PRG - ETSInf. PRÁCTICAS. Curso 2013-14. Parcial 1.
14 de abril de 2014. Duración: 1 hora

1. 2.5 puntos La siguiente implementación del algoritmo recursivo del problema de las torres de Hanoi se compila correctamente y no se produce ningún error en su ejecución.

```
public static void hanoi(int discos, String origen, String destino, String auxiliar) {
  if (discos==1)
     moverDisco(origen, destino);
  else {
      hanoi(discos-1, origen, auxiliar, destino);
      moverDisco(origen, destino);
      hanoi(discos-2, origen, destino, auxiliar);
  }
}
```

Sin embargo, contiene errores lógicos, es decir, el resultado de su ejecución es incorrecto con respecto al visto en prácticas.

Se pide: localizar los errores, explicarlos y corregir el código para que la solución sea la vista en prácticas.

Solución: La segunda llamada recursiva, `hanoi(discos-2, origen, destino, auxiliar)`, es incorrecta por las siguientes razones:

- Después de desplazar un disco, aún quedan `discos-1` en la torre auxiliar, no `discos-2`.
- En esa segunda llamada recursiva, la torre `auxiliar` debe hacer la función de `origen`, y la torre `origen` la función de `auxiliar`.

En consecuencia, el error se corrige cambiando la citada instrucción por la siguiente:

```
hanoi(discos-1, auxiliar, destino, origen);
```

2. 2.5 puntos Se dispone de un método `boolean esSufijo(String a, String b)` que devuelve `true` si la cadena `a` es un sufijo de `b` y `false` en caso contrario.

Se pide: implementar un método recursivo que, haciendo uso del método anterior, devuelva si una cadena dada `s` es subcadena de otra `t`. La cabecera del método deberá ser necesariamente:

```
public static boolean esSubcadena(String s, String t)
```

Recuerda que `s.substring(i,j)` es un método disponible en la librería de Java que devuelve un objeto `String` que representa la substring de `s` formada por los caracteres comprendidos entre el `i` y el `j-1`.

Solución:

```
public static boolean esSubcadena(String s, String t) {
   if (s.length()>t.length()) return false;
   else if (esSufijo(s, t)) return true;
   else return esSubcadena(s, t.substring(0, t.length()-1));
}
```

3. 2.5 puntos El método estático `ordena(int[])` de la clase `AlgoritmosMedibles` tiene, en el caso promedio, un coste cuadrático, tomando como talla el tamaño del array que se le pasa como parámetro.

   Se pide: completar el código del siguiente método para estudiar su coste empírico en el caso promedio, realizando 50 repeticiones para cada talla `t`. Se pueden utilizar los siguientes métodos:

   - `public static int[] crearArrayAleatorio(int talla)`, que devuelve un array de tamaño `talla` con sus elementos generados de forma aleatoria.
   - `public static long nanoTime()`, de la clase `java.lang.System`, que devuelve el valor actual del temporizador más preciso del sistema en nanosegundos.

```
public static void medidaOrdenacion() {
    System.out.printf("# Talla    Tiempo\n");
    System.out.printf("#----------------\n");
    long t1 = 0, t2 = 0, tt = 0; double tmedt = 0;   // Tiempos
    for (int t=10000; t<=100000; t+=10000) {

      //       COMPLETAR

      System.out.printf("%8d  %8d\n", t, tmedt/1000);
    }
}
```

Solución:

```
public static void medidaOrdenacion() {
    System.out.printf("# Talla    Tiempo\n");
    System.out.printf("#----------------\n");
    long t1 = 0, t2 = 0, tt= 0; double tmedt = 0;   // Tiempos
    for (int t=10000; t<=100000; t+=10000) {
      tt = 0;                          // Tiempo acumulado inicial a 0
      for (int r=0; r<50; r++) {
        int[] a = crearArrayAleatorio(t);
        t1 = System.nanoTime();        // Tiempo inicial
        AlgoritmosMedibles.ordena(a);
        t2 = System.nanoTime();        // Tiempo final
        tt += (t2-t1);                 // Actualizar tiempo acumulado
      }
      tmedt = (double)tt/50;           // Tiempo promedio
      System.out.printf("%8d  %8d\n", t, tmedt/1000);
    }
}
```

4. 2.5 puntos Suponiendo que los resultados del estudio empírico anterior se han almacenado en el fichero `resultados` en dos columnas (talla y tiempo), **se pide:**

   a) (1.75 puntos) Definir la función a ajustar a la curva definida por el estudio del coste teórico y realizar el ajuste de dicha función mediante la orden `fit` de Gnuplot, cuya sintaxis es la siguiente:

   `fit` *función fichero* `using` i:j `via` *parámetros*

   en donde:

   - *función*: indica el nombre de la función a ajustar.

- *fichero*: indica el nombre del fichero con los datos a ajustar (se especifica entre comillas dobles).
- `using` i`:`j: especifica las columnas del fichero de datos que se usarán (i para el eje X y j para el eje Y).
- `via` *parámetros*: especifica los parámetros (separados por comas) de la función a ajustar.

b) (0.75 puntos) Una vez realizado el ajuste, ¿cómo calcularías el tiempo del método para una talla igual a 25000?

Solución:

a) `f(x) = a*x*x + b*x + c;` `fit f(x) "resultados" using 1:2 via a,b,c`

b) Una vez conocidos `a`, `b` y `c` (resultado de haber realizado la orden `fit`), calcular `f(25000)`