Escuela Politécnica Superior

Universidad de Alcalá

Algoritmia y Complejidad

Laboratorio – Sesión 1

Laboratorio Jueves 12:00 – 14:00

Grado en Ingeniería Informática – Curso 2018/2019

Eduardo Graván Serrano – 03212337L

Marcos Barranquero Fernández – 51129104N

Ejercicio 5: Primos

Determinar si un número es primo es tan sencillo como comprobar que todos los números anteriores son coprimos con ese número:

```
def es_primo(numero):
    """ Dado un numero, devuelve true si es primo """

    # Comparo el numero con cada uno de sus anteriores excepto el 1
y él mismo.
    for i in range(2, numero-1):
        # Si el resto es 0, no es primo.
        if (numero % i == 0):
            return False
    return True
```

La complejidad será determinada por el máximo entre el peor caso y mejor caso.

Peor caso:

$$\sum_{n=2}^{n-1} \cdot (1)$$

Mejor caso: no es primo y sale antes de recorrer todo: O(n).

Max (peor caso, mejor caso) = Max(O(n),O(n)) = O(n)

Por tanto, la complejidad es de O(n).

Ejercicio 6: Perfecto

```
def es_perfecto(numero):
    """ Dado un número, devuelve true si es perfecto, es decir,
    la suma de sus divisores propios positivos es el propio número
"""

    # Suma de divisores
    suma = 0

    for i in range(1, numero):
        # Para cada divisor, lo añado a la suma
        if (numero % (i) == 0):
            suma += (i)

    # Si el nº es igual a la suma de sus divisores, devuelvo True:
    if numero == suma:
        return True
```

```
    else:
        return False
```

El bucle contiene 1 if y una operación. Además, fuera del bucle hay una comparación:

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

La complejidad es O(n).

Ejercicio 7: Primos y perfectos

```
def primos_y_perfectos(numero):
    """ Devuelve una lista de los numeros primos y perfectos
    que hay entre 1 y el numero dado"""
    # Lista de nºs entre 1 y el numero dado:
    numeros = [i for i in range(1, numero)]

    # Listas donde almacenaré los numeros que haya:
    primos = []
    perfectos = []

    for numero in numeros:
        # Primos:
        if(es_primo(numero)):
            primos.append(numero)
    # Perfectos:
        if(es_perfecto(numero)):
            perfectos.append(numero)

    return primos, perfectos
```

La complejidad viene determinada por:

$$\sum_{n=1}^{n} + 1 + 1 +$$
$$\sum_{n=1}^{n}\left(1 + \sum_{n=1}^{n}(1) + 1 + 1 + \sum_{n=1}^{n}(1) + 1\right) + 1$$

El primer sumatorio es debido al bucle usado para crear la lista de 1 a n. El segundo contiene el recorrido de todos los números, y dentro de este se encuentra la comparación de si es primo y la de si es perfecto, que como se ha visto antes resulta ser cada una O(n). Por tanto, la complejidad queda determinada como:

$$2n^2 + 5n + 2$$

(Sustituyo los sumatorios como “n”).

La complejidad queda como $O(n^2)$.

Ejercicio 3:

3) Analizar la eficiencia del siguiente código:

```
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)    { ÷ es la división entera }
      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
```

Analizando el código:

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

Montando la fórmula:

$$T(n) = 1 + \sum_{i=x}(1+1) + max$$

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

Simplificando la escala de la complejidad y sustituyendo sumatorios por “n”:

$$T(n) = 1 + n + 2 + n + n^2 + T\left(\frac{n}{2}\right)$$

Agrupamos:

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

Y utilizando el teorema maestro:

$$T(n) = \begin{cases} \Theta(n^p) & k<b^p \\ \Theta(n^p \log n) & k=b^p \\ \Theta(n^{\log_b k}) & k>b^p \end{cases}$$

n **= tamaño del problema**

k **= nº de subcasos**

n/b **= tamaño de los subcasos**

Comprobamos que k=1, b = 2, p=2, y aplicando el teorema:

$k = b^p; k = 1; b = 2; p = 2; 1 < 2^2;$ *Por tanto* $O(n^p)$ *y para nuestro problema* $O(n^2)$