PRG - ETSInf. TEORÍA. Curso 2017-18. Parcial 1.
16 de abril de 2018. Duración: 2 horas.

Nota: El examen se evalúa sobre 10 puntos, pero su peso específico en la nota final de PRG es de **3 puntos**.

1. $\boxed{\text{4 puntos}}$ Dados un array `v` de `String` y un número natural `n`, escribe un método **recursivo** que devuelva cuántos elementos del array `v` tienen una longitud `n`.

   Por ejemplo, si el array `v` contuviera {`"barco", "autobus", "tren", "moto", "bici"`}, y `n` fuera 4, entonces el método devolvería 3. Si el array fuera el mismo, y `n` fuera 5, entonces el método devolvería 1. Para el mismo array `v`, y `n` igual a 3, el método devolvería 0.

   Se pide:

   a) (0.75 puntos) Perfil del método, con los parámetros adecuados para resolver recursivamente el problema, y precondición relativa a dichos parámetros.
   b) (1.25 puntos) Caso base y caso general.
   c) (1.50 puntos) Implementación en Java.
   d) (0.50 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:

```java
/** Precondición: n >= 0 && 0 <= pos <= v.length */
public static int contarPal(String[] v, int n, int pos)
```

de manera que devuelva cuántos elementos de longitud `n` hay en el array `v[pos..v.length - 1]`, siendo $0 \leq$ `pos` $\leq$ `v.length`.

b)
- Caso base, `pos = v.length`: Subarray vacío. Devuelve 0.
- Caso general, `pos` $<$ `v.length`: Subarray con uno o más elementos. Se calcula el número de elementos de longitud `n` en el subarray `v[pos + 1..v.length - 1]` y a este número se le suma 1 si la longitud de `v[pos]` es `n`. Se devuelve ese número.

c)
```java
public static int contarPal(String[] v, int n, int pos) {
    if (pos == v.length) { return 0; }
    else {
        if (v[pos].length() == n) { return 1 + contarPal(v, n, pos + 1); }
        else { return contarPal(v, n, pos + 1); }
    }
}
```

d) Para un array `v` y un número `n`, la llamada `contarPal(v, n, 0)` resuelve el problema enunciado.

2. $\boxed{\text{3 puntos}}$ Dada una matriz cuadrada `m` de enteros, cuyas componentes son todas positivas, el siguiente método comprueba para qué filas la suma de todas sus componentes es menor que `tope`. Para cada una de dichas filas, escribe el valor de la suma de sus componentes.

```java
/** Precondicion: m es una matriz cuadrada, y sus elementos
 *  son todos positivos. El valor de tope es > 0. */
public static void sumas(int[][] m, int tope) {
    int n = m.length;
    for (int i = 0; i < n; i++) {
        int suma = m[i][0];
        int j = 1;
        while (j < n && suma < tope) {
            suma += m[i][j];
            j++;
```

```
            }
            if (suma < tope) {
                System.out.println("Fila " + i + ": " + suma);
            }
        }
    }
```

Se pide:

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

b) (0.75 puntos) Indica, y justifica, si existen diferentes instancias significativas para el coste temporal del algoritmo e identifícalas si es el caso.

c) (1.50 puntos) Elige una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y de acuerdo con ella obtén 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) (0.50 puntos) Expresa el resultado anterior utilizando notación asintótica.

Solución:

a) La talla del problema es la dimensión de la matriz `m` y la expresión que la representa es `m.length`. De ahora en adelante, llamaremos a este número n. Esto es, $n =$ `m.length`.

b) Sí que existen diferentes instancias. El caso mejor se da cuando todas las filas tienen en la componente 0 un valor mayor o igual que `tope`. El caso peor se da cuando para todas las filas se cumple que la suma de todas las componentes de la fila no llega a `tope`.

c) Si elegimos como unidad de medida el paso de programa, se tiene:

- En el caso mejor: $T^m(n) = 1 + \sum_{i=0}^{n-1} 1 = 1 + n \ p.p.$
- En el caso peor: $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=1}^{n-1} 1) = 1 + \sum_{i=0}^{n-1} n = 1 + n^2 \ p.p.$

Si elegimos como unidad de medida la instrucción crítica y considerando como tal, por ejemplo, la evaluación de la guarda `j < n && suma < tope` (de coste unitario), se tiene:

- En el caso mejor $T^m(n) = \sum_{i=0}^{n-1} 1 = n \ i.c.$
- En el caso peor: $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=1}^{n} 1 = \sum_{i=0}^{n-1} n = n^2 \ i.c.$

d) En notación asintótica: $T^m(n) \in \Theta(n)$ y $T^p(n) \in \Theta(n^2)$. Por lo tanto, $T(n) \in \Omega(n)$ y $T(n) \in O(n^2)$.

3. $\boxed{\text{3 puntos}}$ Se desea calcular el coste del siguiente método recursivo:

```
public static double testMethod(double[] v, int left, int right) {
    if (left > right) { return 1.0; }
    else {
        int middle = (left + right) / 2;
        return v[middle]
            * testMethod(v, left, middle - 1)
            * testMethod(v, middle + 1, right);
    }
}
```

Se pide:

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

b) (0.75 puntos) Indica, y justifica, si existen diferentes instancias significativas para el coste temporal del algoritmo e identifícalas si es el caso.

c) (1.50 puntos) Escribe la ecuación de recurrencia del coste temporal en función de la talla para cada uno de los casos si hay más de uno, o una única ecuación si únicamente hubiera un caso. Debe resolverse por sustitución.

d) (0.50 puntos) Expresa el resultado anterior utilizando notación asintótica.

Solución:

a) La talla del problema es el número de elementos del subarray `v[right..left]`, dado por la expresión `right - left + 1` $= n$.

b) Se trata de un recorrido y, por tanto, no hay instancias significativas.

c) Planteamos la ecuación de recurrencia considerando las tallas correpondientes al caso base y al caso general. Por simplicidad, aproximamos a $n/2$ la talla de las dos llamadas del caso general.

$$T(n) = \begin{cases} 2 \cdot T(n/2) + 1 & \text{si } n > 0 \\ 1 & \text{si } n = 0 \end{cases}$$

Resolviendo por sustitución:

$T(n) = 2 \cdot T(n/2) + 1 = 2^2 \cdot T(n/2^2) + 2 + 1 = 2^3 \cdot T(n/2^3) + 4 + 2 + 1 = 2^3 \cdot T(n/2^3) + 2^3 - 1 = \ldots = 2^i \cdot T(n/2^i) + 2^i - 1$. Si $1 \leq n/2^i < 2 \rightarrow i = \lfloor log_2 n \rfloor$, con lo que $T(n) = 2^{\lfloor log_2 n \rfloor} \cdot T(n/2^{\lfloor log_2 n \rfloor}) + 2^{\lfloor log_2 n \rfloor} - 1$.

Tomando $2^{\lfloor log_2 n \rfloor} \approx n$, se tiene que $T(n) = n \cdot T(1) + n - 1 = n \cdot (2 \cdot T(0) + 1) + n - 1$. Se llega al caso base con $T(0) = 1$, con lo que $T(n) = 4n - 1$ *p.p.*

d) En notación asintótica, el coste temporal será: $T(n) \in \Theta(n)$.