Primer Parcial de Prácticas de PRG
ETSInf - Curso 2012/13

22 de abril de 2013. Duración: 1 hora

1. 2.5 puntos Considera el siguiente algoritmo visto en prácticas para resolver el problema de las "Torres de Hanoi":

```
public static void hanoi(int n, char org, char dest, char aux) {
    if (n==1) System.out.println("Mueve disco desde " + org + " a " + dest);
    else {
        hanoi(n-1,org,aux,dest);
        System.out.println("Mueve disco desde " + org + " a " + dest);
        hanoi(n-1,aux,dest,org);
    }
}
```

Contesta a las siguientes cuestiones:

- Sabiendo que para resolver el problema en el caso de una torre de 4 discos el algoritmo realizaría 15 movimientos de disco, ¿cuántos realizaría si la torre tuviese 5 discos?
- ¿Cuál debería ser la llamada inicial al algoritmo anterior si se quisiera mover una torre de 12 discos desde la aguja `'z'` hasta la `'h'` utilizando como auxiliar la `'s'`?

Solución:

- El número de movimientos para una torre de 5 discos es dos veces el número de movimientos para una torre de 4 discos más un movimiento adicional, esto es: 31 movimientos.
- La llamada inicial sería: `hanoi(12,'z','h','s');`

2. 2.5 puntos Indica qué modificación realizarías en el siguiente código para que el método `subcadena(String, String)` fuera correcto. Se supone que el método `prefijo(String,String)` ya está implementado y funciona correctamente.

```
/** Devuelve cierto sii a es subcadena de b.
 *  @param a. String.
 *  @param b. La otra String.
 *  @return boolean: cierto sii a es subcadena de b. */
public static boolean subcadena(String a, String b) {
    if (a.length()<=b.length())
        return prefijo(a,b) || subcadena(a,b.substring(1));
}
```

NOTA: recuerda que `s.substring(int)` devuelve una nueva `String` con los caracteres de `s` desde la posición que se recibe como argumento hasta el final de `s`.

Solución:

Falta el caso base, de forma que si la condición inicial no se cumple tendría que devolver `false`. Esto es, escrito correctamente sería:

```
public static boolean subcadena(String a, String b) {
    if (a.length()<=b.length())
        return prefijo(a,b) || subcadena(a,b.substring(1));
    else return false;
}
```

3. 2.5 puntos Se ha realizado una aproximación con la orden `fit` de los tiempos medidos para un algoritmo, cuya talla es el número de elementos de un array, y los resultados obtenidos son los que se muestran a continuación:

```
gnuplot> f(x)=a*x+b
gnuplot> fit f(x) "algoritmo.out" using 1:2 via a,b
...
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 0.599844         +/- 0.005358     (0.8932%)
b               = 1.00006          +/- 3324         (3.324e+05%)
```

A partir de estos resultados, realiza una predicción del tiempo de ejecución (en nanosegundos) requerido para un array de 10^8 elementos.

Solución:

Para un array de 10^8 elementos, el tiempo requerido para la ejecuión sería de $10^8 \cdot 0{,}599844 + 1{,}00006 \approx 59984401$ nanosegundos.

4. 2.5 puntos Se ha medido el tiempo de ejecución del algoritmo de inserción directa para el caso promedio mediante las siguientes instrucciones:

```
public static void medidaInsercion() {
    int[] a;          // Array del problema
    int t, r;         // Contadores de talla y repeticiones
    long tmed1, tmed2, tmedt;   // Tiempo
    // Imprimir cabecera de resultados para Inserción Directa
    System.out.printf("# INSERCIÓN DIRECTA \n");
    System.out.printf("#  Talla    Promedio (en microsegundos)\n");
    System.out.printf("#--------------------\n");
    // Repetir el proceso para distintas tallas
    for(t=INITALLA; t<=MAXTALLA; t+=INCRTALLA) {
        // Crear y rellenar el array con valores aleatorios
        a = new int[t];
        arrayAleatorio(a);
        tmedt = 0;                          // Tiempo acumulado inicial a 0
        for(r=0; r<REPETICIONES; r++) {
           tmed1 = System.nanoTime();       // Tiempo inicial
           AlgoritmosMedibles.insercion(a);
           tmed2 = System.nanoTime();       // Tiempo final
           tmedt+=(tmed2-tmed1);            // Actualizar tiempo acumulado
        }
        tmedt = tmedt/REPETICIONES;         // Tiempo promedio del caso promedio
```

```
            // Imprimir resultados
            System.out.printf("%8d  %10.3f\n",t,tmedt/Math.pow(10,3));
        }
    }
```

Obteniéndose la tabla de tiempos siguiente:

```
# INSERCIÓN DIRECTA
#  Talla    Promedio (en microsegundos)
#--------------------
   10000      44,247
   20000      80,929
   30000     123,136
   40000     166,019
   50000     209,868
   60000     254,777
   70000     300,706
   80000     347,435
   90000     395,428
  100000     443,917
```

A la vista de estos resultados, ¿son los esperados para el caso promedio? ¿Hay algún error en el código anterior? ¿Cómo lo corregirías?

Solución:

Los resultados de tiempo obtenidos muestran un crecimiento lineal cuando tendría que ser cuadrático. El error está en dónde se hace la inicialización del array `a` ya que, para una talla `t` dada, después de la primera llamada al método `insercion` el array ya está ordenado ascendentemente (caso mejor de la inserción directa). La llamada `arrayAleatorio(a);` tendría que ser la primera instrucción del bucle `for(r=0; r<REPETICIONES; r++)` para que en cada repetición el array se inicializara con valores aleatorios.