Cálculo del Orden de Complejidad de un Algoritmo Iterativo

Dado el siguiente procedimiento, calcula y demuestra mediante la regla del umbral o del límite a qué orden de complejidad pertenece:

```
procedimiento proc(E entero: n)
var
    entero: z, d, x
inicio
    z ← 0
    d ← 1
    mientras d ≤ n hacer
        x ← d
        mientras x ≥ 0 hacer
            z ← z + x
            x ← x – 1
        fin_mientras
        d ← d + 1
    fin_mientras
fin_procedimiento
```

Para calcular el coste total de este algoritmo iterativo, se ha de sumar el coste de cada iteración del bucle externo y del bucle interno. Un método general (no es el único método para encontrar el orden de complejidad) es el que se detalla a continuación y que seguiremos en esta asignatura para todos los problemas relacionados con el cálculo del orden de complejidad de un algoritmo iterativo.

1. Determinación del número de veces que se ejecuta cada bucle

El primer paso sería establecer cuántas veces se ejecuta cada uno de los bucles de este algoritmo. Para ello se pueden ir realizando varias trazas y se va observando el comportamiento de las variables que controlan cada bucle. Es importante prestar atención a los valores iniciales y finales que estas variables toman, así como a la expresión usada para asignar nuevos valores a estas variables, las más frecuentes: sumando o restando valores constantes, o bien, multiplicando o dividiendo por un valor constante.

En el siguiente ejemplo n=3:

n	d	x	z	
3	**1**		0	Valores Iniciales
		1	1	
		0	2	
		-1		Fin bucle interno, para d=1 el bucle interno se ejecuta 2 veces
	2			
		2	4	
		1	5	
		0	5	
		-1		Fin bucle interno, para d=2 el bucle interno se ejecuta 3 veces
	3			
		3	8	
		2	10	
		1	11	
		0	11	
		-1		Fin bucle interno, para d=3 el bucle interno se ejecuta 4 veces
	4			**Fin bucle externo**
				Fin procedimiento

El bucle externo está controlado por la variable d, que se inicia en 1 y se va incrementando de 1 en 1 hasta que llega a n. En la traza realizada, para n=3 se ha ejecutado 3 veces, ya que d se va incrementando de forma lineal de 1 en 1, comenzando en 1 y acabando para d=n, por tanto el bucle externo se ejecuta **n** veces.

El bucle interno está controlado por la variable x, que se inicia en el valor **d**, y se va decrementando de 1 en 1 (de forma lineal también) hasta que x toma el valor -1. Cuando d=1, se ha ejecutado 2 veces, cuando d=2 se ha ejecutado 3 veces, cuando d=3 el bucle interno se ejecutó 4 veces. Por tanto, al tener un decremento lineal de 1 en 1, el bucle interno se ejecuta **d+1** veces.

Estas observaciones pueden ser confirmadas ejecutando otras trazas del mismo algoritmo para valores diferentes de n, por ejemplo es posible comprobar que cuando n= 8, el bucle externo se ejecutará 8 veces, mientras que el bucle interno se ejecutará para d=8, 9 veces, para d=7, 8 veces, y así sucesivamente.

2. Obtención de la expresión t(n)

```
procedimiento proc(E entero: n)
var
    entero: z, d, x
inicio
    z ← 0  (ta)
    d ← 1  (ta)
    mientras d ≤ n hacer  (tc)
        x ← d  (ta)
        mientras x ≥ 0 hacer  (tc)
            z ← z + x  (ta + to)
            x ← x – 1  (ta + to)
        fin_mientras
        d ← d + 1  (ta + to)
    fin_mientras
fin_procedimiento
```

Los bucles pueden ser expresados mediante sumatorios, por tanto:

- El bucle externo se ejecuta n veces, desde d=1 hasta n, lo cual expresaremos en el sumatorio usando una variable α que toma valores desde α=1 hasta n.
- El bucle interno se ejecuta d+1 veces, como d está representada en el sumatorio por α, expresaremos el bucle interno como otro sumatorio cuya variable β toma valores desde 1 hasta α+1.

$$t(n) = t_a + t_a + t_c + \sum_{\alpha=1}^{n}\left(t_c + t_a + t_c + \sum_{\beta=1}^{\alpha+1}\left(t_c + t_o + t_a + t_o + t_a\right) + t_o + t_a \right)$$

Todas las instrucciones básicas tienen un coste costante por tanto:

$$t(n) = 3 + \sum_{\alpha=1}^{n}\left(5 + \sum_{\beta=1}^{\alpha+1} 5 \right)$$

3. Resolución de la expresión obtenida para t(n)

$t(n) = 3 + \sum_{\alpha=1}^{n} \left(5 + \sum_{\beta=1}^{\alpha+1} 5 \right)$ Resolviendo el sumatorio interno obtendremos:

$= 3 + \sum_{\alpha=1}^{n} (5 + 5(\alpha + 1)) = 3 + \sum_{\alpha=1}^{n} (5 + 5\alpha + 5)$ Descomponiendo este sumatorio:

$= 3 + \sum_{\alpha=1}^{n} 10 + 5 \sum_{\alpha=1}^{n} \alpha$ Resolviendo cada uno de ellos:

$= 3 + 10n + 5 * \left(n \frac{(n+1)}{2} \right)$

$t(n) = 3 + 10n + \frac{5}{2} n^2 + \frac{5}{2} n$ Agrupando los términos

$t(n) = 3 + \frac{25}{2} n + \frac{5}{2} n^2$

4. Determinación del Orden Superior de t(n)

Aplicando la Regla del Máximo:

Si $f_1 \in O(g) \wedge f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\max(g,h))$

Si:

- $3 \in O(1)$
- $\left(\frac{25}{2} n \right) \in O(n)$
- $\left(\frac{5}{2} n^2 \right) \in O(n^2)$

Entonces $t(n) = 3 + \frac{25}{2} n + \frac{5}{2} n^2 \in O(\max(1, n, n^2))$

Por tanto: $O(t(n)) \in O(n^2)$

5. Demostración

$$3+\frac{25}{2}n+\frac{5}{2}n^2 \in O(n^2)$$

Existen dos reglas que se pueden utilizar para realizar la demostración.

a) Por la Regla del Umbral

{t: N$\rightarrow$R$^+$ | $\exists$ c $\in$ R+ , $\exists$ $n_o \in$ N | $\forall n \geq n_o$ t(n) $\leq$ c*f(n)* }

Se cumple si existe n_o y c tal que:

$$t(n) \leq cf(n)$$

$$3+\frac{25}{2}n+\frac{5}{2}n^2 \leq c \cdot n^2$$

por ejemplo para n_o=1 y c=18, por tanto se cumple.

Por tanto demostramos que t(n) pertenece al orden superior de n^2.

b) Por la Regla del Límite:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=k$$

$$a) Si \quad k=0 \ entonces \ f \in O(g) \ pero \ f \notin \Theta(g)$$

$$b) Si \quad k=+\infty \ entonces \ f \in \Omega(g) \ pero \ f \notin \Theta(g)$$

$$c) Si \quad k \neq 0 \ y \ k<\infty \ entonces \ f \in \Theta(g)$$

$$\lim_{n\to\infty}\frac{3+\frac{25}{2}n+\frac{5}{2}n^2}{n^2}=\frac{5}{2}$$

Por tanto pertenece al orden superior de n^2, y además con esta misma regla del límite, se demuestra que t(n) pertenece al orden exacto de n^2.