-ANÁLISIS DE LA EFICIENCIA DE LOS ALGORITMOS-

Álvaro Fernández García
Alexandra-Diana Dumitru
Ana María Romero Delgado
Marino Fajardo Torres
Germán Castro Montes

ÍNDICE

1.- DESCRIPCIÓN DEL PROCESO DE MEDIDA

Para llevar a cabo el proceso de medida en los distintos algoritmos se han seguido los pasos descritos en el guión de prácticas. Primero se ha introducido en el código de cada uno de los algoritmos la biblioteca correspondiente, (en algunos se ha empleado la biblioteca ctime y en otros chrono), a continuación se han declarado las estructuras correspondientes (clock_t para ctime o time_point para crono) para representar la medida del tiempo antes de ejecutar el algoritmo (tantes) y la medida del tiempo después del algoritmo (tdespues). En tercer lugar se asigna a tantes y a tdespues sus valores correspondientes con las funciones de cada una de las bibliotecas:

ctime →
```
tantes = clock();
//Llamada al algoritmo.
tdespues = clock();
```

chrono →
```
tantes = high_resolution_clock::now();
//sentencia o programa a medir
tdespues = high_resolution_clock::now();
```

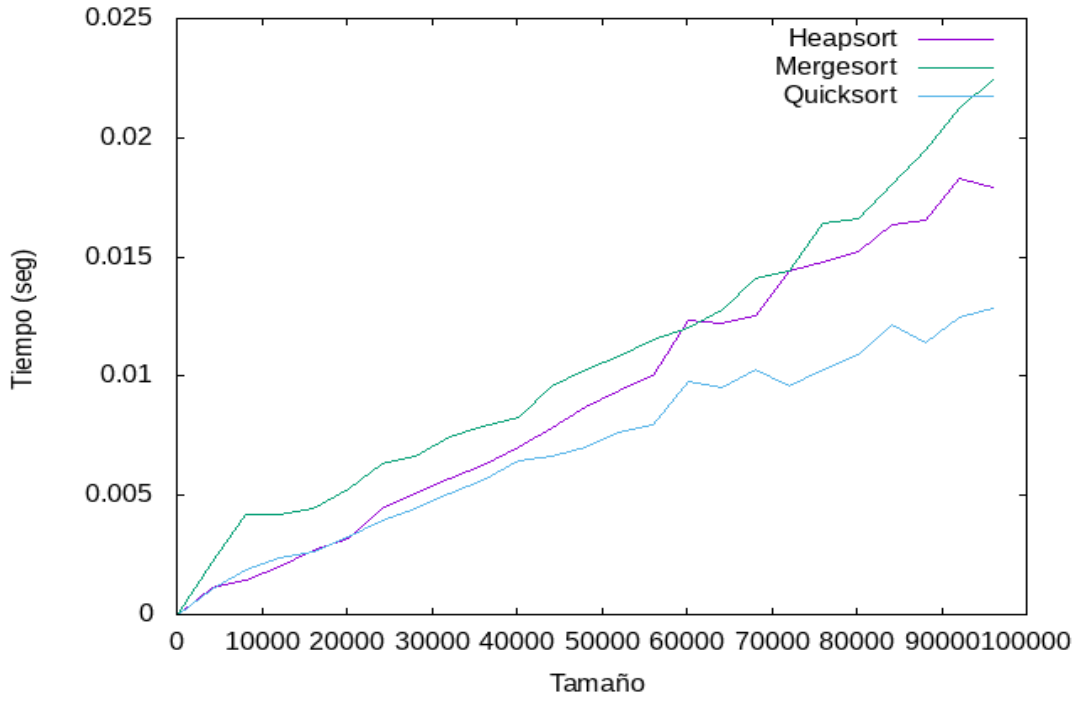
Por último en la variable “distancia” se almacena el periodo de tiempo transcurrido (tdespues – tantes) expresado en segundos. Dicho tiempo se imprime junto con el tamaño de la entrada.
Anotar también, que para realizar distintas medidas se ha realizado un script sencillo que llama al algoritmo con distintas entradas y genera un archivo con los resultados.

2.- ALGORITMOS DE ORDEN NLOG(N)

El análisis práctico se ha realizado con los mismos datos (desde 100 hasta 96100 de 4000 en 4000 obteniéndose un total de 25 mediciones), en el mismo computador. Los resultados obtenidos son los siguientes:

Tamaño	Mergesort	Heapsort	Quicksort
100	4e-05	1.8e-05	2.4e-05
4100	0.002206	0.001159	0.001068
8100	0.004195	0.001421	0.001901
12100	0.004222	0.00203	0.002368
16100	0.004433	0.002698	0.002643
20100	0.005249	0.003219	0.003267
24100	0.00631	0.004458	0.003967
28100	0.006646	0.005063	0.00445
32100	0.007448	0.005671	0.005101
36100	0.007866	0.00628	0.005629
40100	0.008283	0.007035	0.00645
44100	0.009595	0.00781	0.006656
48100	0.010266	0.008714	0.007021
52100	0.010831	0.009393	0.007654
56100	0.011533	0.01005	0.007986
60100	0.0120	0.012314	0.009799
64100	0.012803	0.012234	0.009552
68100	0.014112	0.012542	0.010274
72100	0.014409	0.014414	0.009594
76100	0.016435	0.014791	0.010272
80100	0.016632	0.015224	0.010883
84100	0.018028	0.016367	0.012181
88100	0.019503	0.016524	0.011433
92100	0.021254	0.018294	0.01244
96100	0.022452	0.017892	0.012827

A continuación encontramos una gráfica en la que aparecen representados los distintos algoritmos para que sea más apreciable la comparativa entre ellos:

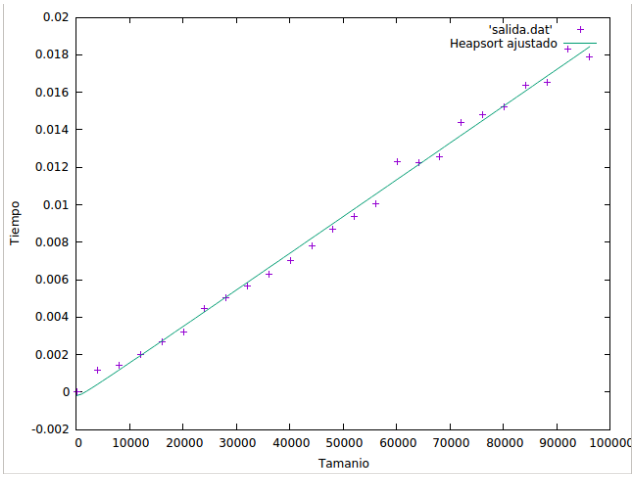


Notar que, aunque todos los algoritmos tienen exactamente el mismo orden, las gráficas no son iguales entre si. Para tamaños pequeños, los algoritmos de Heapsort y Quicksort presentan tiempos similares, aunque a partir de un tamaño aproximadamente de 20000 elementos, el algoritmo de Quicksort se hace mucho más eficiente. Por otra parte el algoritmo de Mergesort es el que presenta una peor eficiencia. Esto se debe a que no todos los algoritmos presentan las mismas constantes ocultas en sus funciones. Para poder determinar las constantes ocultas se realiza el análisis híbrido. Su valor se expresa a continuación:

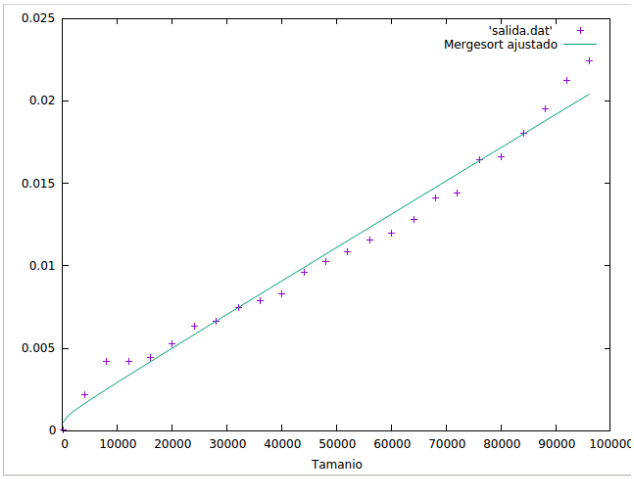
Función de ajuste: f(x) = a0*x + a1*log(x)	**Constantes ocultas.**
Heapsort	a0 = 1.97162e-07 a1 = -4.53394e-05
Mergesort	a0 = 2.00414e-07 a1 = 9.88497e-05
Quicksort	a0 = 1.25338e-07 a1 = 9.6846e-05

En un principio, podemos observar que las gráficas de estos algoritmos presentan una forma muy irregular con múltiples picos, sin embargo tras haber realizado el ajuste por mínimos cuadrados con la función a0*x + a1*log(x) podemos comprobar que, efectivamente, los datos se ajustan a ella. A continuación se presentan las gráficas que lo demuestran:

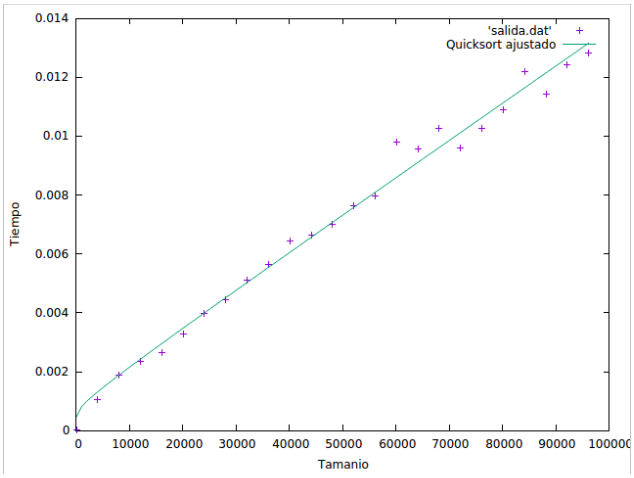
Heapsort



Mergesort



Quicksort



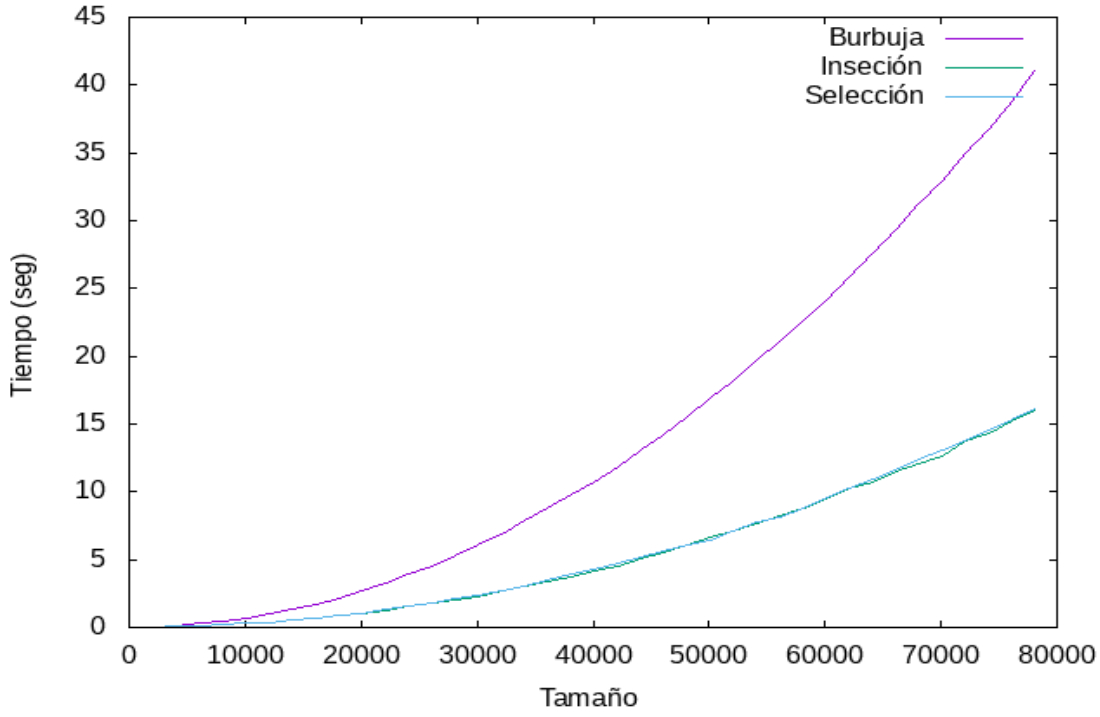
3.- ALGORITMOS DE ORDEN N^2

El análisis práctico se ha realizado con los mismos datos (desde 100 hasta 78100 de 2000 en 2000 obteniéndose un total de 40 mediciones), en el mismo computador. A continuación aparece una muestra de las primeras 25 mediciones:

Tamaño	Burbuja	Inserción	Selección
100	6.9327e-05	4.136e-05	4.602e-05
2100	0.0377461	0.0136332	0.0135491
4100	0.141303	0.0523087	0.0569532
6100	0.314067	0.115516	0.126246
8100	0.445964	0.215586	0.220492
10100	0.684526	0.288065	0.28406

12100	0.977858	0.387925	0.391265
14100	1.33341	0.512428	0.529779
16100	1.72952	0.652784	0.684621
18100	2.19487	0.880539	0.847446
20100	2.69719	0.996596	1.03424
22100	3.27687	1.20188	1.29995
24100	3.89536	1.5307	1.54228
26100	4.54979	1.78464	1.74997
28100	5.29986	2.00134	2.09664
30100	6.06074	2.24743	2.31665
32100	6.91871	2.70785	2.67355
34100	7.83896	3.07436	3.07575
36100	8.75753	3.40693	3.45194
38100	9.8041	3.74844	3.92946
40100	10.766	4.14869	4.27888
42100	11.8968	4.54335	4.69598
44100	13.0799	5.09701	5.13608
46100	14.233	5.55601	5.63834
48100	15.5145	6.11225	6.13644

Aquí se muestra una gráfica comparativa de los tres algoritmos:

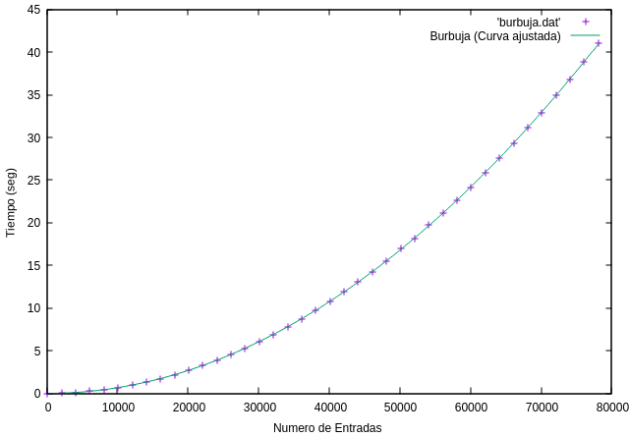


Como podemos observar en la gráfica comparativa, el algoritmo de la burbuja a pesar de ser también de orden n^2 presenta una eficiencia mucho más pobre. Con respecto a los algoritmos de inserción y selección, a simple vista puede parecer que son exactamente iguales, pero recurriendo nuevamente al análisis híbrido para determinar las constantes ocultas nos damos cuenta de que las constantes que presenta ambos no son iguales, luego a la larga acabará notándose la diferencia. Una vez más las constantes ocultas provocan que algoritmos con el mismo orden de eficiencia presenten tiempos distintos:

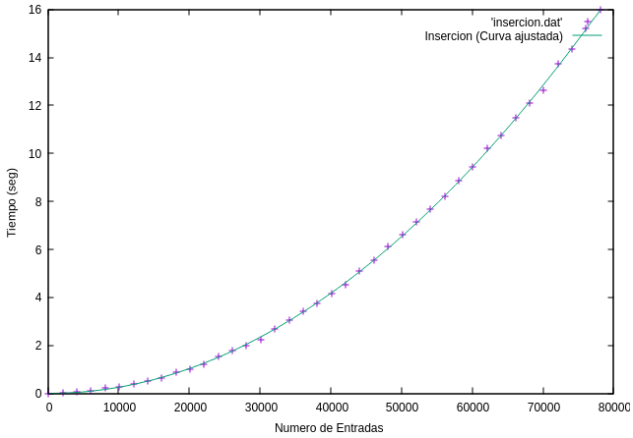
Función de ajuste: f(x) = a0*x*x+a1*x+a2	Constantes ocultas.
Burbuja	a0 = 6.71335e-09 a1 = -1.25877e-07 a2 = 0.0082703
Inserción	a0 = 2.63748e-09 a1 = -1.40659e-06 a2 = 0.00851581
Selección	a0 = 2.68701e-09 a1 = -3.21316e-06 a2 = 0.0345924

La representación gráfica del ajuste de las tres gráficas es la siguiente:

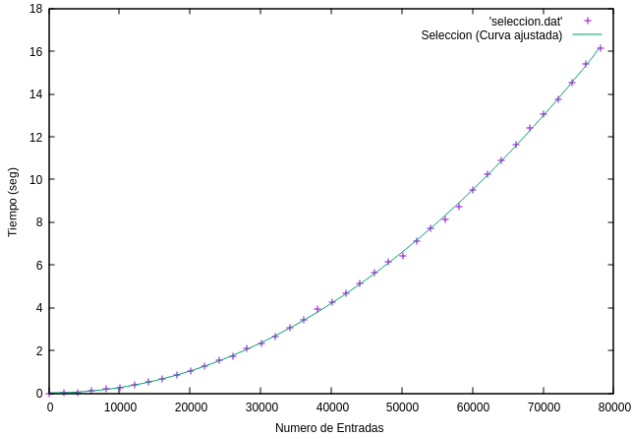
Burbuja



Inserción



Selección



Como podemos ver, en este caso la función se ajusta mucho mejor a los datos que en los algoritmos con orden nlog(n).

4.- ALGORITMOS DE ORDEN N^3

El algoritmo de Floyd, que calcula el costo del camino mínimo entre cada par de nodos de un grafo dirigido, presenta los siguientes tiempos para los siguiente tamaños (empezando en 400 hasta 9600 de 400 en 400, lo que da un total de 24 datos) y con una optimización -O3:

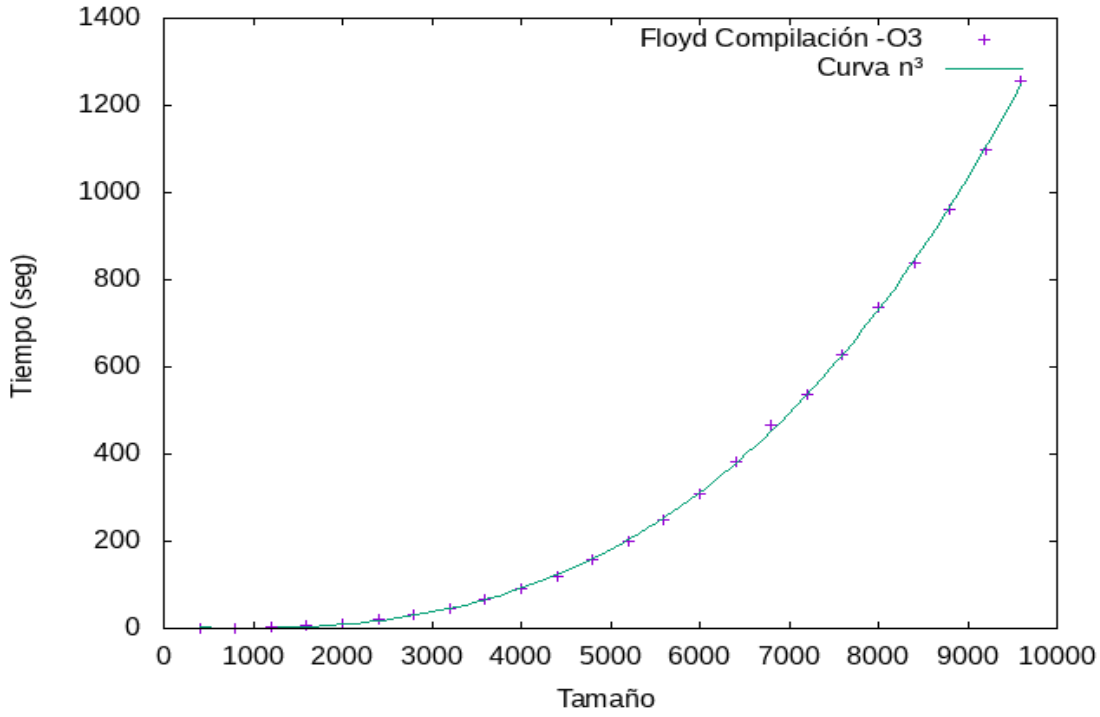
Tamaño	Floyd
400	0.070921
800	0.585181
1200	2.10569
1600	4.90275
2000	10.7982
2400	21.9545
2800	35.0722
3200	52.1105
3600	74.3218
4000	101.655
4400	135.081
4800	174.37
5200	224.848
5600	279.936
6000	343.186
6400	430.934
6800	501.181
7200	591.866
7600	696.234
8000	806.792
8400	936.535
8800	1075.04
9200	1222.63
9600	1239.4

Podemos observar los tiempos tan altos que presenta este algoritmo comparado con otros órdenes de eficiencia, para tamaños muchos más pequeños. En definitiva un orden n^3 no es deseable.

Sus constantes ocultas son las siguientes:

Función de ajuste: f(x) = a0*x*x*x+a1*x*x+a2*x+a3	Constantes ocultas.
Floyd	a0 = 6.65808e-10 a1 = 1.15485e-05 a2 = -0.0382268 a3 = 27.4892

Su representación gráfica ya ajustada es la siguiente:



Como podemos ver el ajuste es también bastante bueno.

5.- SUCESIÓN DE FIBONACCI

La sucesión de Fibonacci presenta un orden exponencial b^n donde b es el valor del número aureo. Se han realizado una serie de mediciones que van desde 0 hasta 38 de uno en uno. A continuación se muestran los primeros 25 datos:

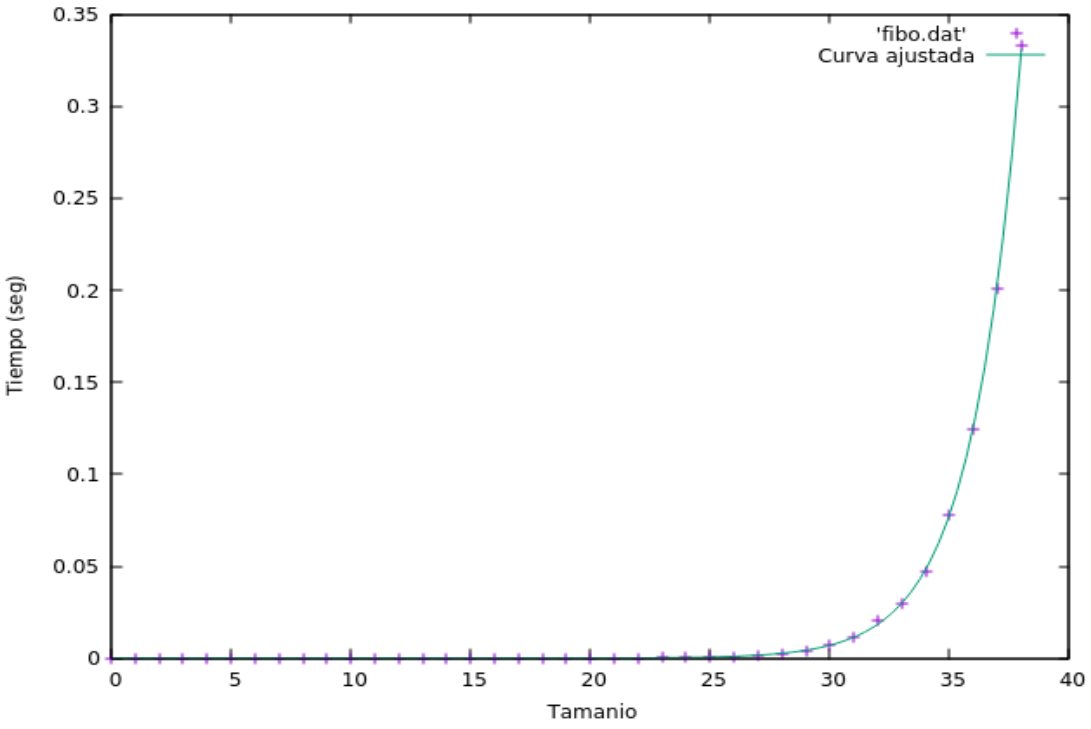
Tamaño	Fibonacci
0	1.4e-05
1	2e-06
2	2e-06
3	2e-06

4	1e-06
5	2e-06
6	2e-06
7	1e-06
8	2e-06
9	3e-06
10	3e-06
11	3e-06
12	4e-06
13	6e-06
14	8e-06
15	1e-05
16	1.6e-05
17	2.6e-05
18	3.1e-05
19	4.6e-05
20	5.9e-05
21	0.000107
22	0.000146
23	0.000236
24	0.00038

Sus constantes ocultas son las siguientes:

Función de ajuste: f(x) = $a_0 * \left(\frac{1+\sqrt{5}}{2}\right)^x$	**Constantes ocultas.**
Sucesión de Fibonacci	a0 = 3.78054e-09

Los datos se ajustan bien a la función como se puede apreciar en la siguiente gráfica:



6.- COMPARACIONES: PRINCIPIO DE INVARIANZA DISTINTOS COMPUTADORES Y DISTINTAS OPTIMIZACIONES

El principio de invarianza viene a decirnos lo siguiente: “Dos implementaciones distintas de un mismo algoritmo difieren en eficiencia más que a lo sumo en una constante multiplicativa”.

$$t_1(n) \leq c * t_2(n)$$

Con implementaciones distintas nos referimos al uso de distintos compiladores para generar el ejecutable, distintas optimizaciones, distintos lenguajes de programación, etc. Nunca podremos cambiar el orden de eficiencia de un algoritmo solo modificando alguno de estos elementos, como mucho conseguiremos que tenga mejores tiempos o peores dependiendo del valor de esa constante multiplicativa.

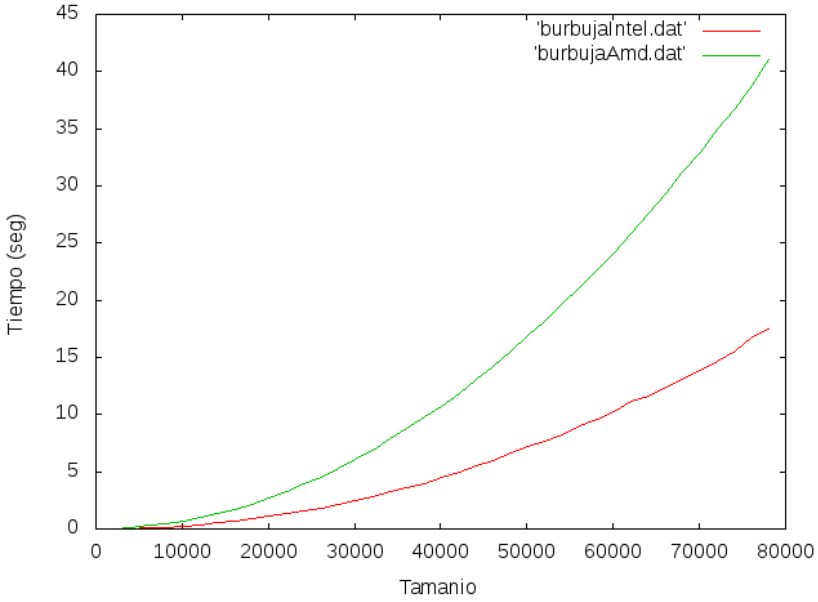
En este apartado se pretende ilustrar este principio. Para ello se han hecho las siguientes mediciones:

1. Implementación de los algoritmos de búsqueda en distintos computadores. Se han ejecutado los algoritmos de burbuja, inserción y selección en dos máquinas con especificaciones completamente distintas. Dichas especificaciones son las siguientes:

   - Intel Core i7-6700HQ CPU @ 2.60GHz
   - AMD A10-7300 Radeon R6 @ 1.90GHz

A continuación se presentan una serie de gráficas en las que se muestra la comparativa de los datos obtenidos junto con el calculo de la constante multiplicativa. Para ello se ha realizado la media de los tiempos obtenidos en los dos computadores:

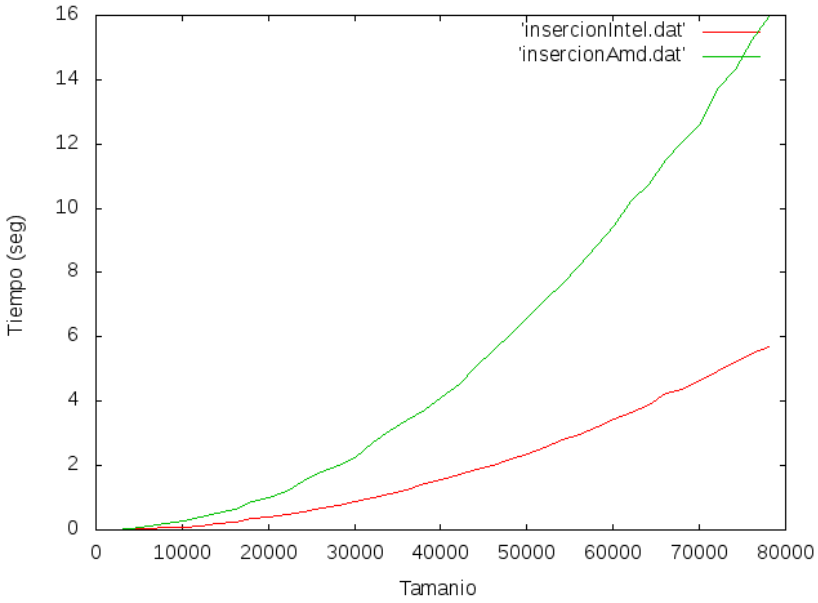
Burbuja



AMD = 13.84
Intel = 5.83

c = 13.84 / 5.83 = 2.37

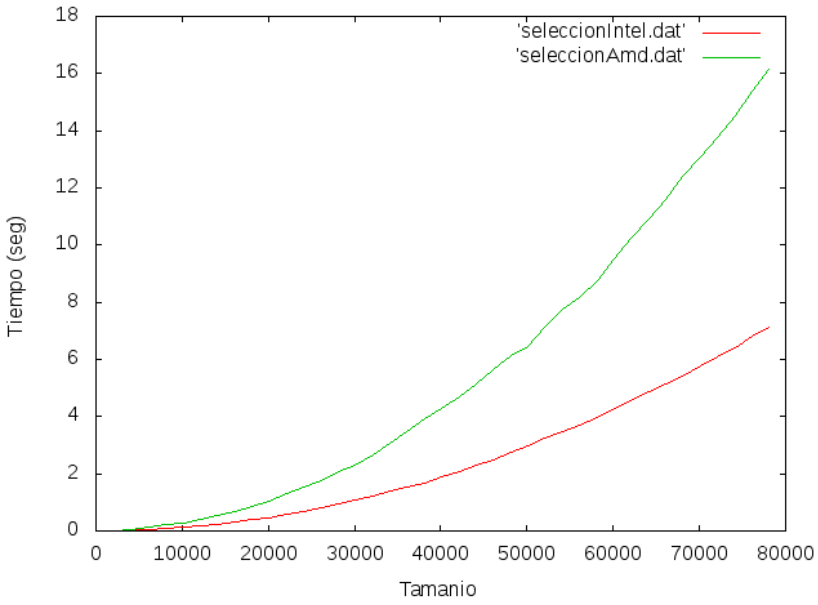
Inserción



AMD = 5.39
Intel = 1.96

c = 5.39 / 1.96 = 2.7

Selección

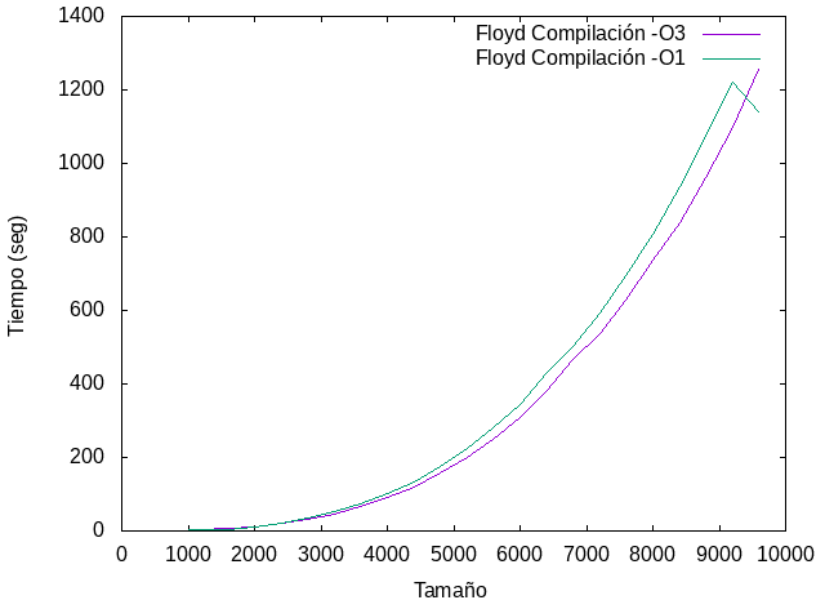


AMD = 5.45
Intel = 2.42

c = 5.45 / 2.42 = 2.25

Aproximadamente las constantes rondan los 2.5. Sin embargo los algoritmos siguen teniendo un orden n^2, solo que en la maquina de Intel se ejecutan 2.5 veces más rápido.

2. Compilación del algoritmo de Floyd con distinta optimización. La forma en la que el compilador genera el código puede afectar el rendimiento de un algoritmo. A continuación se muestra un ejemplo en el que se ha compilado el código con optimización -O1 y -O3. La constante multiplicativa se ha calculado de forma similar al apartado anterior.

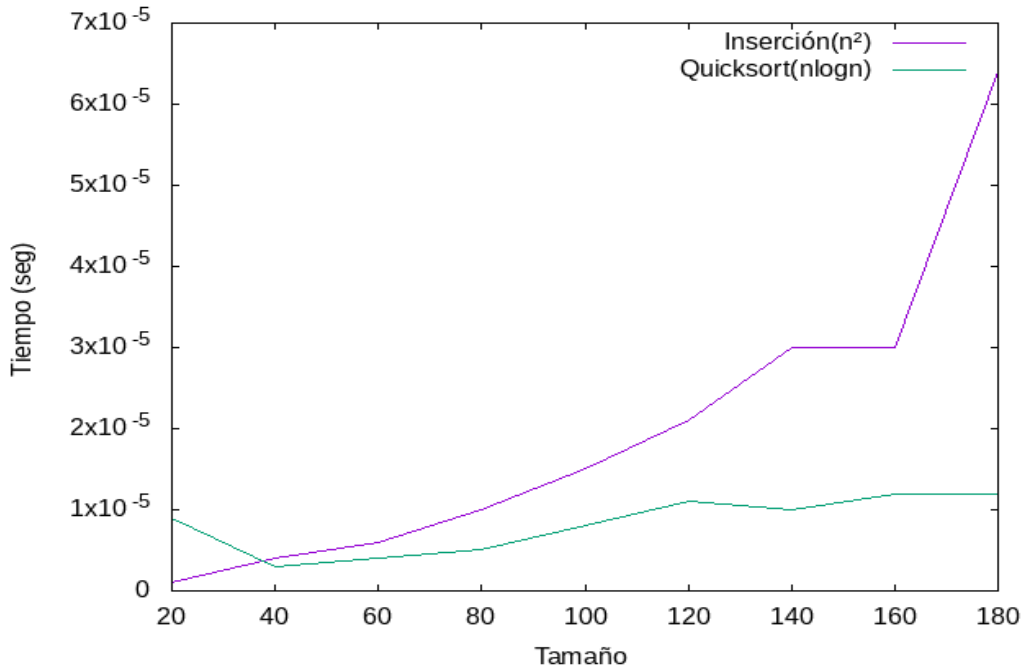


-O1 = 373.40
-O2 = 341.98

c = 373.40 / 341.98 = 1.091

El orden continúa siendo cúbico pero en este caso, el haber compilado con la opción -O3 hemos conseguido que se ejecute 1.091 veces más rápido.

Por último, simplemente apuntar la diferencia abismal que existe entre los dos órdenes más eficientes que encontramos en esta práctica: el orden nlog(n) y el orden n^2:



En definitiva podemos afirmar que es preferible tener un orden nlog(n).