Examen Parcial

Sánchez Sauñe Cristhian

① Determine la complejidad, usando definición, de un algoritmo cuyo costo temporal es $f(n) = 3 \cdot 2^n + n^2$

Según definición de Big O:

$f, g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$

$O(g(n)) = \{ f(n) : \exists c \in \mathbb{R}^+, \exists K \in \mathbb{N}, \forall n \geq K, f(n) \leq c \times g(n) \}$

Sea $K = 1$, encontraremos un c

$\forall n \geq 1 \qquad 3 \cdot 2^n + n^2 \leq c\, g(n)$

Supongamos que $f(n) \in O(2^n)$, eligiendo $K = 1$ y $C = 4$

Demostraremos $\forall n \geq 1 \quad 3 \cdot 2^n + n^2 \leq 4 \cdot 2^n$

Caso base $n = 1$

$$3 \cdot 2^1 + 1^2 \leq 4 \cdot 2^1$$
$$6 + 1 \leq 8$$
$$7 \leq 8$$

para $n+1$ h.i. $\quad 3 \cdot 2^{n+1} + (n+1)^2 \leq 4 \cdot 2^{n+1}$

$$3 \cdot 2^{n+1} + n^2 + 2n + 1 \leq 4 \cdot 2^{n+1}$$

$$(n+1)^2 \leq 2^{n+1}$$
$$\log (n+1)^2 \leq \log 2^{n+1}$$

$$2 \log (n+1) \leq (n+1) \log 2$$

como $\log 2 \neq 0$, dividimos en ambos lados

$$2 \frac{\log (n+1)}{\log 2} \leq n+1$$

$$2 \log (n+1) \leq n+1$$
$$< \log (n+1)$$
$$\log (n+1) \leq n+1$$

Siempre se cumple

∴ Demostrado para $n+1$ //

② Demuestre que $f(n) = n^3 + n^2 \log n^2 \in \Theta(n^3)$

Por definición:

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

$$\Theta(g(n)) = \{ f(n) : \exists c_1, c_2 \in \mathbb{R}^+, \exists K \in \mathbb{N}, \forall n \geq K, c_1 g(n) \leq f(n) \leq c_2 g(n) \}$$

Sabemos $f(n) = n^3 + 2n^2 \log n$

Un modo de hacerlo es por inducción sobre n. Elegimos $K = 1$ y $c_1 = 1$, $c_2 = 3$, es decir, demostraremos $\forall n \geq 1$

$$1n^3 \leq n^3 + 2n^2 \log n \leq 3n^3$$

Caso base: $n = 1$

$$1^3 \leq 1^3 + 2(1^2)(0) \leq 3(1^3)$$

$$1 \leq 1 \leq 3$$

Paso inductivo:

h.i $\quad 1n^3 \leq n^3 + 2n^2 \log n \leq 3n^3$. Demostraremos para $n+1$

$$1(n+1)^3 \leq (n+1)^3 + 2(n+1)^2 \log(n+1) \leq 3(n+1)^3$$

como $(n+1)^2 > 0$, podemos dividir por ese término

$$(n+1) \leq (n+1) + 2\log(n+1) \leq 3(n+1)$$

tenemos:

$$(n+1) \leq (n+1) + 2\log(n+1) \quad \wedge \quad (n+1) + 2\log(n+1) \leq 3(n+1)$$

Sabemos que $2\log(n+1) > 1$ (*)

Sabemos que $2\log(n+1) \leq 2(n+1)$

$$(n+1) + 2\log(n+1) \leq (n+1) + 2(n+1)$$

$$(n+1) + 2\log(n+1) \leq 3(n+1) \quad (**)$$

$\Rightarrow$ (*) $\wedge$ (**) se cumple

$\Rightarrow$ Se cumple para $n+1$.

$\therefore f(n) = n^3 + n^2 \log n^2 \in \Theta(n^3)$

(3) Según el algoritmo, debemos ver el número de veces en el que esta llamando a la recurrencia (3 veces); analizando también notamos que La reducción del problema en cada llamada viene dada por una escala de 2. (n div 2).

El coste total de las llamadas distintas a la recursividad viene dada por $c * n^k$. Dichas llamadas son constantes, es decir

$$c * n^k = 1 \Rightarrow k = 0 \quad \text{con } c \in \mathbb{R}^+$$

Ahora, podemos dar la forma de recurrencia como

$$T(n) = \begin{cases} c * n^{k=0} = c & 1 \leq n < 2 \\ 3T\left(\frac{n}{2}\right) + c * n^{k=0} = 3T\left(\frac{n}{2}\right) + c & n \geq 2 \\ \hookrightarrow n \text{ div } 2 & \end{cases}$$

Podemos usar el teorema Maestro, para lo cual tenemos 3 condiciones.

$$T(n) = \begin{cases} (1)\ \Theta(n^k) & \text{si } a < b^k \\ (2)\ \Theta(n^k * \log n) & \text{si } a = b^k \\ (3)\ \Theta(n^{\log_b a}) & \text{si } a > b^k \end{cases} \quad \text{con} \quad T(n) = \begin{cases} c * n^k & 1 \leq n < b \\ a\,T\left(\frac{n}{b}\right) + c * n^k & n \geq b \end{cases}$$

Entonces , vemos que cumple para el 3er caso.

$$a > b^k \qquad 3 > 2^0$$
$$3 > 1$$

El coste viene dado por $\Theta(n^{\log_2 3})$

(4)

Sea una suma de $a, b \in M$

$a + b = S$, haremos el artificio de $a = x$ y $b = x - y$

Para analizar este problema, debemos primeramente tener un conjunto ordenado.
Para tener una complejidad parecida a la que piden en el problema, podemos usar Merge Sort. Entonces $\forall y \in M$ separadamente tendríamos que usar búsqueda binaria para verificar que $x - y$ existe en el conjunto M.
Sabemos que la búsqueda binaria toma una complejidad temporal de $O(\log n)$ y esta es ejecutada 'n' veces. Entonces el tiempo total sería $\Theta(n \log n)$.
Si M está ordenado, el problema puede ser resuelto en tiempo lineal leyendo el conjunto M (puede también ser una lista) al mismo tiempo hacia delante y hacia atrás.
A continuación se presenta un pseudocódigo:

Input: Lista M ordenada ascendentemente, S
Output: Verdadero si se cumple la condición del problema: existen 2 números en M cuya suma sea exactamente S.
Falso en otro caso.

```
i ← 1 , j ← n
mientras i ≤ j hacer
    Si M(i) + M(j) = S entonces
        retornar Verdadero
    Si M(i) + M(j) < S entonces
        i ← i + 1
    Si no
        j ← j - 1
retornar Falso
```

El algoritmo no distingue si $M[i] \neq M[j]$, es decir los elementos de la suma pueden ser el mismo.