PRG - ETSInf. TEORÍA. Curso 2016-17. Parcial 1.
10 de abril de 2017. 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. $\boxed{\text{4 puntos}}$ Dado un array `a` de `int` y un entero `x`, escribe un método **recursivo** que devuelva cuántos múltiplos de `x` hay en el array `a`.

   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) (1.50 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:

```
/** Precondición: 0 <= pos */
public static int multiplosX(int[] a, int x, int pos)
```

de manera que devuelva cuántos múltiplos de `x` hay en el array `a[pos..a.length - 1]`, siendo `0` $\leq$ `pos`.

b)
- Caso base, `pos` $\geq$ `a.length`: Subarray vacío. Devuelve 0.
- Caso general, `pos` $<$ `a.length`: Subarray con uno o más elementos. Si `a[pos] % x == 0`, devuelve 1 más el número de múltiplos de `x` en `a[pos + 1..a.length - 1]`; si no, devuelve el número de múltiplos de `x` en `a[pos + 1..a.length - 1]`.

c)
```
/** Devuelve el número de múltiplos de x en a[pos..a.length - 1].
 *  Precondición: 0 <= pos */
public static int multiplosX(int[] a, int x, int pos) {
    if (pos >= a.length) { return 0; }
    else if (a[pos] % x == 0) { return 1 + multiplosX(a, x, pos + 1); }
    else { return multiplosX(a, x, pos + 1); }
}
```

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

2. $\boxed{\text{3 puntos}}$ Dado un array de caracteres `a` y un carácter `c` cualquiera, el siguiente método escribe en la salida estándar, línea a línea, todos los prefijos de la secuencia de caracteres en `a`, de longitud `1` en adelante, que no acaben en el carácter `c`.

```
public static void prefijos(char[] a, char c) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] != c) {
            for (int j = 0; j <= i; j++) {
                System.out.print(a[j]);
            }
            System.out.println();
        }
    }
}
```

Por ejemplo, si `a = {'g', 't', 'a', 't', 'c'}`, los prefijos de longitudes sucesivas son `g`, `gt`, `gta`, `gtat` y `gtatc`. Para `a` y `c = 't'`, el método escribe:

```
g
gta
gtatc
```

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 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 el número de elementos del array `a` y la expresión que la representa es `a.length`. De ahora en adelante, llamaremos a este número n. Esto es, $n =$ `a.length`.

b) Sí que existen diferentes instancias. El caso mejor se da cuando todos los caracteres del array `a` son el carácter `c`. El caso peor se da cuando todos son distintos del carácter `c`.

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

- En el caso mejor: $T^m(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1\ p.p.$
- En el caso peor: $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i} 1) = 1 + \sum_{i=0}^{n-1}(2 + i) = 1 + 2n + \sum_{i=0}^{n-1} i = 1 + n + \frac{n(n+1)}{2} = 1 + \frac{3n}{2} + \frac{n^2}{2}\ p.p.$

Si elegimos como unidad de medida la instrucción crítica y considerando como tal:

- la condición `a[i] != c` de la instrucción `if` (de coste unitario), en el caso mejor se tiene: $T^m(n) = \sum_{i=0}^{n-1} 1 = n\ i.c.$
- la instrucción del cuerpo del bucle interno `System.out.print(a[j])` (de coste unitario), en el caso peor se tiene: $T^p(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(1 + i) = n + \sum_{i=0}^{n-1} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}\ i.c.$

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. $\boxed{\text{3 puntos}}$ El siguiente método determina si, dado un número entero no negativo `num`, su literal puede estar expresado en una base determinada `b` ($2 \leq$ `b` ≤ 10). Para ello comprueba que todos los dígitos del número tengan un valor estrictamente menor que la base `b`. Por ejemplo, el número 453123 puede representar un valor en base 6, 7, 8, 9 y 10 ya que todos sus dígitos son estrictamente inferiores a los valores de estas posibles bases.

```
/** Precondición: 2 <= b <= 10 y num >= 0 */
public static boolean basePosible(int num, int b) {
    if (num == 0) { return true; }
    else {
        int ultDig = num % 10;
        if (ultDig < b) { return basePosible(num / 10, b); }
        else { return false; }
    }
}
```

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 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 puede ser:

(1) el valor del primer argumento del método, esto es, `num`; llamaremos a este número m.

(2) el número de cifras de `num`; llamaremos a este número n.

b) Sí que hay instancias significativas, ya que se trata de una búsqueda. En el mejor caso, la cifra de las unidades de `num` es mayor o igual que la base `b` y en el caso peor todas las cifras de `num` son menores que `b`, esto es, `num` se puede representar en base `b`.

c) Planteamos la ecuación de recurrencia para cada una de las instancias significativas, en pasos de programa, considerando cada una de las tallas posibles.

(1) Para la talla m (valor de `num`) se obtiene:

- En el caso mejor: $T^m(m) = 1$ *p.p.*
- En el caso peor:

$$T^p(m) = \begin{cases} T^p(m/10) + 1 & \text{si } m > 0 \\ 1 & \text{si } m = 0 \end{cases}$$

Resolviendo por sustitución:
$T^p(m) = T^p(m/10) + 1 = T^p(m/10^2) + 2 = \ldots = T^p(m/10^i) + i$. Si $1 \leq m/10^i < 10 \rightarrow i = \lfloor log_{10} m \rfloor$, con lo que $T^p(m) = T^p(m/10^{\lfloor log_{10} m \rfloor}) + \lfloor log_{10} m \rfloor = T^p(0) + 1 + \lfloor log_{10} m \rfloor$. Se llega al caso base en el que $T^p(0) = 1$. Con lo que $T^p(m) = 2 + \lfloor log_{10} m \rfloor$ *p.p.*

(2) Para la talla n (número de cifras de `num`) se obtiene:

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

$$T^p(n) = \begin{cases} T^p(n-1) + 1 & \text{si } n > 0 \\ 1 & \text{si } n = 0 \end{cases}$$

Resolviendo por sustitución:
$T^p(n) = T^p(n-1) + 1 = T^p(n-2) + 2 = \ldots = T^p(n-i) + i$. Se llega al caso base (talla 0) cuando $n - i = 0 \rightarrow i = n$. Con lo que $T^p(n) = 1 + n$ *p.p.*

d) En notación asintótica:

(1) Para la talla m (valor de `num`), el coste temporal será: $T^m(m) \in \Theta(1)$ y $T^p(m) \in \Theta(\log_{10} m)$, esto es, las cotas para el coste son $T(m) \in \Omega(1)$ y $T(m) \in O(\log_{10} m)$.

(2) Para la talla n (número de cifras de `num`), el coste temporal será: $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(n)$, esto es, las cotas para el coste son $T(n) \in \Omega(1)$ y $T(n) \in O(n)$.

Como se puede observar, el coste temporal del método es el mismo, aunque expresado en función de tallas distintas. Recuerda que el número de cifras de un número entero m es $1 + \lfloor log_{10} m \rfloor$, es decir, n.