

# **Ejercicios resueltos de programación 3**

## **Tema 4. Análisis algorítmico.**

El *índice* de los ejercicios será el siguiente. Se observa que solo hay cuestiones de exámenes (de los ejercicios cortos), por lo que se debería hacer hincapié en ellos:

1. Introducción teórica ..... 3
2. Cuestiones de exámenes ..... 4

## Introducción teórica:

Previo a resolver los ejercicios pondremos un poco de **teoría**, ya que estos ejercicios por norma general se resuelven de la misma manera. Si se pidiera algo distinto se explicará la teoría en el propio ejercicio.

### - Reducción por sustracción:

La ecuación de la recurrencia es la siguiente:

$$T(n) = \begin{cases} c * n^k & \text{si } 0 \leq n < b \\ a * T(n - b) + c * n^k & \text{si } n \geq b \end{cases}$$

La resolución de la ecuación de recurrencia es:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < 1 \\ \theta(n^{k+1}) & \text{si } a = 1 \\ \theta(a^{n \div b}) & \text{si } a > 1 \end{cases}$$

### - Reducción por división:

La ecuación de la recurrencia es la siguiente:

$$T(n) = \begin{cases} c * n^k & \text{si } 1 \leq n < b \\ a * T(n/b) + c * n^k & \text{si } n \geq b \end{cases}$$

La resolución de la ecuación de recurrencia es:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

siendo:

**a:** Número de llamadas recursivas.

**b:** Reducción del problema en cada llamada.

**c \* n<sup>k</sup>:** Todas aquellas operaciones que hacen falta además de las de recursividad.

## **1ª parte.** Cuestiones de exámenes:

### **Diciembre 2002 (ejercicio 2)**

Enunciado: calcular la ecuación de recurrencia y hallar el coste del siguiente algoritmo:

```
proc P (n) {  
  var i, j: enteros  
  j ← 1  
  si n ≤ 1 entonces terminar  
  si no {  
    para i ← 1 hasta 7 hacer P (n DIV 2)  
    para i ← 1 hasta 4 * n3 hacer j ← j + 1  
  }  
}
```

Respuesta: Este ejercicio está hecho por una alumna, por lo que la solución dada no se asegura que este correcta.

Se nos pide que hallemos la **ecuación de recurrencia**, que puede ser la siguiente, siguiendo los dos bucles:

$$T(n) = 7 * t(n \text{ DIV } 2) + \theta(n^3)$$

Siguiendo esta ecuación de recurrencia observamos que estamos ante una **recursión por división**. Veremos las siguientes variables:

**a:** Número de llamadas recursivas = 7

**b:** Reducción del problema en cada llamada recursiva = 2

**c \* n<sup>k</sup>:** Todas aquellas operaciones que hacen falta además de la recursividad. Tendremos que siguiendo la ecuación de recurrencia previamente escrita, nos queda:

$$c * n^k = n^3 \Rightarrow k = 3$$

Por tanto, siguiendo esta fórmula, nos quedará:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Calculamos  $a = b^k$ , que sustituyendo tenemos  $7 < 2^3$ , siendo, por tanto, el primer caso. El coste será  $\theta(n^3)$ , lo que significa que lo determinarán las llamadas externas a la recursividad.

**Septiembre 2003 (ejercicio 1) (igual al ejercicio 1 de Septiembre 2007-reserva y ejercicio 2 de Septiembre 2008-reserva)**

Enunciado: Hallar formalmente el coste de los siguientes algoritmos siendo  $h(n, r, k) \in O(n)$ .

Procedimiento uno (n,k: entero)

VAR i, r: entero;

COMIENZO

SI  $n < 2$  ENTONCES DEVOLVER 1;

SI NO

COMIENZO

$r \leftarrow \text{uno}(n \text{ DIV } 2, k - 1);$

$r \leftarrow r + \text{uno}(n \text{ DIV } 2, k + 1);$

$r \leftarrow r * \text{uno}(n \text{ DIV } 2, k + 2);$

DEVOLVER r;

FIN

FIN

procedimiento dos (n,k: entero)

VAR i, r: entero;

COMIENZO

SI  $n < 2$  ENTONCES DEVOLVER 1;

SI NO

COMIENZO

$r \leftarrow \text{dos}(n \text{ DIV } 2, k - 1);$

$r \leftarrow r + \text{dos}(n \text{ DIV } 2, k + 1);$

PARA  $i \leftarrow 1$  HASTA  $n/2$  HACER

COMIENZO

$r \leftarrow h(n, r, i);$

$r \leftarrow r + h(n, r - 1, i)$

FIN

$r \leftarrow r + \text{dos}(n \text{ DIV } 2, k + 2);$

DEVOLVER r;

FIN

FIN

Respuesta: Este ejercicio lo veremos en más ocasiones y se hace de la misma manera. Pasamos a ver las distintas funciones:

Función uno: Resolvemos la **reducción por división** como vimos previamente:

La ecuación de la recurrencia es la siguiente:

$$T(n) = \begin{cases} c * n^k & \text{si } 1 \leq n < b \\ a * T(n/b) + c * n^k & \text{si } n \geq b \end{cases}$$

La resolución de la ecuación de recurrencia es:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Como es habitual en estos tipos de ejercicios las variables que tenemos son:

**a**: Número de llamadas recursivas = 3

**b**: Reducción del problema en cada llamada = 2

**$c * n^k$** : Coste de las llamadas externas a la recursividad. Las llamadas extras son constantes, por lo que  $c * n^k = 1 \Rightarrow k = 0$ .

Por tanto, estaremos en el caso tercero  $a > b^k$ , siendo, por tanto, el coste  $\theta(n^{\log_2 3})$ .

Función dos: Como previamente hemos dicho será también una recursión con **reducción por división**, con estos datos:

**a**: Número de llamadas recursivas = 3

**b**: Reducción del problema en cada llamada = 2

**c \* n<sup>k</sup>**: Coste de las llamadas externas a la recursividad. Por existir un bucle "para" con coste lineal y la función  $h(n, r, k)$  con coste línea también, tal y como vimos en el ejercicio anterior, tendremos que el coste de las llamadas externas es  $n^2$ . Por tanto,  $c * n^k = n^2 \Rightarrow k = 2$ .

El coste será el correspondiente con el caso primero  $a < b^k$ ,  $\theta(n^2)$ , siendo estas llamadas externas más costosas que las recursivas, como antes, por lo que determinarán el coste total.

### Febrero 2004 -2ª (ejercicio 1)

Enunciado: ¿En qué orden está el tiempo de ejecución del siguiente algoritmo? Justifica tu respuesta.

```

procedimiento  $h(n, i, j)$ 
  si  $n > 0$  entonces
     $h(n - 1, i, 6 - i - j)$ ;
    escribir  $i \rightarrow j$ ;
     $h(n - 1, 6 - i - j, j)$ ;
  fsi
  
```

Respuesta: Tendremos la siguiente ecuación de recurrencia que resuelve este algoritmo:

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

Como es habitual en estos tipos de ejercicios deducimos las siguientes variables:

**a**: Número de llamadas recursivas = 2

**b**: Reducción del problema en cada llamada recursiva = 1

**c \* n<sup>k</sup>**: Todas aquellas operaciones que hacen falta además de la recursividad. Tendremos que  $k = 0$ , por ser el tiempo extra constante.

Aplicando la recurrencia de **reducción por sustracción** siguiendo esto:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < 1 \\ \theta(n^{k+1}) & \text{si } a = 1 \\ \theta(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

Tendremos que  $a = 2 > 1$ , por tanto, aplicando la tercera fórmula, el coste del algoritmo es:

$$T(n) \in \theta(a^{n \text{ div } b}) = \theta(2^n).$$

## Febrero 2005 -2ª (ejercicio 1)

Enunciado: Demuestra cuál es el orden exacto de complejidad en función de la variable  $n$ :

```
procedimiento lona (n: entero, j: entero)
  para i desde 1 hasta  $n \text{ div } 4$  hacer
     $j := j + 1$ ;
  fpara
  si  $n > 1$  entonces
    lona( $n - 2$ ,  $j$ );
    escribir "j";
    lona( $n - 2$ ,  $n - j$ );
  fsi
fprocedimiento
```

Respuesta: En este caso, tendremos unas llamadas recursivas que corresponden con una **recursión por sustracción**. Se necesitará averiguar, por tanto, qué variable determina nuestro problema, pero nos lo dicen en el enunciado, que es  $n$ . Nuestra ecuación de recurrencia será:

$$T(n) = \begin{cases} c * n^k & \text{si } 0 \leq n < b \\ a * T(n - b) + c * n^k & \text{si } n \geq b \end{cases}$$

Los datos que tendremos a partir del algoritmo son:

**a** : Número de llamadas recursivas = 2

**b**: Reducción del subproblema en cada llamada = 2

**$c * n^k$** : Las operaciones externas a las llamadas recursivas. En este caso, tendremos el bucle:

```
para i desde 1 hasta  $n \text{ div } 4$  hacer
   $j := j + 1$ ;
fpara
```

Al tener un bucle desde 1 hasta  $n \text{ div } 4$  de una operación elemental  $j := j + 1$  tendremos que están en el orden de  $O(n)$ , entonces para averiguar el valor de  $k$  tendremos:

$$c * n^k = n \Rightarrow k = 1.$$

Aplicando la siguiente fórmula, tendremos:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < 1 \\ \theta(n^{k+1}) & \text{si } a = 1 \\ \theta(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

Estaremos en este caso en la tercera opción, por lo que aplicando lo anterior tendremos:

$$T(n) \in \theta(2^{n \text{ div } 2})$$

### Septiembre 2005-reserva (ejercicio 3)

Enunciado: Analizar y hallar el coste de los algoritmos siguientes (Considerar de orden  $O(n^2)$  la función  $h(n, r, k)$ ):

Procedimiento a (n,k: entero)

VAR i, r: entero;

COMIENZO

SI  $n < 3$  ENTONCES DEVOLVER (1);

SI NO

COMIENZO

$r \leftarrow a(n \text{ DIV } 2, k - 1)$ ;

$r \leftarrow r + a(n \text{ DIV } 2, k + 1)$ ;

PARA  $i \leftarrow 1$  HASTA  $n^2$  HACER

COMIENZO

$r \leftarrow r + k$ ;

FIN

DEVOLVER (r);

FIN

FIN

procedimiento b (n,k: entero)

VAR i, r: entero;

COMIENZO

SI  $n < 4$  ENTONCES DEVOLVER (1);

SI NO

COMIENZO

$r \leftarrow b(n \text{ DIV } 2, k + 1)$ ;

PARA  $i \leftarrow 1$  HASTA  $n/2$  HACER

COMIENZO

$r \leftarrow h(n, r, i)$ ;

$r \leftarrow r + h(n, r - 1, i)$

FIN

$r \leftarrow r + b(n \text{ DIV } 2, k + 2)$ ;

DEVOLVER (r);

FIN

FIN

Respuesta:

Este ejercicio se haría de igual manera a los demás, por lo que realmente es copiarlo y pegarlo modificando los datos nuevos que se nos den.

Función "a": Nos damos cuenta que  $n$  es el que indica el tamaño del problema, que determinará nuestro problema. Por lo anteriormente expuesto tendremos esta ecuación de **recursión por división** con la que resolveremos este algoritmo:

$$T(n) = \begin{cases} c * n^k & \text{si } 1 \leq n < b \\ a * T(n/b) + c * n^k & \text{si } n \geq b \end{cases}$$

Por este motivo, tendremos estas variables, que deduciremos del algoritmo:

**a**: Número de llamadas recursivas = 2

**b**: Reducción del problema en cada llamada = 2, que corresponde con la división de nuestro tamaño del problema ( $n$ ).

**$c * n^k$** : Coste de las llamadas externas a la recursividad. Para ello observamos que existe este bucle:

```
PARA i ← 1 HASTA n² HACER
  COMIENZO
    r ← r + k;
  FIN
```

}  $O(n^2)$

Explicamos lo que hemos puesto más arriba. Tendremos, por tanto, un bucle desde 1 hasta  $n^2$ , por lo que el coste es  $O(n^2)$ , siendo, por tanto,  $k = 2$ .

La resolución de la **recursión por división** será:



$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Con esta información, ya estamos dispuestos a resolver la recurrencia, entonces, tendremos que saber si  $a < b^k$ ,  $a = b^k$  ó  $a > b^k$ . Por lo que, como antes hicimos

$$2 < 2^2.$$

Estamos en el **caso primero**, por lo que el coste del algoritmo es  $\theta(n^2)$ , lo que significa que realmente el que determina el coste total es el de las llamadas externas a la recursividad, por tener un coste elevado, como hemos visto en otros ejercicios anteriores.

Función "b": Nuestra variable que determina el coste es  $n$  igualmente. A continuación, vemos que es una **recursión por división**, con la siguiente ecuación de recurrencia:

$$T(n) = \begin{cases} c * n^k & \text{si } 1 \leq n < b \\ a * T(n/b) + c * n^k & \text{si } n \geq b \end{cases}$$

De nuevo, tenemos que averiguar cuáles son las variables:

**a**: Número de llamadas recursivas = 2.

**b**: Reducción del problema en cada llamada recursiva = 2.

**$c * n^k$** : Coste de las operaciones externas a la recursividad. Seguimos el mismo planteamiento que antes y vemos que el bucle "para" es el siguiente:

$$\left. \begin{array}{l} \text{PARA } i \leftarrow 1 \text{ HASTA } n/2 \text{ HACER} \\ \quad \text{COMIENZO} \\ \quad \quad r \leftarrow h(n, r, i); \\ \quad \quad r \leftarrow r + h(n, r - 1, i); \\ \quad \text{FIN} \end{array} \right\} O(n^2) \left\} O(n * n^2) = O(n^3)$$

Una aclaración con respecto a este ejercicio es que consideramos el coste del bucle "para" sin considerar las constantes multiplicativas, es decir, en el caso del primer paréntesis tendríamos realmente coste  $2 * n^2$ , solo que vemos que tomaremos la cota superior dicho coste. Pasaría, por tanto, el mismo resultado que con el paréntesis exterior, que descartaremos la constante multiplicativa (1/2) para quedarnos con el cota superior. Dicho esto, haremos este mismo procedimiento en ejercicios sucesivos.

En este caso, tendríamos  $k = 3$ , razonando de igual manera que la función  $a$ .

Pasamos a calcular el coste de la recurrencia con esta fórmula (aunque esté en la misma página, repito que hay que machacarla mucho)

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Tendremos, por tanto, que  $2 < 2^3$ , siendo el primer caso y llegando a la conclusión del coste  $\theta(n^3)$ .

### Septiembre 2006-reserva (ejercicio 1)

Enunciado: Hallar el coste de los siguientes algoritmos siendo  $h(n, r, k) \in O(n^2)$ .

Procedimiento uno (n,k: entero)

```
VAR i, r: entero;
COMIENZO
  SI n < 6 ENTONCES DEVOLVER 1;
SI NO
  COMIENZO
    r ← uno(n DIV 6, k - 1);
    r ← r + uno(n DIV 6, k + 1);
    r ← r * uno(n DIV 6, k + 2);
    PARA i ← 1 HASTA 2 * n HACER
      COMIENZO
        r ← h(n, r, i) + h(n, r - 1, i);
      FIN
    DEVOLVER r;
  FIN
FIN
```

procedimiento dos (n,k: entero)

```
VAR i, r: entero;
COMIENZO
  SI n < 8 ENTONCES DEVOLVER 1;
SI NO
  COMIENZO
    r ← dos(n DIV 8, k - 1);
    r ← r + dos(n DIV 8, k + 1);
    PARA i ← 1 HASTA n/2 HACER
      COMIENZO
        r ← h(n, r, i);
        r ← r + h(n, r - 1, i);
      FIN
    r ← r + dos(n DIV 8, k + 2);
    DEVOLVER r;
  FIN
FIN
```

Respuesta:

Función uno: Tendremos una recursividad que hay que resolver, tal y como vimos en el ejercicio anterior. Nos damos cuenta que n es el que indica el tamaño del problema, por lo que lo que determine nuestro problema. Por lo anteriormente expuesto tendremos esta fórmula de **recursión por división** con la que resolveremos este algoritmo:

$$T(n) = \begin{cases} c * n^k & \text{si } 1 \leq n < b \\ a * T(n/b) + c * n^k & \text{si } n \geq b \end{cases}$$

Por este motivo, tendremos estas variables, que deduciremos de dicho algoritmo:

**a**: Número de llamadas recursivas = 3

**b**: Reducción del problema en cada llamada = 6, que corresponde con la división de nuestro tamaño del problema (n).

**c \* n<sup>k</sup>**: Coste de las llamadas externas a la recursividad. Para ello observamos que existe este bucle:

$$\left. \begin{array}{l} \text{PARA } i \leftarrow 1 \text{ HASTA } 2 * n \text{ HACER} \\ \quad \text{COMIENZO} \\ \quad \quad r \leftarrow h(n, r, i) + h(n, r - 1, i); \\ \quad \quad \text{FIN} \end{array} \right\} O(n^2) \left. \vphantom{\begin{array}{l} \text{PARA } i \leftarrow 1 \text{ HASTA } 2 * n \text{ HACER} \\ \quad \text{COMIENZO} \\ \quad \quad r \leftarrow h(n, r, i) + h(n, r - 1, i); \\ \quad \quad \text{FIN} \end{array}} \right\} O(n * n^2) = O(n^3)$$

Explicamos lo que hemos puesto más arriba. Tendremos, por tanto, un bucle desde 1 hasta  $2 * n$ , por lo que, evidentemente el coste del mismo es  $O(n)$ . En el enunciado se nos da que  $h(n, r, k) \in O(n^2)$ , por lo que el coste de las llamadas externas es  $O(n^3)$ .

Resolviendo  $c * n^k = n^3$ , vemos que  $k = 3$ .

La fórmula para hallar el coste de la **recursión por división** será:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Con esta información, ya estamos dispuestos a resolver la recurrencia, entonces, tendremos que saber si  $a < b^k$ ,  $a = b^k$  ó  $a > b^k$ . Por lo que, como antes hicimos

$$3 < 6^3.$$

Estamos en el caso primero, por lo que el coste del algoritmo es  $\theta(n^3)$ , lo que significa que realmente el que determina el coste total es el de las llamadas externas a la recursividad.

Función dos: Según vemos en esta función el planteamiento es el mismo que en el anterior. Nos preguntamos antes de nada sobre qué variable determina el tamaño del problema, que es  $n$ , como antes. A continuación, vemos que es una **recursión por división**, con el siguiente esquema:

$$T(n) = \begin{cases} c * n^k & \text{si } 1 \leq n < b \\ a * T(n/b) + c * n^k & \text{si } n \geq b \end{cases}$$

De nuevo, tenemos que averiguar cuáles son las variables:

**a**: Número de llamadas recursivas = 3. Este número tiene truco, ya que debajo del bucle "para" hay una llamada recursiva, que, por supuesto, se tiene en cuenta.

**b**: Reducción del problema en cada llamada recursiva = 8.

**$c * n^k$** : Coste de las operaciones externas a la recursividad. Seguimos el mismo planteamiento que antes y vemos que el bucle "para" es el siguiente:

$$\left. \begin{array}{l} \text{PARA } i \leftarrow 1 \text{ HASTA } n/2 \text{ HACER} \\ \quad \text{COMIENZO} \\ \quad \quad r \leftarrow h(n, r, i); \\ \quad \quad r \leftarrow r + h(n, r - 1, i); \\ \quad \quad \text{FIN} \end{array} \right\} O(n^2) \left. \vphantom{\begin{array}{l} \text{PARA } i \leftarrow 1 \text{ HASTA } n/2 \text{ HACER} \\ \quad \text{COMIENZO} \\ \quad \quad r \leftarrow h(n, r, i); \\ \quad \quad r \leftarrow r + h(n, r - 1, i); \\ \quad \quad \text{FIN} \end{array}} \right\} O(n * n^2) = O(n^3)$$

Llegamos a la misma conclusión que la función uno, en la que teníamos que  $k = 3$ .  
 Pasamos a calcular el coste de la recurrencia con esta fórmula (aunque esté en la misma página, repito que hay que machacarla mucho)

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Tendremos, por tanto, que  $3 < 8^3$ , siendo el primer caso y llegando a la conclusión anterior  $\theta(n^3)$ .

### Septiembre 2006 (ejercicio 1)

Enunciado: Demuestra que el tiempo de ejecución en función de  $n$  del siguiente fragmento de código está acotado superiormente por  $O(n^2)$  e inferiormente por  $\Omega(n)$ . Demuestra también su orden exacto de complejidad

```
para i desde 1 hasta n hacer
    para j desde 1 hasta n div i hacer
        escribir "i, j, k"
    fpara
fpara
```

Supón que el coste de "escribir" es constante.

Respuesta: En primer lugar debemos plantear el tiempo en función de  $n$ . Para ello, vamos a fijarnos en la **instrucción barómetro**, que por definición es aquella que se ejecuta por lo menos con tanta frecuencia como cualquier otra instrucción del algoritmo. En este caso, escogeremos la instrucción más interna, que es *escribir "i, j, k"*.

A continuación, tendremos que contar cuántas veces se ejecuta, ya que es la que determinará el coste del algoritmo entero. En este caso, asumimos que la instrucción barómetro tiene coste constante, recordemos que es  $O(1)$ .

El tiempo  $T(n)$  que tenemos será:

$$T(n) = \sum_{i=1}^n \sum_{j=1}^{n \text{ div } i} 1 = \sum_{i=1}^n \frac{n}{i} = n * \sum_{i=1}^n \frac{1}{i}.$$

Para calcular la cota superior tendremos:

$$\sum_{i=1}^n \sum_{j=1}^{n \text{ div } i} 1 \leq \sum_{i=1}^n \sum_{i=1}^n 1 = \sum_{i=1}^n n = n^2. \quad T(n) \in O(n^2)$$

Esto significa que está acotado superiormente por  $n^2$ . Modificando uno de los sumatorios, hemos visto que  $n \geq n \text{ div } i$  (o  $n/i$ ), por lo que hemos conseguido acotarlo.

Para la cota inferior tendremos:

$$\sum_{i=1}^n \sum_{j=1}^{n \text{ div } i} 1 \geq \sum_{i=1}^n \sum_{i=1}^1 1 = \sum_{i=1}^n 1 = n. \quad T(n) \in \Omega(n)$$

Al igual que antes, hemos modificado el sumatorio para verificar que la cota inferior es  $n$ , en este caso,  $1 \leq n \text{ div } i$ .

Se nos pide en la segunda parte del ejercicio que tenemos que demostrar el orden exacto. Por tanto, necesitamos saber cómo crece la serie  $\sum_{i=1}^n \frac{1}{i}$ . Para ello, podemos aproximar la serie con la integral

$$\int_1^n \frac{1}{x} dx$$

Esta integral es el **logaritmo natural** de  $n$ . Por tanto, la serie crece tan rápidamente como el  $\log(n)$ . Así pues, se pueden encontrar dos constantes  $c$  y  $d$ , tal que  $T(n)$  esté acotado superiormente por  $c * n * \log(n)$  e inferiormente por  $d * n * \log(n)$ . En conclusión, el orden exacto es:

$$T(n) \in \theta(n * \log(n)).$$

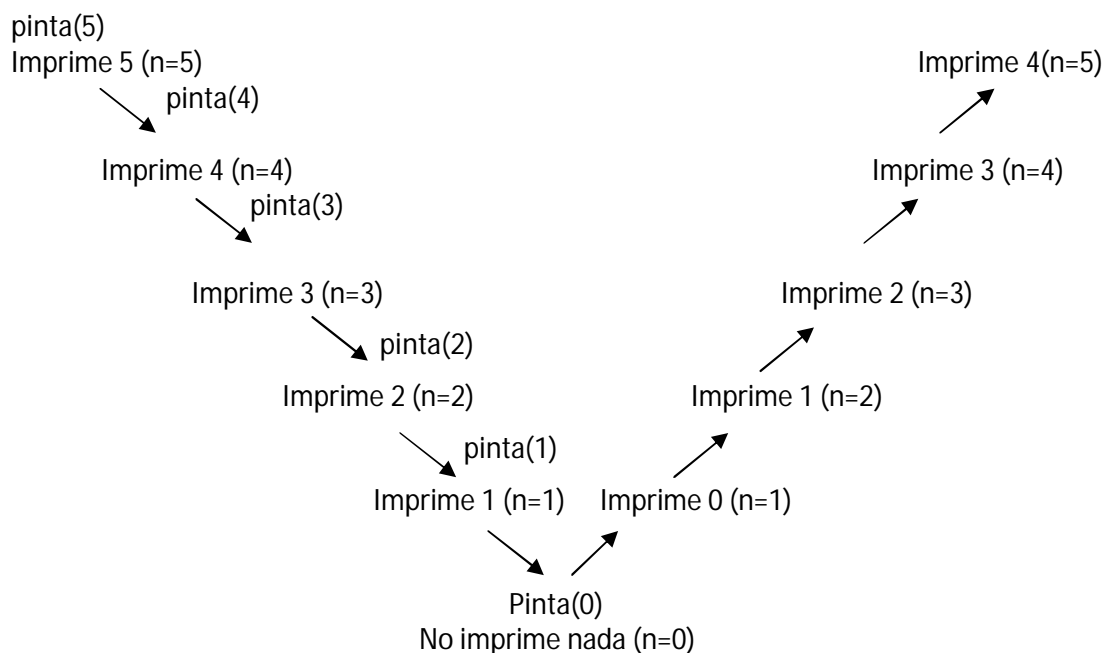
## Septiembre 2006 (ejercicio 2)

Enunciado: Escribe la salida al realizar la llamada "pinta (5)", dado el siguiente código:

```
funcion pinta (int n)
  si n > 0 entonces
    escribir "n";
    pinta (n-1);
    escribir "n - 1";
  fsi
ffuncion
```

Demuestra el coste computacional de la función "pinta" suponiendo que "escribir" tiene coste constante.

Resultado: Para escribir la salida al realizar la llamada "pinta (5)" tendremos que tener muy claros los conceptos de **pila**, ya que al llamar a la recursividad apila y al dejar la llamada (finalizarlo) desapila. Por lo que, apilará hasta llegar a  $n = 0$ , en este caso, empezará a desapilar. Pondremos un dibujo para que se vea más claro:



Para resolver la recurrencia tendremos que hacerlo usando la **reducción por sustracción**, como hemos visto antes, siendo, por tanto, la ecuación de recurrencia y la resolución las siguientes:

$$T(n) = \begin{cases} c * n^k & \text{si } 0 \leq n < b \\ a * T(n - b) + c * n^k & \text{si } n \geq b \end{cases}$$

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < 1 \\ \theta(n^{k+1}) & \text{si } a = 1 \\ \theta(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

En este caso, tendremos:

**a:** número de llamadas recursivas = 1

**b:** Reducción del problema en cada llamada = 1

**$c * n^k$ :** Coste de las operaciones externas a la recursividad. Tendremos que todas las operaciones son de coste constante, por lo que:

$$c * n^k = 1 \Rightarrow k = 0$$

Vemos que es el caso segundo, por ser  $a = 1$ . Por lo que el coste es  $\theta(n)$ .

### Diciembre 2006 (ejercicio 1)

Enunciado: Cuenta el número de operaciones elementales que efectúa el siguiente fragmento de código  $f$  en los casos mejor y peor, y a partir de estos resultados da su orden de complejidad,  $O(f(n))$ , y su orden exacto  $\theta(f(n))$ .

```
(1)      para i desde 1 hasta n hacer
(2)          para j desde n hasta i + 1 dec -1 hacer
(3)              si  $a[j - 1] > a[j]$  entonces
(4)                   $temp = a[j - 1]$ ;
(5)                   $a[j - 1] = a[j]$ ;
(6)                   $a[j] = temp$ 
(7)          fsi
(8)      fpara
(9)      fpara
```

Respuesta: Este ejercicio se asemeja mucho a los hechos en el tema 2 en la primera parte, por lo que lo dejaremos sin hacer salvo tener en cuenta los distintos tiempos que existen, recordemos que eran tiempo de acceso a vector, tiempo de resta, etc etc. Por otro lado, la segunda parte del ejercicio es propiamente de este tema, por lo que lo hemos añadido por ese motivo. Por eso, no resolveremos el ejercicio aunque sí que diremos algunas pistas claves para ello.

### Febrero 2007-2ª (ejercicio 1)

Enunciado: ¿Cuál es el tamaño del problema,  $n$ , que determina el orden de complejidad del siguiente procedimiento? Calcula el orden de complejidad del algoritmo en función de dicho tamaño.

```
tipo vector = array de enteros;
procedimiento examen(var a: vector; prim, ult, x: entero): boolean;
var
    mitad: entero;

(1)  si ( $prim \geq ult$ ) entonces
(2)      devolver  $a[ult] = x$ 
(3)  si no
(4)       $mitad = (prim + ult) \div 2$ ;
(5)      si ( $x = a[mitad]$ ) entonces
(6)          devolver cierto;
(7)      si no si ( $x < a[mitad]$ ) entonces
(8)          devolver examen(a,prim,mitad-1,x)
(9)      si no
(10)         devolver examen(a,mitad+1,ult,x);
(11)  fsi
(12) fsi
```

Respuesta:

El tamaño del problema será  $n = (ult - prim) + 1$ , correspondiente con la condición del bucle "if" ( $prim \geq ult$ ).

En cuanto al análisis del orden de complejidad tendremos que realizar estos pasos:

1. **Análisis de su funcionamiento**, en la que lo simularemos.
2. **Análisis del coste** propiamente dicho.

En cuanto al análisis del funcionamiento, sólo decir que el objetivo de la búsqueda binaria es hallar un elemento  $x$  de un vector  $a[1..n]$ , el cual está ordenado de modo no decreciente (sólo podremos aplicar la búsqueda binaria si este vector está ordenado así).

Se nos dan tres casos:

- Elemento  $x$  está a la izquierda del elemento mitad (usaremos la variable que se llama *mitad* en el algoritmo).

- Elemento x coincide con la mitad del vector.
- Elemento x está a la derecha de la mitad del vector.

En cuanto al análisis del coste, tenemos que si se cumple la condición de la línea (1) el coste es constante, recordemos  $O(1)$ . A continuación, si nos fuéramos por la línea (3), tendríamos estas derivaciones, es decir, puede ser cualquiera de estas opciones:

1. Por línea (6), en cuyo caso el coste es constante,  $O(1)$ .
2. Por línea (8), en cuyo caso es coste  $t(n/2) + \text{cte}$ .
3. Por línea (10), en cuyo caso es coste  $t(n/2) + \text{cte}$ .

Planteamos la recurrencia con **reducción por división**:

$$T(n) = \begin{cases} c * n^k & \text{si } 1 \leq n < b \\ a * T(n/b) + c * n^k & \text{si } n \geq b \end{cases}$$

Tendremos:

- a**: Número de llamadas recursivas = 1, ya que hay una recursividad por cada derivación.  
**b**: Reducción del problema en cada llamada = 2, que corresponde con la división del problema la realizar la búsqueda.  
 **$c * n^k$** : Coste de las llamadas externas a la recursividad. Encontramos que el resto de operaciones es constante, es decir, se asumen que son operaciones elementales:  
 $c * n^k = 1 \Rightarrow k = 0$

Con estos datos pasamos a resolver la recurrencia:

$$T(n) = \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k * \log(n)) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

En este caso,  $a = b^k$ , cuyo coste es  $\theta(n^0 * \log(n)) = \theta(\log(n))$ .