Algoritmia y Complejidad

Entrega ejercicios 1

Grado en Ingeniería Informática

Raúl López Llana
Grupo Laboratorio: Miércoles 12.00 – 14.00 h.

Índice

Ejercicio 3

```
1 fun Calculo(x,y,z: entero) dev valor:entero
2 var i,j,t: entero
3         valor = 0
4         Desde i=x hasta y Hacer valor=valor+i fdesde
5         si(valor/(x+y)) <= 1 entonces
6                 Devolver z
7         si no
8                 t=x+((y-x)/2)
9                 Desde i=x hasta y Hacer
10                        Desde j=(3*x) hasta (3*y) Hacer
11                                valor = vavlor+Minimo(i,j)
12                        fdesde
13                fdesde
14                valor = valor+4*Calculo(t,y,valor)
15                Devolver valor
16        fsi
17 ffun
```

Análisis de la eficiencia

Para calcular la eficiencia de este código, se analizará individualmente las lineas y al final se sumaran todas para formar T(n)

Calculo de T(n):

Línea 3: 1

Línea 4: Bucle que se repite desde 1 hasta n y

dentro del bucle: 1

Línea 5: 1

Línea 6: 1

Línea 8: 1

Línea 9: Bucle que se repite desde x hasta y

Línea 10: Bucle que se repite desde 3*x hasta 3*y

Línea 11: 1

Línea 14: Recursividad, tenemos una llamada recursiva donde tenemos como parámetros de entrada t,y,valor donde t =(x+((y-x)/2))

Línea 15: 1

Tras sumar todas las lineas, nos da T(n):

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

Explicación de la recursividad reflejada en la línea 14 del algoritmo:

Yo voy a considerar la siguiente igualdad:

n=y-x

En base a esta igualdad, puedo deducir lo siguiente.

t =(x+((y-x)/2))

t =(x+(n/2))

t=(n/2) (primera mitad)

En conclusión, podemos decir que resulta un T(n/2)

Cálculo de soluciones y complejidad:

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

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

$$T(n) = 2 + n + 2 + T(n/2) + n^2$$

$$T(n) = 4 + n + T(n/2) + n^2$$

$$n = 2^k$$

$$T(2^k) - T\left(\frac{2^k}{2}\right) = 4 + 2^k + (2^k)^2$$

$$T(2^k) - T(2^{k-1}) = 4 + 2^k + 4^k$$

$$x^k = T(2^k)$$

$$x^k - x^{k-1} = 4 + 2^k + 4^k$$

Homogenea

$$x^k - x^{k-1} = 0$$

Saco factor común

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

Raiz $x = 1$

$$x^{(h)} = A$$

Particular

$$x^k - x^{k-1} = 4 + 2^k + 4^k$$

Raices

$x = 2$

$x = 4$

$x = 1$

$$x^{(p)} = B\cdot k + C\cdot 2^k + D\cdot 4^k$$

$$x = A + Bk + C\cdot 2^k + D\cdot 4^k$$

$$2^k = n$$

$$k = \log_2 n$$

$$x = A + B\cdot\log_2 n + C\cdot n + D\, n^2 \Rightarrow O(n^2)$$

Ejercicio 5

```
1 fun es_primo?(num: entero) dev resul:bool
2 aux:entero
3         si (num<=0) entonces
4                 resul = false
5         sino
6                 resul=false
7                 aux=2
8                 mientras (aux<num) y (resul==false) entonces
9                         si (num%aux==0)entonces
10                                resul=true
11                        fsi
12                        aux=aux+1
13                fmientras
14        Devolver resul
15 ffun
```

Análisis de la eficiencia

Para calcular la eficiencia de este código, se analizará individualmente las lineas y al final se sumaran todas para formar T(n)

Calculo de T(n):
Línea 3: 1

Línea 4: 1

Línea 6: 1

Línea 7: 1

Línea 8: Bucle mientras resul sea false y aux sea menor que num

Línea 9: 1

Línea 10: 1

Línea 12: 1

Línea 14: 1

Tras sumar todas las lineas, nos da T(n):

$$T(n) = 1 + max\left(1, 1 + 1 + \sum (1 + max(1, 0) + 1) + 1\right)$$

Cálculo de soluciones y complejidad:

$$T(n) = 1 + max\left(1, 3 + \sum (2 + max(1, 0))\right)$$

$$T(n) = 1 + max\left(1, 3 + \sum (3)\right)$$

$$T(n) = 1 + max(1, 3 + 3n)$$

$$T(n) = 4 + 3n$$

$$O(n)$$

Ejercicio 6

```
1 fun perfecto(numero:entero)dev resul:bool
2         suma:entero
3         resu:bool
4         suma=0
5         Desde i=1 hasta numero HACER
6                 si numero%i==0 entonces
7                         suma=suma+i
8                 fsi
9         fDesde
10        si numero==suma entonces
11                resul=true
12        sino
13                resul=false
14        fsi
15        Devolver resul
16 ffunc
```

Análisis de la eficiencia

Para calcular la eficiencia de este código, se analizará individualmente las lineas y al final se sumaran todas para formar T(n)

Calculo de T(n):

Línea 4: 1
Línea 5: Bucle desde i=1 hasta numero
    Línea 6:1
    Línea 7:1
Línea 10: si
    Línea 11:1
    Línea 13:1
Línea 15:1

Tras sumar todas las lineas, nos da T(n):

$$T(n) = 1 + \sum (1 + max(1,0)) + 1 + max(1,1) + 1$$

Cálculo de soluciones y complejidad:

$$T(n) = 1 + \sum (1 + max(1,0)) + 1 + max(1,1) + 1$$

$$T(n) = 3 + \sum (1 + 1) + max(1,1)$$

$$T(n) = 3 + 2n + 1$$

$$T(n) = 4 + 2n$$

$$O(n)$$

Ejercicio 7

```
1 fun primo_Perfecto()
2         numero:entero
3         numero=lecturaUsuario
4         primos:entero
5         primos=0
6         desde i=1 hasta numero hacer
7                 si(es_primo?(i))entonces
8                         primos=primos+1
9                 fsi
10        fdesde
11        perfectos:entero
12        perfectos=0
13        desde i=1 hasta numero hacer
14                si(perfecto(i))entonces
15                        perfectos=perfectos+1
16                fsi
17        fdesde
18        mensaje_pantalla("primos: "+primos)
19        mensaje_pantalla("perfectos: "+perfectos)
20 ffunc
```

Comentarios del código

Línea 3: leo un número introducido por el usuario.

Línea 6: bucle desde 1 hasta el numero introducido en el que miro cuanto números primos existen en ese rango, para eso hago uso de la función creada y explicada en ejercicios anteriores.

Línea 13: bucle desde 1 hasta el numero introducido en el que miro cuanto números perfectos existen en ese rango, para eso hago uso de la función creada y explicada en ejercicios anteriores.

Línea 18: muestro los resultados por pantalla.

Análisis de la eficiencia

Para calcular la eficiencia de este código, se analizará individualmente las lineas y al final se sumaran todas para formar T(n)

Calculo de T(n):

Línea 3: 1
Línea 5: 1

Línea 3: Bucle desde 1 hasta numero
    Línea 7: 1
    Línea 8: Llamada a una función definida en apartados anteriores
Línea 12: 1
Línea 13: Bucle desde 1 hasta numero
    Línea 14: 1
    Línea 15: Llamada a una función definida en apartados anteriores
Línea 18: 1
Línea 19: 1

Tras sumar todas las lineas, nos da T(n):

$$T(n) = 1 + 1 + \sum (max(4 + 3n), 0) + 1 + \sum (max(4 + 2n, 0)) + 1 + 1$$

Cálculo de soluciones y complejidad:

$$T(n) = 5 + \sum (4 + 3n) + \sum (4 + 2n)$$
$$T(n) = 5 + 8n + 5n^2$$

$$O\left(n^2\right)$$

Ejercicio 8

```
1 func fun_inverso(num:entero):entero
2
3 inverso,residuo:entero
4 inverso=0
5
6 residuo = num MOD 10
7 num = num / 10
8 inverso=inverso*10 + residuo
9
10 si (num==0)
11         Devolver inverso + fun_inverso(num)
12 sino
13         Devolver inverso
14 ffunc
```

Análisis de la eficiencia

Para calcular la eficiencia de este código, se analizará individualmente las lineas y al final se sumaran todas para formar T(n)

Calculo de T(n):

Línea 4: 1

Línea 6: 1

Línea 7: 1

Línea 8: 1

Línea 10: 1

Línea 11: Recursividad, tenemos una llamada recursiva donde tenemos como parámetros de entrada num =num/10

Línea 13: 1

Explicación de la recursividad reflejada en la línea 11 del algoritmo:

Considerando que num es la entrada de la función, podemos decir que n=num

En la línea 7, num=num / 10
lo que resulta n=n/10 y por lo tanto T(n/10)

Tras sumar todas las lineas, nos da T(n):

$$T(n)=1+1+1+1+max(T(n/10),1)$$

$$T(n)=4+max(T(n/10,1))$$

$$T(n)=4+T(n/10)$$

$$n=10^k$$

$$T(10^k)=4+T(10^k/10)$$

$$T(10^k)-T(10^k/10)=4$$

$$x^k =T(10^k)$$

$$x^k-x^{k-1} =4$$

Cálculo de soluciones:

Homogenea

$$x^k-x^{k-1} =0$$

Racices x=1

$$x^{(h)}=A * 1^n$$

Particular

$$x^k-x^{k-1} =4$$

Raices x=1

$$x^{(p)}=B * 1^n * n$$

Complejidad:

$$X=A+Bn$$

$$O(n)$$