



Algoritmia y Complejidad

Ejercicios tema 1

Laboratorio Miércoles 15:00-17:00

Integrantes:

Carlos Javier Hellín Asensio
Diego Gutiérrez Marcos

Curso: 2020-2021

Ejercicio 3

Como n tiene más peso, contamos siempre que una línea de código o una condición vale 1, aunque en esa misma línea haya sumas, multiplicaciones, asignaciones, etc.... Como, por ejemplo, hacemos en valor \leftarrow valor + i

```

fun Calculo(x,y,z: entero) dev valor:entero
var i,j,t: entero
valor ← 0
Desde i ← x hasta y Hacer valor ← valor + i fdesde
si (valor ÷ (x+y)) <= 1, entonces Devolver z
si no
    t ← x + ((y-x) ÷ 2)
    Desde i ← x hasta y Hacer
        Desde j ← (3*x) hasta (3*y) Hacer
            valor ← valor + Minimo(i,j)
        fdesde
    fdesde
    valor ← valor + 4*Calculo(t,y,valor)
    Devolver valor
fsi
ffun
    
```

Handwritten annotations for complexity analysis:

- $\text{valor} \leftarrow 0 \rightarrow T(n) = 1$
- $\text{Desde } i \leftarrow x \text{ hasta } y \text{ Hacer } \text{valor} \leftarrow \text{valor} + i \text{ fdesde} \rightarrow T(n) = \sum_{i=x}^y 1$
- $\text{si } (\text{valor} \div (x+y)) \leq 1, \text{ entonces Devolver } z \rightarrow T(n) = 1$
- $\text{si no} \rightarrow T(n) = 1$
- $t \leftarrow x + ((y-x) \div 2) \quad (\div \text{ es la división entera }) \rightarrow T(n) = 1$
- $\text{Desde } i \leftarrow x \text{ hasta } y \text{ Hacer}$
 - $\text{Desde } j \leftarrow (3*x) \text{ hasta } (3*y) \text{ Hacer}$
 - $\text{valor} \leftarrow \text{valor} + \text{Minimo}(i,j) \rightarrow T(n) = \sum_{j=3x}^{3y} 1$
- $\text{fdesde} \rightarrow T(n) = \sum_{i=x}^y \sum_{j=3x}^{3y} 1$
- fdesde
- $\text{valor} \leftarrow \text{valor} + 4 * \text{Calculo}(t,y,\text{valor}) \rightarrow T(n) = 1 + T(n/2)$
- $\text{Devolver valor} \rightarrow T(n) = 1$
- fsi
- ffun

Final complexity equations:

$$T(n) = 1 + \max \left\{ 1, 1 + \sum_{i=x}^y \left(\sum_{j=3x}^{3y} 1 \right) + 1 + T(n/2) + 1 \right\}$$

$$T(n) = 1 + \sum_{i=x}^y 1 + 1 + \max \left\{ 1, 1 + \sum_{i=x}^y \left(\sum_{j=3x}^{3y} 1 \right) + 1 + T(n/2) + 1 \right\}$$

$$\begin{aligned}
 \textcircled{3} \quad T(n) &= 1 + \sum_{i=x}^y 1 + 1 + \max \left\{ 1, 1 + \sum_{i=x}^y \left(\sum_{j=3x}^{3y} 1 \right) + 1 + T(n/2) + 1 \right\} = \\
 &= 1 + n + 1 + 1 + n \cdot 3n + 1 + T(n/2) + 1 = \\
 &= 5 + n + 3n^2 + T(n/2)
 \end{aligned}$$

Cambio variable $\Rightarrow n = 2^k$

$$T(2^k) = 5 + 2^k + 3 \cdot 4^k + T(2^{k-1})$$

$$X^k = 5 + 2^k + 3 \cdot 4^k + X^{k-1}$$

$$X^k - X^{k-1} = 5 + 2^k + 3 \cdot 4^k$$

$$\text{Homogénea} \Rightarrow X^{(h)} \Rightarrow X^k - X^{k-1} = 0$$

$$X^{k-1} (X - 1) = 0$$

$\searrow \rightarrow x=1 \quad X^{(h)} = A$

$$\text{Particular} \Rightarrow X^{(p)} = B \cdot k + C \cdot 2^k + D \cdot 4^k$$

$$\text{General} \Rightarrow X^{(g)} = X^{(h)} + X^{(p)}$$

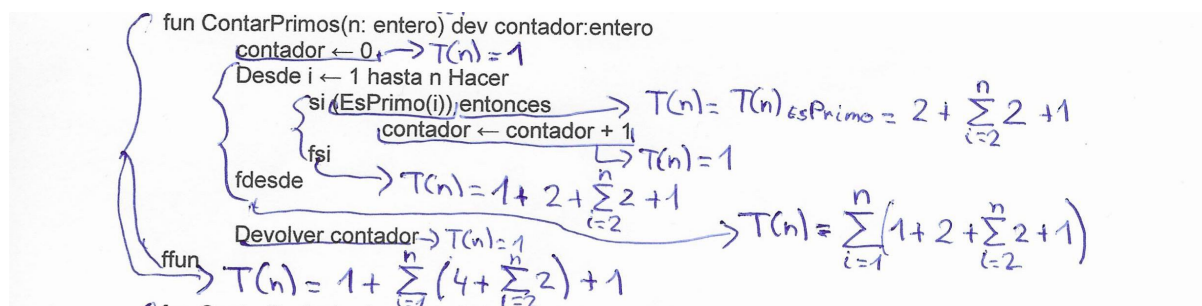
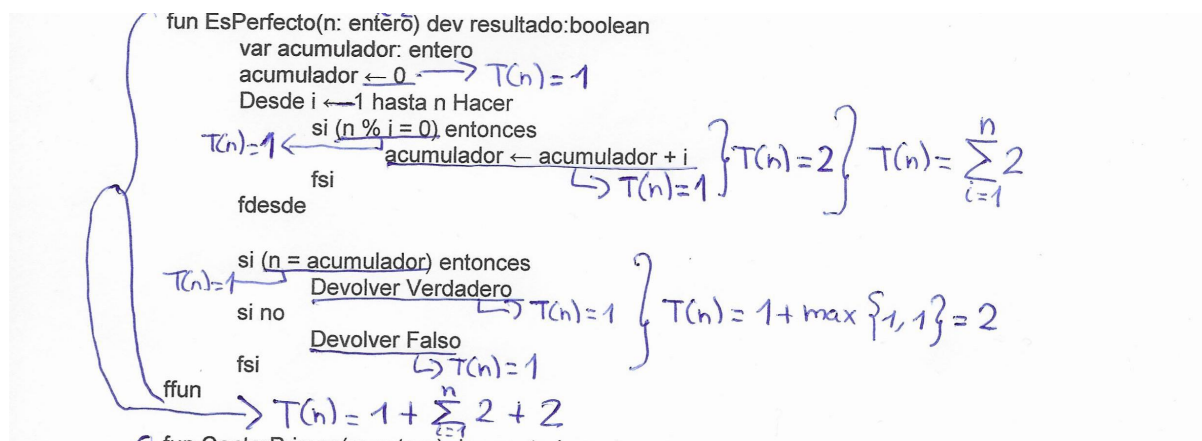
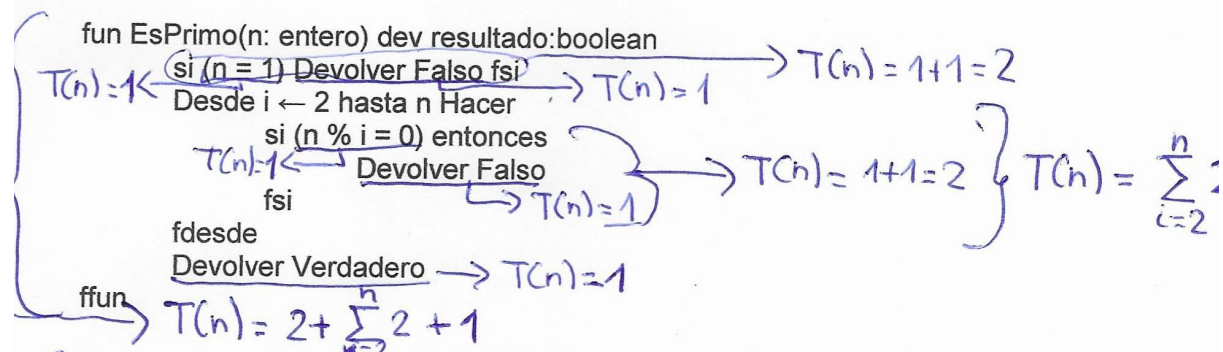
$$X^{(g)} = A + B \cdot k + C \cdot 2^k + D \cdot 4^k$$

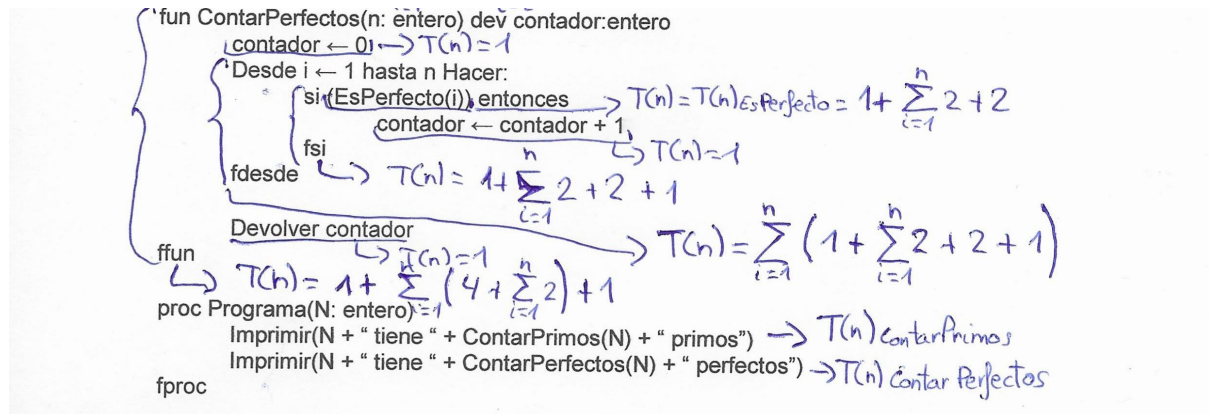
$$\text{Destruir cambio variable} \Rightarrow n = 2^k \Rightarrow k = \log_2 n$$

$$X^{(g)} = A + B \cdot \log_2 n + C \cdot n + D \cdot n^2$$

En esta ecuación, el peor caso es $D \cdot n^2$. Por tanto, la complejidad es $\boxed{O(n^2)}$

Ejercicio 7





Para este ejercicio se ha dividido el problema en varias funciones. Los principales son EsPrimo y EsPerfecto para saber si un número es primo y si es perfecto respectivamente.

En el caso de EsPrimo como parámetros de entrada tiene número a comprobar que llamaremos n. Si n es igual a 1 devuelve Falso y después se hace un bucle desde i igual a 2 hasta n para comprobar si el resto de la división de n entre i es igual a cero (usando módulo) entonces se devuelve Falso. En caso contrario y cuando haya terminado el bucle, se devuelve Verdadero.

Para el caso de EsPerfecto como parámetros el número a comprobar que llamaremos n. Se crea una variable acumulador iniciada a 0, y se hace un bucle desde i igual a 1 hasta n para comprobar si el resto de la división de n entre i es igual a cero (usando módulo) entonces se le suma al acumulador el valor i.

Una vez terminado el bucle, si n es igual al acumulador se devuelve Verdadero y en caso contrario, se devuelve Falso.

Por último, tenemos las funciones ContarPrimos y ContarPerfectos que son llamados desde el procedimiento Programa para mostrarlo en pantalla. Estas dos funciones tienen los mismos parámetros de entrada que es n (el número positivo del usuario) y se crea una variable contador para saber cuántos hay empezando por 0.

Se crea un bucle desde i igual 1 hasta n y se llama a las funciones respectivas (ContarPrimos llama EsPrimo y ContarPerfectos llamada EsPerfecto) si es cierto, a la variable contador se le suma un 1 más, hasta terminar el bucle y devolver el contador.

⑦

Función EsPrimo:

$$T(n) = 2 + \sum_{i=2}^n 2 + 1 = 3 + \sum_{i=2}^n 2 = 3 + 2 \cdot n$$

Función ContarPrimos:

$$T(n) = 1 + \sum_{i=1}^n \left(4 + \sum_{i=2}^n 2 \right) + 1 = 2 + \sum_{i=1}^n \left(4 + \sum_{i=2}^n 2 \right) =$$

$$= 2 + 4 \cdot n + 2 \cdot n^2$$

Complejidad $\Rightarrow n^2$

Función EsPerfecto:

$$T(n) = 1 + \sum_{i=1}^n 2 + 2 = 3 + \sum_{i=1}^n 2 = 3 + 2 \cdot n$$

Función ContarPerfectos:

$$T(n) = 1 + \sum_{i=1}^n \left(4 + \sum_{i=1}^n 2 \right) + 1 = 2 + \sum_{i=1}^n \left(4 + \sum_{i=2}^n 2 \right) =$$

$$= 2 + 4 \cdot n + 2 \cdot n^2$$

Complejidad $\Rightarrow n^2$

La complejidad del programa, tras aplicar la regla de la suma entre ContarPrimos y ContarPerfectos, es

$O(\max(n^2, n^2))$. Por tanto, la complejidad es de

$$\boxed{O(n^2)}$$

Ejercicio 9

The diagram shows a handwritten code snippet for a recursive function `Sumatorio` and its corresponding recurrence relation $T(n)$. The code is written in a pseudo-language with curly braces for function definition and control flow. The recurrence relation is derived from the code's logic.

```
fun Sumatorio(n: entero) dev valor:entero
{
  si (n = 1) entonces Devolver 1
  si no Devolver n + Sumatorio(n - 1)
}
ffun
```

Handwritten annotations and arrows show the mapping from code to the recurrence relation:

- The base case `si (n = 1) entonces Devolver 1` is annotated with $T(n) = 1$.
- The recursive case `si no Devolver n + Sumatorio(n - 1)` is annotated with $T(n) = 1 + T(n-1)$.
- The overall recurrence relation is derived as $T(n) = 1 + \max\{1, 1 + T(n-1)\}$.

Para esta función se ha hecho de forma recursiva. La condición de parada es si n es igual a 1 entonces devuelve 1, en caso contrario se llama de forma recursiva a `Sumatorio` para $n - 1$ y el resultado se suma a n para luego devolverlo.

De esta forma se consigue el sumatorio de $1 + 2 + 3 + 4 + \dots + n - 1 + n$

Como n tiene más peso, contamos siempre que una línea de código o una condición vale 1, aunque en esa misma línea haya sumas, multiplicaciones, asignaciones, etc.... Como, por ejemplo, hacemos en `Devolver n + Sumatorio(n-1)` que es $1 + T(n-1)$

$$\textcircled{9} \quad T(n) = 1 + \max \{ 1, 1 + T(n-1) \}$$

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

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

$$X^n - X^{n-1} = 2$$

$$\text{Homogénea} \Rightarrow X^{(h)} \Rightarrow X^n - X^{n-1} = 0$$

$$X^{n-1} (X - 1) = 0$$

$$\rightarrow X = 1 \quad X^{(h)} = A$$

$$\text{Particular} \Rightarrow X^{(p)} = B \cdot n$$

$$\text{General} \Rightarrow X^{(g)} = X^{(h)} + X^{(p)}$$

$$X^{(g)} = A + B \cdot n$$

El peor caso de esta ecuación es $B \cdot n$. Por tanto, la complejidad es $\boxed{O(n)}$