PRG - ETSInf. TEORÍA. Curso 2012-13. Parcial 1. Recuperación
17 de junio de 2013. Duración: 1h 50min.

1. $\boxed{\text{2.5 puntos}}$ Para determinar si cierto número entero no negativo `n` puede encontrarse expresado en una base determinada `b` ($2\leq$`b`≤10), es suficiente 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 esas posibles bases.

   Se pide: Implementar en Java un **método recursivo** que, dados `n` y `b`, resuelva el problema planteado, especificando los casos base y general de la recursión.

Solución:

```
/** Devuelve si es o no posible que n este en base b.
  * PRECONDICIÓN: 2<=b<=10 y n >=0 */
public static boolean basePosible(int n, int b) {
    if (n==0) return true;                              // caso base
    else {                                              // caso general
        int ultDig = n%10;
        if (ultDig<b) return basePosible(n/10,b);
        else return false;
    }
}
```

2. $\boxed{\text{2.0 puntos}}$ Dado un array `a` de valores reales y cierto valor real `x`, se desea determinar, **recursivamente**, el número de elementos de `a`, desde la posición `pos` incluida hasta la final (`a[pos..a.length-1]`), que tengan valor mayor que x. Para ello, partiendo del perfil:

```
public static int numMayor(double[] a, double x, int pos)
```

   Se pide:

   - Implementar el **método recursivo** pedido, especificando los casos base y general de la recursión.
   - Escribir la llamada inicial para obtener el número de elementos mayores que `x` de todo un array.

Solución:

```
/**
  * Calcula el numero de elementos con valor mayor que uno dado a partir de
  * la posicion pos.
  */
public static int numMayor(double[] a, double x, int pos) {
    if (pos>=a.length) return 0;                        // caso base
    else if(a[pos]>x) return 1+numMayor(a,x,pos+1);     // caso general
    else return numMayor(a,x,pos+1);                    // caso general
}
```

Llamada inicial: `int num = numMayor(v,x,0);` cumpliendo `v` y `x` las condiciones dadas por el perfil del método.

3. 3.0 puntos Dada cierta matriz `m` cuadrada de valores reales, el siguiente método permite determinar si la misma es triangular inferior (esto es, si todos sus elementos superiores a la diagonal principal son iguales a 0).

```
/**
  * Determina si una matriz cuadrada es triangular inferior, esto es:
  * si todos los elementos superiores a la diagonal principal valen 0.
  */
public static boolean esInferior(double[][] m) {
    boolean esInf = true;
    for(int i=0; i<m.length && esInf; i++)
        for(int j=i+1; j<m.length && esInf; j++)
            esInf = m[i][j]==0;
    return esInf;
}
```

Se pide: estudia el coste temporal del método, para lo que has de:

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

Solución: La talla del problema es $n =$ `m.length`, es decir, la dimensión de la matriz.

b) Identificar, caso de que las hubiere, las instancias del problema que representan el caso mejor y peor del algoritmo.

Solución: El método resuelve un problema de búsqueda y, por tanto, para una misma talla sí que presenta instancias distintas.
Caso mejor: Cuando `m[0][1]!=0` con lo que ambos bucles finalizarán al acabar su primera iteración. *Caso peor:* Cuando la matriz es triangular inferior y todos los elementos superiores a la diagonal principal valen 0. En ese caso, se efectúan todas las iteraciones posibles.

c) Elegir una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y acorde con ella, obtener una expresión matemática, lo más precisa posible, del coste temporal del programa, a nivel global o en las instancias más significativas si las hay.

Solución:

- Si optamos por escoger como unidad de medida el paso de programa, se obtiene:
  - *Caso mejor:* $T^m(n) = 1$ paso de programa.
  - *Caso peor:* $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=i+1}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(n - i) = n^2/2 + n/2 + 1$ pasos.
- Si optamos por escoger la instrucción crítica como unidad de medida para la estimación del coste:

- *Caso mejor:* $T^m(n) = 1$ instrucción crítica.
- *Caso peor:* $T^p(n) = \sum_{i=0}^{n-1}(\sum_{j=i+1}^{n-1} 1) = \sum_{i=0}^{n-1}(n-i-1) = n^2/2 - n/2$ instrucciones críticas.

d) Expresar el resultado anterior utilizando notación asintótica.

Solución:
En notación asintótica: $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(n^2)$. Por tanto, $T(n) \in \Omega(1)$ y $T(n) \in O(n^2)$.

4. $\boxed{\text{2.5 puntos}}$ El siguiente método recursivo comprueba si dos `String` que tienen la misma longitud son simétricas. Dos `String` son simétricas cuando el primer elemento de la primera es igual al último de la segunda y así sucesivamente.

Por ejemplo, las `String`: `"HOLA"` y `"ALOH"` son simétricas, mientras que `"HOLA"` y `"ALHA"` no lo son.

```
/**
  * Determina si a y b son o no simetricas
  * PRECONDICION: a.length()==b.length()
  */
public static boolean simetricas(String a, String b) {
    if(a.length()==0) return true;
    else {
        int ult = b.length()-1;
        return a.charAt(0)==b.charAt(ult) &&
               simetricas(a.substring(1), b.substring(0,ult));
    }
}
```

Se pide: realizar el estudio del coste temporal del método anterior, conociendo que **todas las operaciones de la clase `String`** que se aplican a alguna `String` en el algoritmo **tienen un coste constante** (esto es, su coste no depende de la longitud de la `String` a la que se aplique) para ello:

a) Indicar cuál es la talla del problema y qué expresión la define.

Solución: La talla es la longitud `n` de cada `String` (`a.length()`).

b) Determinar si existen instancias significativas. Si las hay, identificar las que representan los casos mejor y peor del algoritmo.

Solución: Sí que hay instancias significativas. El *caso mejor* se da cuando el primer y último carácter de ambas `String` no concuerdan. El *caso peor* se da cuando ambas son simétricas.

c) Escribir la ecuación de recurrencia del coste temporal en función de la talla para cada uno de los casos si hubiera varios, o una única ecuación si sólo hubiera un caso. Resuélvela por sustitución.

Solución:

- Caso mejor: $T^m(n) = 1$ pasos.
- Caso peor: Se tiene que: $T^p(n) = 1$ si $n = 0$ y $T^p(n) = T^p(n-1) + 1$ si $n > 0$.

  Resolviendo por sustitución: $T^p(n) = T^p(n-1) + 1 = T^p(n-2) + 2 = \ldots = T^p(n-i) + i = \ldots = T^p(0) + n = n + 1$ pasos.

d) Expresar el resultado anterior usando notación asintótica.

Solución:

En notación asintótica: $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(n)$. Por tanto, $T(n) \in \Omega(1)$ y $T(n) \in O(n)$.