# PROGRAMACIÓN 2. Curso 2017-18. Grupo de tarde

## 2ª prueba voluntaria de evaluación

Esta es la segunda prueba voluntaria que se plantea en la asignatura ***Programación 2*** para la evaluación de los alumnos matriculados en el grupo de tarde. Tiene un valor de **10 puntos** y debe ser resuelta individualmente.

### Primer problema [5.0 puntos]

El diseño de la función genérica **`organizar`**`(v,n)` se presenta a continuación. Se han etiquetado las líneas de su código con las primeras letras del alfabeto (**`a`**, **`b`**, ..., **`i`** y **`j`**). Se asume que el coste de ejecutar el código de cada una de dichas líneas es constante, es decir $\boldsymbol{O}(1)$, y va a ser denominado $t_a$, $t_b$, ..., $t_i$ y $t_j$, respectivamente.

```
/*
 * Pre: n > 0
 * Post: No es necesario conocer este predicado para resolver el problema
 */
template <typename Dato>
void organizar (Dato v[], const int n) {
    for (int i = n - 2; i >= 0; --i) {                    // a
        Dato d = v[i];                                    // b
        int iLimite = n - 1;                              // c
        int j = i;                                        // d
        while (j != iLimite) {                            // e
            if (d > v[j+1]) {                             // f
                v[j] = v[j+1];                            // g
                j = j + 1;                                // h
            }
            else {
                iLimite = j;                              // i
            }
        }
        v[j] = d;                                         // j
    }
}
```

Se pide analizar el coste en tiempo de ejecutar **`organizar`**`(v,n)` en función del valor que tome el parámetro **`n`**, según el siguiente guión:

1. Explicar y justificar cuáles son las disposiciones iniciales de los **`n`** primeros elementos del vector **`v`** que proporcionan los casos asintóticamente peores y mejores en cuanto al coste en tiempo de **`organizar`**`(v,n)` o, en su caso, justificar si no cabe distinguir casos diferenciados ya que hay un único caso general para caracterizar asintóticamente el coste en tiempo.

2. Determinar la función de coste en tiempo, $t(n)$, en los casos extremos identificados en el apartado anterior o, si ha lugar, en el caso único general. Dicha función o funciones de coste se expresarán en función de los costes en tiempo $t_a$, $t_b$, ..., $t_i$ y $t_j$ de las diferentes líneas del algoritmo. Se deberán detallar y, en su caso, justificar los diferentes pasos dados para calcular la función o funciones de coste pedidas.

3. Caracterizar asintóticamente el orden de cada una de las funciones de coste en tiempo, $\boldsymbol{O}(t(n))$, determinadas en el apartado anterior.

## Segundo problema [5.0 puntos]

Sea $t_{peor}(n)$ la función que determina el coste en tiempo necesario para ejecutar una invocación **elevarRec**(base,n,1,1.0,base) a la función cuyo diseño se presenta más adelante, en los casos peores que puedan presentarse. Y sea $t_{mejor}(n)$ la función que determina el coste en tiempo necesario para ejecutar una invocación **elevarRec**(base,n,1,1.0,base) a la misma función en los casos mejores que puedan presentarse.

En este problema se pide realizar un análisis del coste en tiempo de una invocación **elevarRec**(base,n,1,1.0,base) para valores de **n** suficientemente grandes que responda con claridad y detalle las siguientes preguntas:

1. Justificar por qué las funciones de coste en tiempo $t_{peor}(n)$ y $t_{mejor}(n)$ de una invocación **elevarRec**(base,n,1,1.0,base) dependen del valor del argumento **n** y no dependen del valor de los restantes argumentos.

2. Explicar y justificar para qué valores del argumento **n** se presentan los casos con costes en tiempo asintóticamente peores y mejores de una invocación **elevarRec**(base,n,1,1.0,base).

3. Asumiremos que el coste de ejecutar el código de cada una de las líneas del código de la función es constante es igual a $t_a$ (el coste del código de la línea **a**), a $t_b$ (el coste del código de la línea **b**), a $t_c$ (el coste del código de la línea **c**), a $t_d$ (el coste del código de la línea **d**) y a a $t_e$ (el coste del código de la línea **e**). Se pide determinar la función de coste en tiempo $t_{peor}(n)$ de una invocación **elevarRec**(base,n,1,1.0,base) detallando los cálculos realizados y su justificación.

4. De un modo similar, se pide determinar la función de coste en tiempo $t_{mejor}(n)$ de una invocación **elevarRec**(base,n,1,1.0,base) detallando los cálculos realizados y su justificación.

5. Explicar cuál es la diferencia, si es que existe, entre las funciones de coste de la invocación **elevarRec**(base,n,1,1.0,base) en los casos peores y mejores.

```
/*
 * Pre: i >= 0 AND x^N = r * (x^i)^e  AND pot = x^e
 * Post: elevarRec(x,i,e,r,pot) = x^N
 */
double elevarRec (const double x, const int i, const int e, const double r, const double pot) {
    if (i != 0) {                                                          // a
        if (i % 2 != 0) {                                                  // b
            return elevarRec(x, i / 2, 2 * e, pot * r, pot * pot);         // c
        }
        else {
            return elevarRec(x, i / 2, 2 * e, r, pot * pot);               // d
        }
    }
    else {
        return r;                                                          // e
    }
}
```

## Modo de entrega de la prueba

Cada alumno preparará un documento en papel, manuscrito o impreso, con la resolución de los dos problemas anteriores. El documento será entregado al profesor de la asignatura (al final de clase o en su despacho) con fecha límite el **viernes 4 de mayo de 2018**. No se admitirán trabajos presentados de otro modo, ni entregas fuera de plazo, ni trabajos plagiados total o parcialmente.