MEMORIA PRÁCTICA 2

Algoritmos de Ordenación

Fundamentos de Análisis de Algoritmos

Álvaro Esteban Muñoz y Francisco José Gil Durán

Grado en Ingeniería Informática (1º Curso)

Índice

0. Introducción. Algoritmo de Ordenación.

El objetivo de la práctica es realizar un análisis experimental de tres algoritmos de ordenación dados, Burbuja, Selección e Inserción. A estos métodos de ordenación se les pasará un vector de ordenado de forma aleatoria y su tamaño, dicho método se encargará de ordenarlo de menor a mayor.

1. Cálculo de los tiempos teóricos.

1.1. Pseudocódigos y análisis de coste.

Vamos a realizar un análisis teórico del coste del tiempo de cada uno de los algoritmos que estamos analizando, burbuja, selección e Inserción.

1.1.1. Algoritmo Burbuja

```
procedimiento burbuja(T[1 … n])
    para i ← n-1 hasta 1 hacer                          Bucle de n-1 hasta i
        para j ← 1 hasta i hacer                        Bucle de 1 hasta i
            si T[ j ] > T[ j +1 ] entonces              4 OE's
                Auxiliar ← T[ j ]                       1 acceso y 1 asignación
                T[ j ] ← T[ j + 1 ]                     2 accesos, 1 suma y 1 asignación
                T[ j + 1 ] ← auxiliar                   1 acceso, 1 suma y 1 asignación
            fsi
        fpara
    fpara
fprocedimiento
```

En el caso peor, la condición siempre se cumple puesto que el vector estaría ordenado de más a menos.

$$T_{(n)} = \Sigma\,(i = 1;\ n-1)\,\Sigma\,(j = 1;\ i)\,(T_{Cond} + T_{CuerpoSi})$$

$$T_{(n)} = \Sigma\,(i = 1;\ n-1)\,\Sigma\,(j = 1;\ i)\,13$$

$$T_{(n)} = \Sigma\,(i = 1;\ n-1)13i = 13\,\Sigma\,(i = 1;\ n-1)i = \mathbf{13}\,\frac{(n-1)n}{2} \in \mathbf{O}(n^2)$$

Como podemos ver, en el caso peor, la complejidad del algoritmo pertenece al $O(n^2)$, es decir, es un algoritmo lento.

En el caso mejor, tendríamos la misma situación solo que en este caso las OE's realizadas en el cuerpo de la condición son 0, solo se realizan las operaciones para comprobar si la condición es verdadera, es decir, 4 OE's.

$$\mathbf{T(n) = 4}\,\frac{(n-1)n}{2} \in \mathbf{O}(n^2)$$

En el caso medio se supone que solo se cumple la condición la mitad de las veces, sin embargo, hay que tener en cuenta que las operaciones elementales para la comprobación de la condición siempre se cumplen.

$T(n) = (4 + \frac{9}{2}) \frac{(n-1)n}{2} \epsilon O(n^2) = \mathbf{T(n) = \frac{17}{2} \frac{(n-1)n}{2} \epsilon O(n^2)}$

1.1.2. Algoritmo Inserción

```
procedimiento inserción(T[1 … n])
    para i ← 2 hasta n hacer                    Bucle de 2 hasta n
            x ← T[ i ]                          1 acceso y 1 asignación
            j ← i-1                             1 resta y 1 asignación
            mientras j > 0 AND x < T[ j ] hacer  Bucle mientras: 4 OE's Condición
                    T[ j + 1 ] ← T[ 1 ]         2 accesos, 1 suma y 1 asignación
                    j ← j - 1                   1 resta y 1 asignación
            fmientras
            T[ j + 1 ] ← x                      1 acceso, 1 suma y 1 asignación
    fpara
fprocedimiento
```

$T(n) = \Sigma(i = 2; n)(7 + Tmientras)$

$Tmientras = Tcond + Tsalto + Tcuerpo = 4 + 1 + \Sigma(j = i - 1; ?)4 + 1 + 4 + 2$

$Tmientras = 5 + \Sigma(j = i - 1; ?)11$

$T(n) = \Sigma(i = 2; n)(12 + \Sigma(j = i - 1; ?)11)$

Empezando por el caso peor otra vez, el vector vendría ordenado de mayor a menor, es decir que se realizarían n – 1 comparaciones.

$T(n) = \Sigma\ (i = 2;\ n)(12 + \Sigma\ (j = i - 1;\ 0)11) = \Sigma\ (i=2;\ n)\ (12 + 11\ \Sigma\ (j=i-1;\ 0)\ j)$

$T(n) = \Sigma\ (i = 2;\ n)\ (12 + 11\ (i - 1)) = \Sigma\ (i = 2;\ n)(11i + 1)$

$T(n) = \frac{(11n + 1)\ (n - 2 - 1 + 1)}{2} = \mathbf{T(n) = \frac{(11n + 1)\ (n - 2)}{2} \in O(n^2)}$

Sin embargo para el caso mejor, solo se hace una comparación en cada repetición del bucle para y nunca se entra en el bucle mientras, por lo tanto:

$T(n) = \Sigma\ (i = 2;\ n)(12 + \Sigma\ (j = i - 1;\ i - 1)11) = \Sigma\ (i = 2;\ n)\ 12 + 11$

$T(n) = \Sigma\ (i = 2;\ n)\ 23 = 23\Sigma\ (i = 2;\ n) = \mathbf{23(n - 1) \in O(n)}$

En este caso la complejidad es de orden n y por tanto el algoritmo es algo más rápido que el algoritmo burbuja que es siempre de orden cuadrático.

Para terminar con este algoritmo, cabe decir que puesto que el caso medio está compuesto por la media de comparaciones entre el caso peor y el mejor, el resultado es orden cuadrático también.

1.1.3. Algoritmo Selección

```
procedimiento Selección (T [1 … n])
    para i ← 1  hasta n hacer                          Bucle de 1 hasta n
    posminimo ← i                                      1 asignación
            para j ←i + 1 hasta n hacer                Bucle de i+1 hasta n
                    si T[ j ] < T[posminimo] entonces  2 accesos y 1 comparación
                            posminimo ← j              1 asignación
                    fsi
            fpara
            auxiliar ← T[posminimo]                    1 acceso y 1 asignación
            posminimo ← T[ i ]                         1 acceso y 1 asignación
            T[ i ] ← auxiliar                          1 acceso y 1 asignación
    fpara
fprocedimiento
```

Puesto que el número de comparaciones a realizar siempre es el mismo y es independiente de la situación que se de en el vector, el algoritmo será del mismo orden para los tres casos.

$T(n) = \Sigma\,(i = 1;\ n)\ (1 + \Sigma\,(j = i + 1;\ n)4 + 6)$ donde 4 es un el valor constante que se repite todas las vueltas del bucle.

$$T(n) = \Sigma\,(i = 1;\ n)\big(7 + 4\,(n - 1)\big) = \frac{(7 + 4(n - 1))n}{2} \in O(n^2)$$

1.2. Tablas y Gráficas de coste teórico.

En esta tabla podemos visualizar una evolución del número de operaciones elementales realizadas por cada algoritmo para su caso medio, todo esto son simplemente cálculos teóricos, pero a pesar de ello se puede observar claramente que algoritmo es el que realiza menos OE's y por lo tanto cual es más rápido que los demás.

Talla	Burbuja	Inserción	Selección
1000	4245750	2756238	2001500
2000	16991500	11012488	8003000
3000	38237250	24768738	18004500
4000	67983000	44024988	32006000
5000	106228750	68781238	50007500
6000	152974500	99037488	72009000
7000	208220250	134793738	98010500
8000	271966000	176049988	128012000
9000	344211750	222806238	162013500
10000	424975500	275062488	200015000

1.3. Conclusiones

Los tres algoritmos que estamos estudiando no son de los más rápido ya que ninguno rebasa el orden n, el **burbuja** es el más lento ya que siempre es de orden cuadrático, pero dependiendo de la situación del vector varía el número de comparaciones a realizar. Este algoritmo va seguido por **inserción** que en su mejor caso es orden n pero a pesar de ello en su caso medio sigue siendo más lento que el algoritmo **selección** que es el que menos operaciones elementales realiza de los tres.

2. Cálculo de los tiempos experimentales

Para realizar el cálculo experimental de los tiempos de ejecución de los algoritmos Burbuja, Inserción y Selección hemos comparado las medias de los tiempos de ejecución de los algoritmos para diez tallas diferentes.

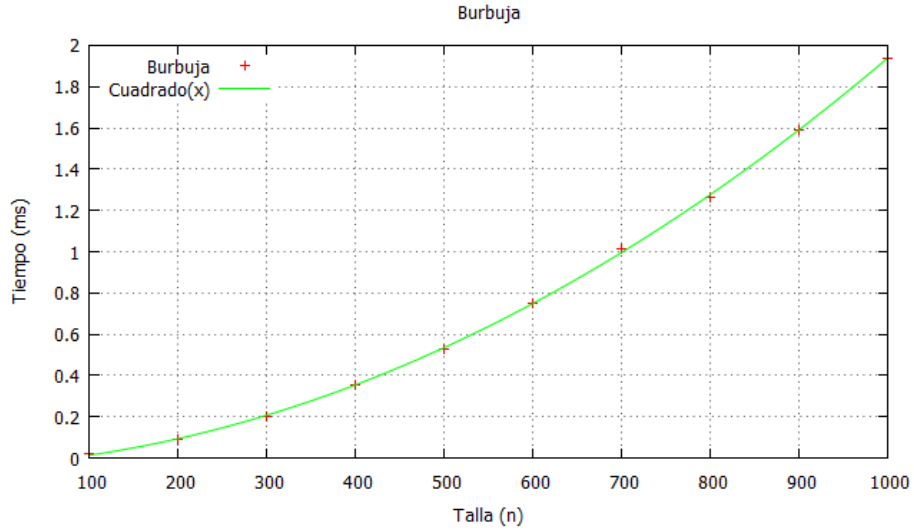
Para conocer el tiempo que ha tardado la ejecución de cada algoritmo con cada una de las tallas, hemos recurrido a la clase ***Mtime***, cuya función es restar la hora a la que finaliza la ejecución y la hora en la que comienza. Los resultados obtenidos se muestran en la tabla del apartado **2.1** .

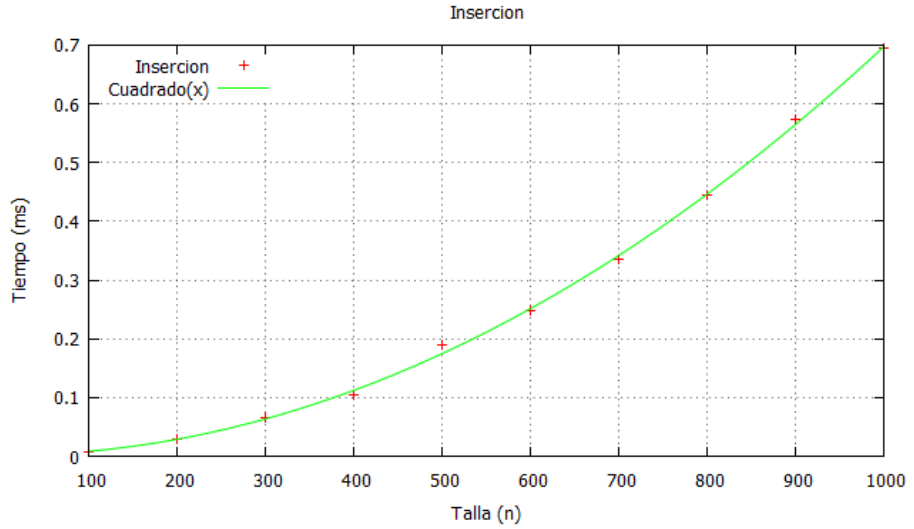
2.1. Tablas y Gráficas de coste experimental.

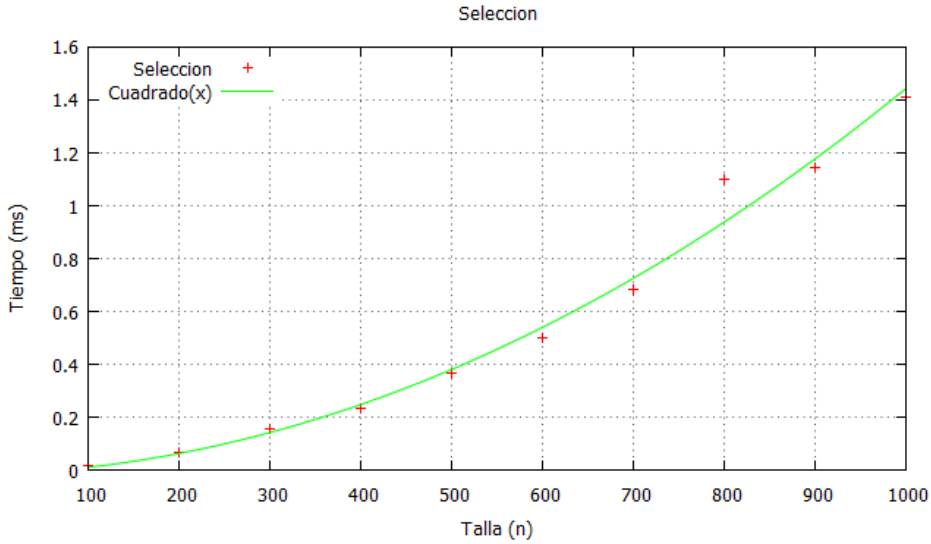
Esta tabla muestra el resultado del análisis experimental del tiempo de ejecución de los tres algoritmos de ordenación.

Talla	Burbuja	Inserción	Selección
1000	0.01985	0.00699	0.01833
2000	0.08816	0.03033	0.06451
3000	0.20423	0.06632	0.15494
4000	0.35292	0.10357	0.23099
5000	0.52997	0.18855	0.36574
6000	0.75216	0.24713	0.50112
7000	1.01182	0.33424	0.68391
8000	1.26531	0.44573	1.09696
9000	1.59212	0.57255	1.14129
10000	1.93575	0.69392	1.40921

Aquí podemos observar la gráfica de cada caso medio.







2.2. Conclusiones.

En este estudio vemos como el algoritmo **Burbuja** es el más lento de los tres sin embargo **Inserción** es bastante más rápido que **Selección.** Claramente en sus gráficas se ve como los tres siguen el orden n^2, al menos en su caso medio de eso forma podemos ver que ninguno de estos algoritmos es excesivamente rápido.

3. Comparación de los resultados teórico y experimental.

Al realizar el cálculo de los tiempos experimentales de los tres métodos, llama la atención que el método de **Inserción** es el que menos tiempo tarda en ejecutarse mientras que en el cálculo teórico nos salía que el algoritmo más rápido era **Selección** esto se debe a que los valores calculados anteriormente son valores idóneos que representan un caso ideal, sin embargo en la práctica nos damos cuenta que debido al rendimiento del PC y muchos otros factores, el algoritmo **Inserción** nos da unos resultados bastante mejores, esto puede deberse a que nuestro compilador realice optimizaciones en el código, como es nuestro caso en el programa que utilizamos para compilar que es **Visual Studio 2017.**

Cabe destacar que en este estudio coincide que el método **Burbuja** es el menos eficiente de los tres, sin embargo es el más fácil de implementar.

4. Diseño de la aplicación.

Nuestra aplicación está compuesta por 6 “.cpp” y 6 “.h”, podemos movernos a través de los diferentes apartados de la práctica a partir de un menú y varios submenús que hacen de la práctica algo muy fácil de analizar. Para analizar los casos medios hemos hecho lo mismo que en la práctica anterior, repetir el algoritmo varias veces y hacer una media del tiempo que tarda.

Cabe destacar que esta vez la clase gráficas crea/sobrescribe un fichero de comandos para la aplicación *gnuplot* en lugar de utilizar unos ficheros creados externamente, de esta forma nos ahorramos muchos ficheros de comandos y dicho fichero se adaptará dependiendo de la gráfica que queramos representar.

5. Conclusiones y valoraciones personales de la práctica.

Esta práctica nos ha servido para comprender tanto el funcionamiento de los algoritmos de ordenación como su eficiencia. Hemos podido observar como evoluciona dicha eficiencia de nuestros algoritmos dependiendo del orden de complejidad de estos y la gran cantidad de

comportamientos diferentes que pueden tener incluso teniendo el mismo orden de complejidad. También hemos aprendido a diferenciar un algoritmo lento de uno rápido.