PRG - ETSInf. TEORÍA. Curso 2015-16. Parcial 1.
11 de abril de 2016. Duración: 2 horas.

1. $\boxed{\text{4 puntos}}$ Dado un array `a` de `int` ordenado ascendentemente y un entero `x`, escribir un método **recursivo** que calcule la suma de los elementos menores que `x` que existen en `a`. Por ejemplo, si `a`=`{2,2,7,8,10,10,12,23,34}` y `x`=10 debe devolver 19; para el mismo array y `x`=1 debe devolver 0. El método debe evitar hacer comparaciones superfluas de elementos de `a` con `x`.

   Se pide:

   a) (0.75 puntos) Perfil del método, con los parámetros adecuados para resolver recursivamente el problema.

   b) (1.25 puntos) Caso base y caso general.

   c) (1.5 puntos) Implementación en Java.

   d) (0.5 puntos) Llamada inicial para que se realice el cálculo sobre todo el array.

Solución:

a) Una posible solución consiste en definir un método con el siguiente perfil:

```
/** Precondición: a ordenado ascendentemente y 0 <= pos. */
public static int sumaMenores(int[] a, int pos, int x)
```

de modo que devuelva la suma de los elementos de `a[pos..a.length-1]` menores que `x`, siendo `0`≤`pos`.

b)
- Caso base, `pos`≥`a.length`: Subarray vacío. Devuelve 0.
- Caso general, `pos<a.length`: Subarray de uno o más elementos. Si `a[pos]` es menor que `x`, devuelve la suma de `a[pos]` más la suma de los elementos menores que `x` en `a[pos+1..a.length-1]`; pero si `a[pos]`≥`x`, y al estar el array ordenado ascendentemente, se devuelve 0 sin necesidad de hacer más comprobaciones.

c)
```
/** Devuelve la suma de los elementos de a[pos..a.length-1] menores que x.
 *  Precondición: a ordenado ascendentemente y 0 <= pos. */
public static int sumaMenores(int[] a, int pos, int x) {
    if (pos < a.length) {
        if (a[pos] >= x) { return 0; }
        else { return a[pos] + sumaMenores(a, pos + 1, x); }
    } else { return 0; }
}
```

d) Para un array `a`, la llamada `sumaMenores(a, 0, x)` resuelve el problema del enunciado.

2. $\boxed{\text{3 puntos}}$ El siguiente método comprueba si un array de enteros se encuentra ordenado de manera ascendente entre las posiciones `ini` y `fin` inclusive:

```
/** Devuelve true si a[ini..fin] está ordenado ascendentemente,
 *  false en caso contrario.*/
public static boolean ordenado(int[] a, int ini, int fin) {
    if (ini >= fin) { return true; }
    else {
        if (a[ini] > a[ini + 1] || a[fin] < a[fin - 1]) { return false; }
        else { return ordenado(a, ini + 1, fin - 1); }
    }
}
```

   Se pide:

   a) (0.25 puntos) Indicar cuál es el tamaño o talla del problema, así como la expresión que la representa.

b) (0.5 puntos) Indicar si existen diferentes instancias significativas para el coste temporal del algoritmo e identificarlas si es el caso.

c) (1.5 puntos) Dar la relación de recurrencia para el coste, y resolverla por sustitución, distinguiendo el coste de las instancias más significativas en caso de haberlas.

d) (0.75 puntos) Expresar el resultado anterior utilizando notación asintótica.

Solución:

a) La talla del problema es el número de elementos del array `a` en consideración, y la expresión que la representa es `fin-ini+1`. De ahora en adelante, denominaremos a este número `n`. Esto es, `n = fin-ini+1`.

b) Se trata de un problema de búsqueda (ascendente y descendentemente, de forma simultánea) y, por lo tanto, para una misma talla presenta instancias distintas. El caso mejor es cuando la primera o la última pareja de elementos en consideración no están ordenadas. El caso peor es cuando `a[ini..fin]` está ordenado.

c) Para obtener el coste del método, estudiamos cada una de las dos instancias significativas:

- Caso mejor: $T^m(n) = 1$ *p.p.*
- Caso peor:

$$T^p(n) = \begin{cases} T^p(n-2)+1 & \text{si } n > 1 \\ 1 & \text{si } n \leq 1 \end{cases}$$

expresado en pasos de programa (p.p.).
Resolviendo por sustitución: $T^p(n) = T^p(n-2)+1 = T^p(n-4)+2 = \ldots = T^p(n-2\cdot i)+i$. Se llega al caso base, talla 0 o 1, para $i = n/2$. Con lo que

$$T^p(n) = 1 + \frac{n}{2} \ p.p.$$

d) En notación asintótica: $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(n)$. Por lo tanto, $T(n) \in \Omega(1)$ y $T(n) \in O(n)$, es decir, el coste temporal depende como mucho linealmente de la talla del problema.

3. $\boxed{\text{3 puntos}}$ El siguiente método transpone una matriz cuadrada:

```
/** Cambia la matriz m a su transpuesta. Precondición: m es una matriz cuadrada. */
public static void transpuesta(int[][] m) {
    for (int i = 0; i < m.length; i++) {
        for (int j = 0; j < i; j++) {
            int aux = m[i][j];
            m[i][j] = m[j][i];
            m[j][i] = aux;
        }
    }
}
```

Se pide:

a) (0.25 puntos) Indicar cuál es el tamaño o talla del problema, así como la expresión que la representa.

b) (0.5 puntos) Indicar si existen diferentes instancias significativas para el coste temporal del algoritmo e identificarlas si es el caso.

c) (1.5 puntos) Elegir una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y de acuerdo con ella obtener una expresión matemática, lo más precisa posible, del coste temporal del método, distinguiendo el coste de las instancias más significativas en caso de haberlas.

d) (0.75 puntos) Expresar el resultado anterior utilizando notación asintótica.

Solución:

a) La talla del problema es la dimensión de la matriz `m`, es decir, `m.length`. De ahora en adelante, denominaremos a este número `n`. Esto es, `n`=`m.length`.

b) No existen diferentes instancias.

c) Eligiendo como unidad de medida el paso de programa, se tiene

$$T(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i-1} 1) = 1 + \sum_{i=0}^{n-1}(1+i) = 1 + \frac{(1+n)\cdot n}{2} = 1 + \frac{n}{2} + \frac{n^2}{2} \; p.p.$$

Si se toma como instrucción crítica la evaluación de la guarda `j<i`, y se cuenta como medida del coste el número de veces que se ejecuta dicha guarda, se llega a la expresión:

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(1+i) = \frac{n}{2} + \frac{n^2}{2} \; i.c.$$

que es como la expresión anterior, aunque despreciando el término de menor orden.

d) En notación asintótica: $T(n) \in \Theta(n^2)$.