n)} = lim_{n\to\infty}\frac{an^{a-1}}{log(e)/n} =$$
$$lim_{n\to\infty}\frac{an^a}{log(e)} = \frac{a}{log(e)}lim_{n\to\infty}n^a = \infty$$

Por lo tanto, $O(log(n)) \subset O(n^a)$ y $\Omega(n^a) \subset \Omega(log(n))$.

Ejercicio

Demuestra o refuta la siguiente afirmación: $2^n + n^{99} \in O(n^{99})$

$2^n + n^{99} \in \underbrace{O(max(2^n, n^{99})) = O(2^n)}_{NosReferimosAsusOrdedes}$ pero $2^n + n^{99} \notin O(n^{99})$

porque:

$$lim_{n\to\infty} \frac{2^n+n^{99}}{n^{99}} = \infty$$

luego, es falsa.

Ejercicio

Demuestra o refuta la siguiente afirmación: $2^n + n^{99} \in \Omega(n^{99})$

Como $2^n + n^{99} \geq n^{99}$, es cierta.

Ejercicio

Demuestra o refuta la siguiente afirmación: $2^n + n^{99} \in \Theta(n^{99})$

Es falso porque ya se ha demostrado que: $2^n + n^{99} \notin O(n^{99})$.

Ejercicio

Demuestra o refuta la siguiente afirmación: $2^n + n^{99} \in O(2^n)$

$2^n + n^{99} \in O(max(2^n, n^{99})) = O(2^n)$. Luego, es cierta.

Ejercicio

Demuestra o refuta la siguiente afirmación: $2^n + n^{99} \in \Omega(2^n)$

$2^n + n^{99} \geq n^{99}$. Luego, es cierta.

Ejercicio

Demuestra o refuta la siguiente afirmación: $2^n + n^{99} \in \Theta(2^n)$

Es cierta por los dos apartados anteriores.

Operaciones elementales: aquellas cuyo tiempo de ejecución es constante:

- Asignación
- Escritura/lectura
- Operaciones algebraicas
- Comparaciones
- $T(N) = cte \Leftrightarrow T(N) \in \Theta(1)$.

Secuencias. Sean P_1 y P_2 dos fragmentos consecutivos de un algoritmo con tiempos de ejecución:

$$P_1 \rightarrow T_1(N) \in \Theta(g)$$
$$P_2 \rightarrow T_2(N) \in \Theta(f)$$

Entonces:

$$P_1; P_2 \rightarrow T_1(N) + T_2(N) \in \Theta(T_1 + T_2) = \Theta(max(T_1, T_2))$$

Sentencias condicionales

si COND **entonces**
  S1
si no
  S2
fin si

- **Caso peor**: $T(N) = T(COND) + maximo\{T(S1), T(S2)\}$
- **Caso más probable**: $T(N) = T(COND) + p_1 t(S1) + p_2 t(S2)$ donde $p_i =$ probabilidad de pasar por la rama S_i.

En ambos casos la eficiencia será la eficiencia de la rama con mayor complejidad → Distinguir entre el caso peor y el caso promedio no es relevante → El enfoque teórico es aproximado.

Bucles Consideremos el siguiente bucle: el compilador lo traduce de la siguiente forma:

$I \leftarrow 1$
mientras $I \leq M$ **hacer**
P(I)
$I \leftarrow I + 1$
fin mientras

Su tiempo de ejecución:

$$T(M) = c + (M+1)c + M * (T(P(I))) + c$$
$$\Downarrow$$
$$T(M) = c_1 + M * c_2$$
$$\Downarrow$$
$$\text{si } T(p(I)) = cte \Rightarrow T(M) \in \Theta(M)$$

Los bucles generan complejidad!!.

Procedimientos y funciones. Tiempo de ejecución de un procedimiento (o función):

- Llamada al porcedimiento o función $\Theta(1)$.
- Evaluación de los parámetros $\Theta(1)$.
- Ejecución del código.

Comentario adicional:
Sean $T_1(N)$ y $T_2(N)$ tales que $T_1(N) = \frac{1}{2}T_2(N)$. $T_1(N)$ es mucho mejor que $T_2(N)$ pero $\Theta(T_1(N)) = \Theta(T_2(N)) \Rightarrow$ El enfoque analítico no es demasiado fino!!!.

Algunas identidades útiles:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{1}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \tag{2}$$

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} \tag{3}$$

$$\sum_{i=1}^{n} 2^i i = (n-1)2^{n+1} + 2 \tag{4}$$

$$\sum_{i=1}^{n} a^i = \frac{a^{n+1}-1}{a-1} \tag{5}$$

Ejercicio

Dado el siguiente algoritmo, determina su complejidad.

fun *valor(n)*
  desde $i \leftarrow 1$ **hasta** N **hacer**
    desde $j \leftarrow 1$ **hasta** 20 **hacer**
      $aux \leftarrow j * 2$
      $aux \leftarrow aux * i + r$
    fin desde
  fin desde
  devolver aux
fin fun

Solución:
El tiempo de ejecución es:

$$T(n) = c_1 + \textstyle\sum_{i=1}^{n} \underbrace{\sum_{j=1}^{20} c_2 + c_3}_{cte}$$

$$\Updownarrow$$

$$T(n) = c_1 + \textstyle\sum_{i=1}^{n} c_4$$

$$\Updownarrow$$

$$T(n) = c_1 + nc_4 \Rightarrow T(n) \in \Theta(n)$$

Nota: El valor concreto de las variables no influye → En el resto de los ejercicios no distinguiremos entre variables.

Ejercicio

Dado el siguiente algoritmo, determina su complejidad.

fun *valor(n)*
  $r \leftarrow 1$
  desde $i \leftarrow 1$ **hasta** n **hacer**
    desde $j \leftarrow 1$ **hasta** n **hacer**
      si $i < j$ **entonces**
        $aux \leftarrow i + j$
      si no
        $aux \leftarrow i - j$
      fin si
    fin desde
  fin desde
  devolver *a ux*
fin fun

Solución:
El tiempo de ejecución es:

$$T(n) = c + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{condicion} + c_{rama}$$
$$\Updownarrow$$
$$T(n) = c + \sum_{i=1}^{n} c * n$$
$$\Updownarrow$$
$$T(n) = c + cn^2 \Rightarrow T(n) \in \Theta(n^2)$$

Ejercicio

Dado el siguiente algoritmo, determina su complejidad.

fun *valor(n)*
  desde $i \leftarrow 1$ **hasta** n **hacer**
    desde $j \leftarrow 1$ **hasta** i **hacer**
      si $i < j$ **entonces**
        $aux \leftarrow i + j$
        si $i + 2 < j$ **entonces**
          $aux \leftarrow aux * 2$
        fin si
      fin si
    fin desde
  fin desde
  devolver *a ux*
fin fun

Solución:
El tiempo de ejecución es:

$$T(n) = c + \sum_{i=1}^{n} \sum_{j=1}^{i} c$$
$$\Updownarrow$$
$$T(n) = c + \sum_{i=1}^{n} i * c$$
$$\Updownarrow$$
$$T(n) = c + c\frac{n(n+1)}{2} \Rightarrow T(n) \in \Theta(n^2)$$

Ejercicio

Ordenación por inserción directa. Determina su complejidad.

fun *insercion(L[1..n])*
  desde $i \leftarrow 2$ **hasta** n **hacer**
    $x \leftarrow L[i]$
    $j \leftarrow i-1$
    mientras $j > 0 \wedge x < L[j]$ **hacer**
      $L[j+1] \leftarrow L[i]$
      $j \leftarrow j-1$
    fin mientras
    $L[j+1] \leftarrow x$
  fin desde
fin fun

Notas:

- En cada iteración del bucle principal colocamos en la sublista ya ordenada el elemento L[i] moviendo los que sean mayores que él
- En el caso promedio supondremos que moveremos $(i-1)/2$.
- $T(n) = \sum_{i=2}^{n}(((i-1)/2+c)+c)+c = \frac{1}{2}\sum_{i=2}^{n} i + nc + c = \frac{1}{2}\frac{n(n+1)}{2} - 1 + nc + c \Rightarrow T(n) \in \Theta(n^2)$

Ejercicio

. *Ordenación por selección directa. Determina su complejidad.*

```
fun seleccion(L[1..n])
   desde i ← 1 hasta n-1 hacer
      menor← L[i]
      pos← i
      desde j ← i+1 hasta n hacer
         si L[j] < menor entonces
            menor ← L[j]
            pos← j
         fin si
      fin desde
      L[pos] ← L[i]
      L[i] ← menor
   fin desde
```

Notas:

- El bucle exterior se ejecuta n-1 veces.
- El bucle interior se ejecuta n-i veces.
- $T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} cte = \sum_{i=1}^{n-1} (n-i)cte = (n-1)n - \sum_{i=1}^{n-1} i = (n-1)n - \frac{(n-1)}{2} \Rightarrow T(n) \in \Theta(n^2)$

Ejercicio

. *Ordenación por el método de la burbuja. Determina su complejidad.*

fun *burbuja(L[1..n])*
  desde $i \leftarrow 1$ **hasta** *n-1* **hacer**
    desde $j \leftarrow 1$ **hasta** *n-i* **hacer**
      si $L[j] > L[j+1]$ **entonces**
        $aux \leftarrow L[j]$
        $L[j] \leftarrow L[j+1]$
        $L[j+1] \leftarrow aux$
      fin si
    fin desde
  fin desde
fin fun

Notas:

- El bucle exterior se ejecuta n-1 veces.
- El bucle interior se ejecuta n-i veces.
- $T(n) = \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} cte = \sum_{i=1}^{n-1} (n-i) = n(n-1) - \frac{(n-1)n}{2} \Rightarrow T(n) \in \Theta(n^2)$

Ejercicio

Dado el siguiente algoritmo, determina su complejidad.

fun *valor(n)*
  desde $i \leftarrow 1$ **hasta** n **hacer**
    desde $j \leftarrow 1$ **hasta** i **hacer**
      desde $k \leftarrow 1$ **hasta** n **hacer**
        $aux \leftarrow aux * (i * j + k)$
      fin desde
    fin desde
  fin desde
  devolver *a ux*
fin fun

Solución:
El tiempo de ejecución es:

$$T(n) = c + \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{n} c$$
$$\Updownarrow$$
$$T(n) = c + \sum_{i=1}^{n} \sum_{j=1}^{i} nc$$
$$\Updownarrow$$
$$T(n) = c + \sum_{i=1}^{n} n \sum_{j=1}^{i} c$$
$$\Updownarrow$$
$$T(n) = c + n \sum_{i=1}^{n} \sum_{j=1}^{i} c$$
$$\Updownarrow$$
$$T(n) = c + nc\frac{n(n+1)}{2} \Rightarrow T(n) \in \Theta(n^3)$$

Ejercicio

Dado el siguiente algoritmo, determina su complejidad.

fun *valor(n)*
  desde $i \leftarrow 1$ **hasta** n **hacer**
    desde $j \leftarrow 1$ **hasta** i **hacer**
      desde $k \leftarrow 1$ **hasta** j **hacer**
        si $i < j$ **entonces**
          $aux \leftarrow i + j$
          si $i + 2 < j$ **entonces**
            $aux \leftarrow aux * 2$
          fin si
        fin si
      fin desde
    fin desde
  fin desde

Solución:
El tiempo de ejecución es:

$$T(n) = c + \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} c$$
$$\Updownarrow$$
$$T(n) = c + c\sum_{i=1}^{n} \sum_{j=1}^{i} j$$
$$\Updownarrow$$
$$T(n) = c + c\sum_{i=1}^{n} \frac{i(i+1)}{2}$$
$$\Updownarrow$$
$$T(n) = c + c\sum_{i=1}^{n} \frac{i^2}{2} + \frac{i}{2}$$
$$\Updownarrow$$
$$T(n) = c + c\frac{n(n+1)(2n+1)}{6} + c\frac{n(n+1)}{4} \Rightarrow T(n) \in \Theta(n^3)$$