ESTRUCTURAS DE DATOS

GRADO EN INGENIERÍA INFORMÁTICA
UGR

PRÁCTICA 1: EFICIENCIA

DAVID CARRASCO CHICHARRO

PROFESOR: CARLOS CANO GUTIÉRREZ

Ejercicios 1 y 2: Ordenación de la burbuja. Ajuste en la ordenación

Eficiencia teórica:

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-2-i} 10 \quad \rightarrow \quad \sum_{i=0}^{n-2} 10(n-i-1) \quad \rightarrow \quad \frac{10n^2}{2}-\frac{3\cdot 10n}{2}+10$$

La eficiencia teórica del algoritmo de ordenación de la burbuja es de orden $O(n^2)$

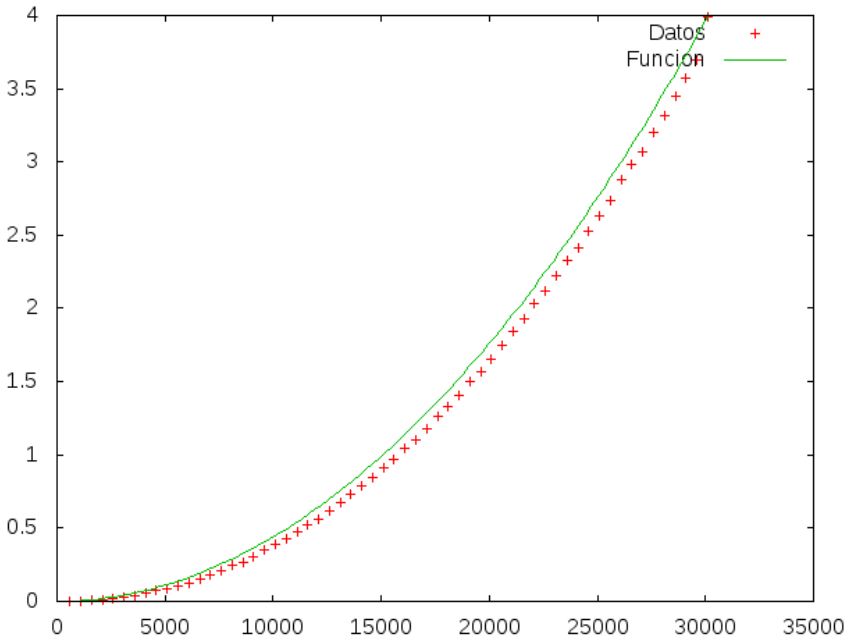
Tras ejecutar un script que ejecuta un programa que realiza la ejecución del algoritmo con tamaños de vector que varían desde los 100 elementos hasta los 30100, con incremento de 500 elementos tras cada ejecución, se ha guardado en un fichero *tiempos_ordenacion.dat* el tiempo que ha tardado en ejecutarse el programa en cada iteración.

```
Terminal type set to 'x11'
gnuplot> f(x) = a*x**2 + b*x + c
gnuplot> fit f(x) 'tiempos_ordenacion.dat' via a,b,c

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 4.56164e-09      +/- 3.981e-11    (0.8727%)
b               = -9.89285e-06     +/- 1.26e-06     (12.74%)
c               = 0.0173619        +/- 0.00839      (48.33%)
```

Ahora se procede a calcular la constante oculta para la curva del algoritmo en la eficiencia teórica. Sabemos que el tiempo que tarda cuando el vector tiene 30100 elementos es 3,99 segundos, luego:

$$3{,}99 = C\cdot 10\cdot\left(\frac{30100^2}{2}-\frac{3\cdot 30100}{2}+1\right) \quad \rightarrow \quad C = 8{,}81\cdot 10^{-10}\, seg$$

```
gnuplot> plot 'tiempos_ordenacion.dat' with p title 'Datos',8.81e-10*
 10*((x**2)/2 - (3*x)/2 +1) with l title 'Funcion'
```



Se puede comprobar que, al superponer la curva de la eficiencia teórica y la empírica, ambas son prácticamente idénticas, lo que indica que se ha hecho un buen ajuste al calcular la primera.

Ejercicios 3: Problemas de precisión

El algoritmo del fichero *ejercicio_desc.cpp* hace lo siguiente:
Realiza una búsqueda binaria de un elemento *x* en un vector cuya primera componente es la posición *inf* y la última es *sup*. Se establece una variable *med* que es la media entre *inf* y *sup*. A continuación se busca el elemento en un bucle que comprueba primeramente si en la componente *v[med]* se encuentra el valor buscado. En caso contrario, si el valor que se pretende encontrar es mayor que el que se encontraba en *v[med]*, se acota el vector a la mitad superior, o viceversa si el valor buscado es menor, de modo que en cada búsqueda se recorren la mitad de componentes que en la pasada anterior. Para ello es totalmente imprescindible que el vector esté previamente ordenado. En caso de encontrar el valor deseado, se devuelve la casilla en la que se encuentra éste; de lo contrario, se devuelve el valor -1.

Eficiencia teórica:

$$n \rightarrow n\cdot\frac{1}{2} \rightarrow n\cdot\frac{1}{2}\cdot\frac{1}{2}=n\cdot\left(\frac{1}{2}\right)^2 \rightarrow \ldots \rightarrow n\cdot\left(\frac{1}{2}\right)^n = \frac{n}{2^k} = 1$$

$$n=2^k \rightarrow \log_2(n)=\log_2(2^k) \rightarrow \log_2(n)=k$$

La eficiencia teórica de la búsqueda binaria es: $O(\log(n))$

Para calcular la eficiencia empírica se ejecuta un script que lleva a cabo el programa estableciendo distintos tamaños para el vector, desde los 1000 hasta los 200.000.000, añadiendo medio millón de componentes más al vector en cada iteración. Para acelerar el procesamiento se han introducido los números ordenados directamente en el vector. Los resultados del tiempo de ejecución de la búsqueda binaria se guardan en un fichero llamado *tiempo_des_ord.dat* para su posterior procesado.

Puesto que la medición de los tiempos es de muy baja precisión, utilizando el compilador del estándar de C++11, se ha hecho uso de la biblioteca <chrono> y las órdenes:

```
time_point<steady_clock> ini = steady_clock::now();
operacion(v, tam, tam+1, 0, tam-1);
time_point<steady_clock> fin = steady_clock::now();
duration<float,milli> tiempo = fin - ini;
```

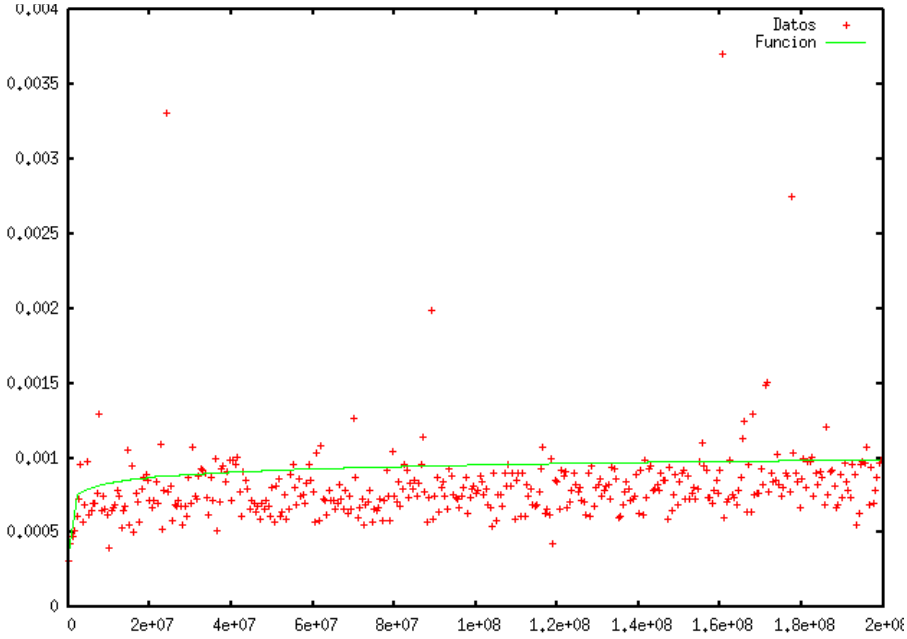
Con ello se ha conseguidos mayor precisión, pues con la biblioteca <ctime> la ejecución del algoritmo daba, en la mayoría de los casos, resultados de $2\cdot10^{-5}$ segundos.

Tras realizar el ajuste de la curva logarítmica y superponer los resultados teóricos y empíricos

```
Terminal type set to 'x11'
gnuplot> g(x) = a*( log10(x+1)/log10(2) )
gnuplot> fit g(x) 'tiempo_desc_ord.dat' via a
```

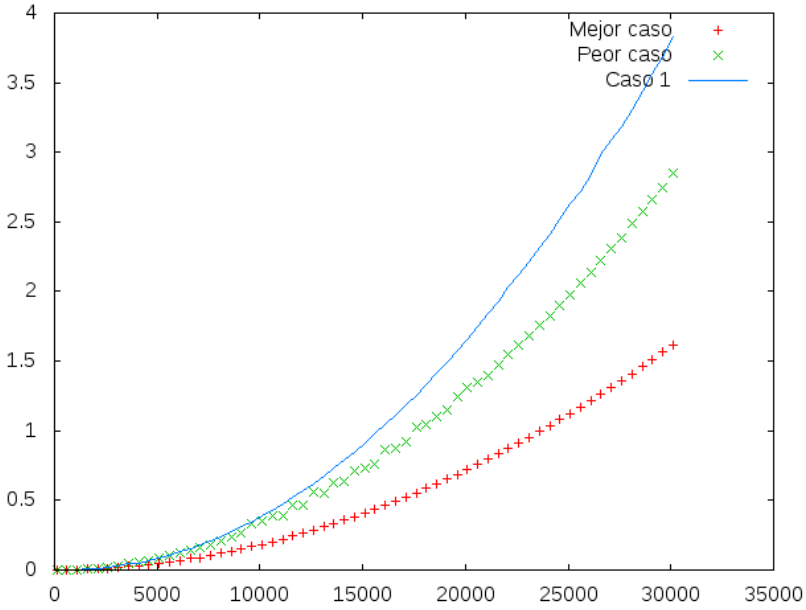
```
Final set of parameters                 Asymptotic Standard Error
=======================                 ==========================

a               = 3.09247e-05           +/- 5.064e-07    (1.638%)

gnuplot> plot 'tiempo_desc_ord.dat' with p title 'Datos',
3.587e-5*( log10(x)/log10(2) ) with l title 'Funcion'
```

Se ha obtenido la siguiente gráfica:



Ejercicios 4: Mejor y peor caso

Se han ejecutados los scripts del algoritmo de la burbuja tres veces: una con el mejor caso, otra con el peor caso y una última con el mismo caso que en el ejercicio primero. Los resultados obtenidos han sido los siguientes:



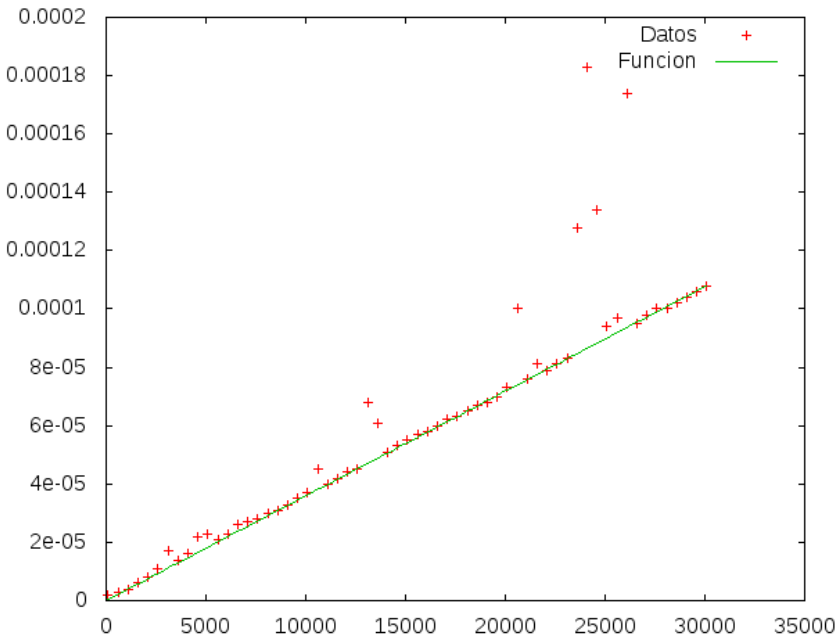
Ejercicio 5: Dependencia de la implementación

Eficiencia teórica mejor caso (vector ordenado):

$$1+\sum_{j=0}^{n-2} 3 \;=\; 1+3(n-2) \;=\; 3n-5$$

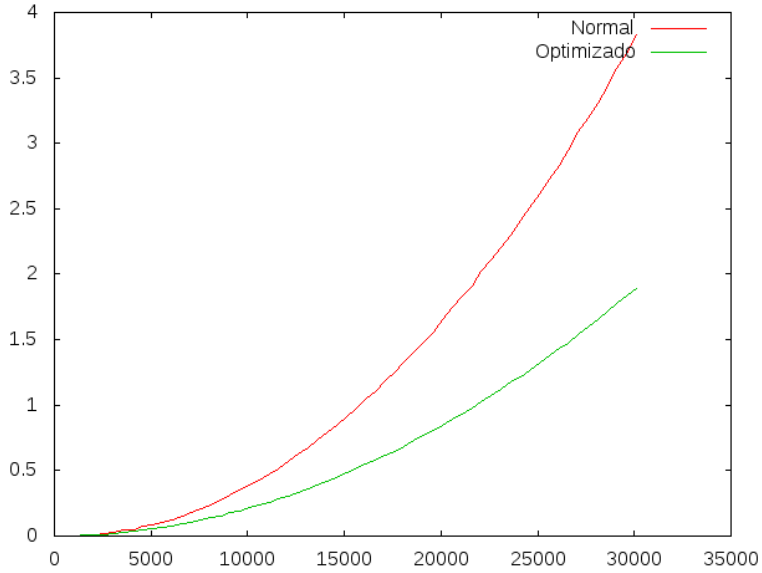
$$t(30100)=0{,}000108\,s \qquad C\cdot(3x-5)=0{,}000108 \;\rightarrow\; C=\frac{0{,}000108}{3\cdot 30100-5}=1{,}196\cdot 10^{-9}$$

```
Terminal type set to 'x11'
gnuplot> h(x) = a*x + b
gnuplot> fit h(x) 'tiempos_ordenacion.dat' via a,b

gnuplot> plot 'tiempos_ordenacion.dat' with p title 'Datos', 1.196e-9*
(3*x - 5) with l title 'Funcion'
```



La gráfica se ajusta completamente a la previsión realizada.

Ejercicio 6: Influencia del proceso de compilación



Ejercicio 7: Multiplicación matricial

Para multiplicar dos matrices bidimensionales se ha empleado el siguiente algoritmo:

```
1 void Multiplica (const int tam, int **A, int **B, int **Res) {
2     for (int i=0 ; i<tam ; i++) {
3         for (int j=0 ; j<tam ; j++) {
4             Res[i][j]=0;
5             for (int k=0 ; k<tam ; k++)
6                 Res[i][j] = Res[i][j] + A[i][k]*B[k][j];
7         }
8     }
9 }
```

Eficiencia teórica:

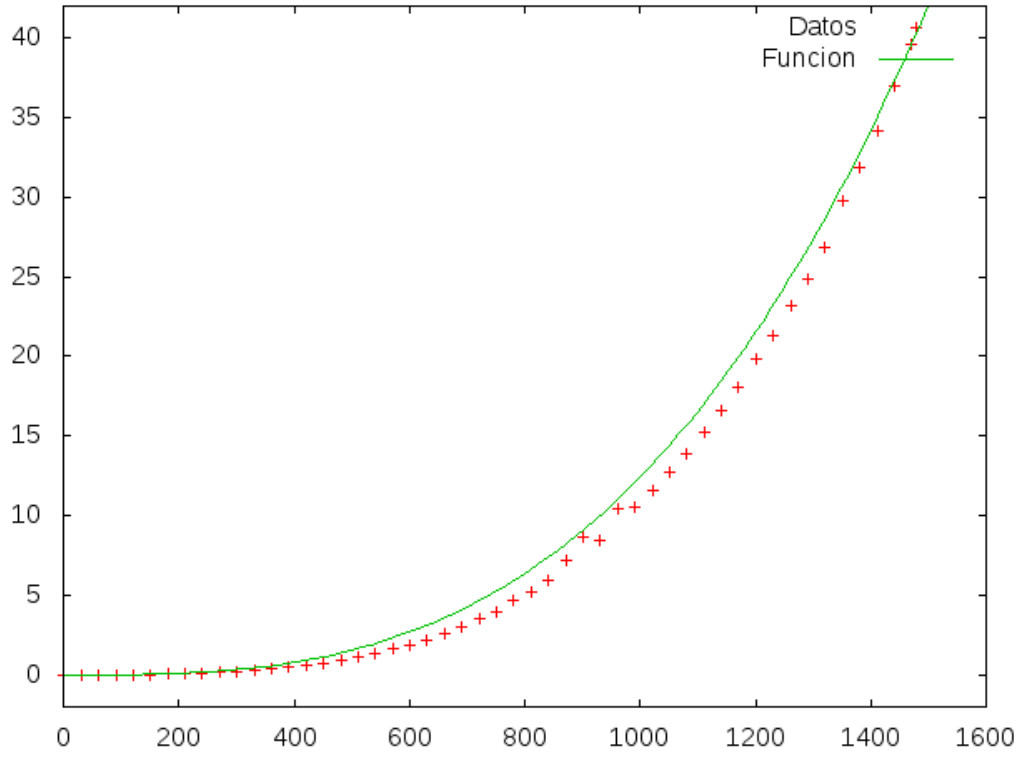
$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(2+\sum_{k=0}^{n-1}5\right) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(2+5n) = \sum_{i=0}^{n-1}(2n+5t^2) = 2n^2+5t^3$$

La eficiencia es de orden $O(n^3)$

Eficiencia empírica:

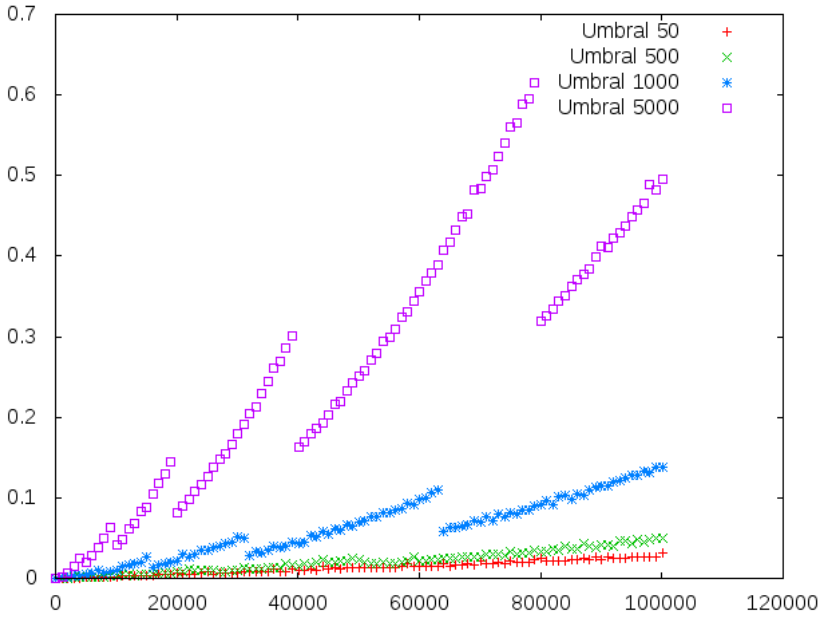
Se ha ejecutado el programa multiplicando matrices de orden $n \times n$ y siendo n un entero que ha variado desde 1 hasta 1500, con incrementos de 30 unidades en cada iteración y siendo los números de las matrices a multiplicar enteros aleatorios entre 0 y 20.

En la siguiente gráfica se puede observar la relación entre la eficiencia teórica calculada y la eficiencia empírica:

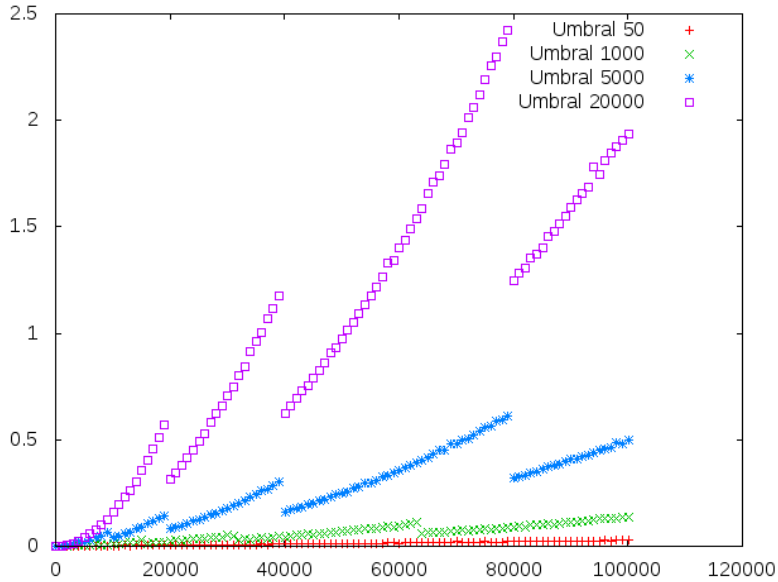


Ejercicio 8: Ordenación por Mezcla

En las siguientes gráficas se muestra el tiempo que ha tardado en ejecutarse el algoritmo. En la primera se han graficado simultáneamente los tiempos de ejecución cuando el umbral se ha situado en 50, 500, 1000 y 5000.



En esta otra gráfica se ha añadido a la comparativa anterior otra muestra cuando el umbral es 20000, con lo que se aprecia una mayor crecida en los tiempos de ejecución.



Con estas gráficas se llega a la conclusión de que, cuanto mayor es el umbral, que implica usar el algoritmo de ordenación por inserción, más tarda en ejecutarse el programa.

Detalles técnicos

Para llevar a cabo esta práctica el computador usado tenía las siguientes características:

Ordenador:

- Marca: Acer
- Modelo: Aspire V3-572G

Hardware

- Procesador: Intel® Core™ i5-5200U
- CPU: 2.20GHz × 4 núcleos
- Reloj: 100 MHz
- Memoria RAM: 8 GB
- Velocidad de reloj: 1600 Mhz (0,6ns)

Sistema Operativo:

- Ubuntu 16.04 LTS 64 bits

Compilador y opciones de compilación:

- Compilador: g++