Práctica 1

Raúl Reguillo Carmona

Ampliación de Programación

Índice

1. Análisis experimental

En esta práctica se estudiarán los algoritmos *Burbuja* $\in C_1$, *Shellsort* $\in C_2$ y *Quicksort* $\in C_3$ que para la parte experimental han sido ejecutados sobre dos máquinas distintas:

- Máquina 1
  - Procesador Intel Centrino core 2 duo 2.1 GHz en cada núcleo
  - 4 GB memoria RAM DDR2 800MHz
- Máquina 2
  - Procesador Intel Pentium IV HT 3.1 GHz
  - 1 GB memoria RAM DDR 400MHz

Se han generado vectores de números enteros aleatorios comprendidos entre 0 y 9 de 10.000, 100.000 y 1.000.000 de elementos y se han medido los tiempos de ejecución del algoritmo. Para obtener una mayor precisión en la medición de la eficiencia del algoritmo se han descartado los tiempos en los que tarda en *mostrar* los vectores una vez generados y ordenados. A la hora de probar los programas bastaría con desmarcar los caracteres de comentario del código y recompilar.

Los tiempos obtenidos al realizar las ordenaciones en los distintos algoritmos han sido los siguientes:

	10K elementos	100K elementos	1M elementos
M1 - Bubble	0,8s	1m 2s	120m 8s
M2 - Bubble	1,023s	1m 27s	126m 42s
M1 - Shellsort	0,005s	0,042s	0,297s
M2 - Shellsort	0,004s	0,029s	0,318s
M1 - Quicksort	0,004s	0,039s	0,0197s
M2 - Quicksort	0,003s	0,019s	0,212s

En sucesivas ejecuciones se obtuvieron datos muy similares pudiendo afirmar que a la hora de generar los números aleatorios no se han dado casos extremos sino habituales y concordantes con los órdenes de complejidad propios de cada algoritmo, que veremos en detalle en la siguiente sección.

2. Cálculo analítico de complejidades

2.1. Bubblesort

Fuera de los bucles, la complejidad es puramente lineal, así que pasaremos directamente a analizar los anidamientos de las iteraciones:

Para cada iteración del `while` entraremos en un bucle `for`:

`for`:

n·(O(1) + O(1) + O(1)) *relativo a las operaciones de cabecera*

+ n·(O(1) + O(1)) *relativo al* `if/else`

lo que nos da O(n), repetido n veces por el while, es $O(n^2)$.

2.2. Shellsort

Puesto que la construcción de los bucles es muy similar, tan sólo debemos cambiar los datos, obteniendo el mismo resultado: $O(n^2)$

2.3. Quicksort

Entraremos en detalles para Quicksort:

Sabemos que Quicksort divide el vector y lanza recursivamente para cada sub-vector un nuevo algoritmo, esto es:

$$T(n) = 2T(n/2) + p \tag{1}$$

Donde para cada sub-vector se calcularía así:

$$T(n/2) = 2T(n/2^2) + (p/2) \tag{2}$$

Una iteración más...

$$T(n/2^2) = 2T(n/2^3) + (p/2^2) \tag{3}$$

Sustituyendo (3) en (2)

$$T(n/2) = 2^2T(n/2^3) + 2p/2^2 + p/2 \tag{4}$$

Sustituyendo (4) en (1)

$$T(n) = 2^3T(n/2^3) + 2^2p/2 + 2p/2 + p \tag{5}$$

Simplificamos ahora,

$$T(n) = 2^3 T(n/2^3) + 3p \tag{6}$$

Hasta aquí para 3 iteraciones, para k llamadas sería:

$$T(n) = 2^k T(n/2^k) + kp \tag{7}$$

Y finalmente simplificando, contando con que el número máximo de llamadas para finalizar es orden $log(n)$:

$$T(n) = nlog(n) \tag{8}$$

Lo que nos da la complejidad del algoritmo; O(n·log(n))

3. Consideraciones

Si se quieren probar los algoritmos con diferentes longitudes del vector (por defecto se generarán vectores de 10.000 posiciones y se mostrarán sus valores tanto al crearlos como al quedar ordenados) valdrá con alterar el parámetro `LVECT` del archivo `oders.h` incluido en la carpeta *include* adjunta.

Para compilar tan sólo ejecutar el `makefile` en el directorio, quedando los ejecutables en la carpeta *exec*. No es necesario pasarles parámetro alguno. No obstante ya se añaden archivos compilados.

Ejemplo:

En el directorio *Práctica 1* tipeamos en la consola

```
$ time ./exec/QS
```

obteniendo la salida correspondiente.