CALCULO DEL TIEMPO DE EJECUCIÓN

El objeto del siguiente seudocódigo es aclarar una posible forma de realizar las mediciones de tiempos de ejecución de los algoritmos de ordenación.

Los algoritmos son pequeños, así como los conjuntos de datos sobre los que se ejecutan, y, a la vez, las computadoras normales que utilizamos son muy rápidas.

Además de ello, trabajando en JAVA solamente contamos con dos funciones que nos ayudan para la medición de tiempos: “system.nanotime” y “system.currenttimemillis”.

En cualquier caso, la verdadera resolución de estas funciones depende de muchos factores, en particular del sistema operativo (ver documentación disponible sobre este tema). Esta resolución es algo sobre lo que no tenemos control (no la podemos cambiar).

Asumamos en principio que la más precisa sea “system.nanotime”, y que su resolución sea de 10 milisegundos.

Introduzcamos ahora el concepto de “error de medición”: si queremos que nuestra medida tenga un error no mayor a un cierto “delta”, entonces la precisión de la medición deberá ser mejor que ese delta. Por ejemplo, si deseamos que el error sea menor o igual al 1% de la medida, entonces nuestro máximo error aceptado podrá ser sólo 1/100 del total del tiempo medido.

Como indicamos anteriormente, nuestra limitante es la resolución de nuestra herramienta de medición. Si, como asumimos, ésta tiene una resolución de 10 ms, y queremos que el error de la medida sea menor o igual a 1%, entonces el total de la medición deberá ser al menos de 10e-3 x 1e2 = 1 segundo.

O sea que necesitamos ejecutar el algoritmo por lo menos durante 1 segundo para poder medir con la resolución deseada.

Pero nuestros algoritmos, sobre todo con los conjuntos de datos pequeños, ordenarán el conjunto en un tiempo mucho menor que 1 segundo.

La solución estaría entonces en ejecutar el mismo proceso, en ***igualdad de condiciones,*** tantas veces como sea necesario para superar la marca de 1 segundo.

Del total deberíamos descontar luego todo el código ejecutado adicional (los bucles de ejecución del algoritmo, la disposición repetida de los vectores de entrada, las llamadas en sí misma), pues no son parte real de cada algoritmo en sí mismo, sino simplemente algo que no tenemos más remedio que agregar para poder hacer la medición.

Entonces nos conviene medir también cuánto tiempo insumen estas “cáscaras”, y descontarlas del total.

```
vectorOriginal = generarVector(T, tipoOrden)
// con el generador de datos aleatorios, para el tamaño T, en orden "tipoOrden"
//ascendente, descendente o aleatorio)

t1 = obtenerTiempoEnNanoSegundos  //system.nanotime()
total = 0
cantLLamadas = 0

Mientras (total < tiemporesolucion) hacer
// cuidado con las unidades que  retornan las funciones
  CantLlamadas <- CantLLamadas +1

  datosCopia = copiarVector(vectorOriginal)
 // tenemos que trabajar siempre con los mismos datos

  clasificador.clasificar(datosCopia, M, TRUE) // EJECUTA EL MÉTODO
  t2 = obtenerTiempoEnNanoSegundos
  total = t2 - t1

Fin Mientras

TiempoMedioAlgoritmobase = total/CantLlamadas
// lo que lleva ejecutar 1 vez el algoritmo, para ese conjunto de datos

// ahora tenemos que calcular cuánto se fue en las "cáscaras" y descontarlo

vectorOriginal = generarVector(T, tipoOrden)
// con el generador de datos aleatorios, para el tamaño T, en orden "tipoOrden"
//ascendente, descendente o aleatorio)

t1 = obtenerTiempoEnNanoSegundos  //system.nanotime()
total = 0
cantLLamadas = 0

Mientras (total < tiemporesolucion) hacer
// cuidado con las unidades que  retornan las funciones
  CantLlamadas <- CantLLamadas +1
  datosCopia = copiarVector(vectorOriginal)
  clasificador.clasificar(datosCopia, M, FALSE)
// EJECUTA  LAS LLAMADAS AL MÉTODO ("vacias")
  t2 = obtenerTiempoEnNanoSegundos
  total = t2 - t1
Fin Mientras

TiempoMedioCascara= total/CantLlamadas
// lo que lleva ejecutar 1 vez la infraestructura del algoritmo, para ese
// conjunto de datos

TiempoMedioAlgoritmo = TiempoMedioAlgoritmoBase - TiempoMedioCascara
```