ALGORÍTMICA Jorge Gangoso Klöck 49398653N

Práctica 1

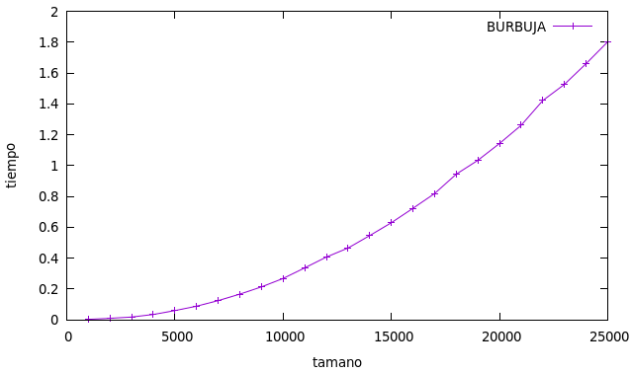
I - Eficiencia empírica

Tiempos calculados con procesador: Intel Core i5-3350P 3.10Ghz

I.I Algoritmo **Burbuja** de Ordenación

Como se ha visto mediante el cálculo teórico este algoritmo tiene una eficiencia $O(n^2)$. En la gráfica se puede apreciar como para valores algo más elevados del vector el tiempo requerido asciende de forma cuadrática en una curva que deja sospechas de que otros algoritmos podrían ser más eficientes, es decir, se aprecia un amplio margen de mejora. A la izquierda tenemos los tiempos obtenidos durante la ejecución empleando el reloj de la librería <ctime>. Para obtener los datos de forma rápida se ha incluído un bucle que itera sobre el tamaño del vector en el código del algoritmo burbuja y después se ha redireccionado la salida estándar a un fichero de datos desde la consola de comandos utilizando la redireccion de flujo (>).

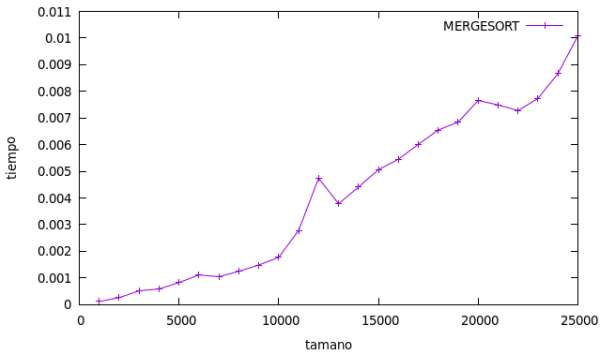
Tamaño Vector	Tiempo(s)
1000	0.004252
2000	0.010889
3000	0.019175
4000	0.035863
5000	0.061193
6000	0.089652
7000	0.12549
8000	0.167802
9000	0.214516
10000	0.269087
11000	0.337959
12000	0.407812
13000	0.46631
14000	0.546542
15000	0.630252
16000	0.723711
17000	0.818956
18000	0.944842
19000	1.03455
20000	1.14363
21000	1.26147
22000	1.42162
23000	1.52653
24000	1.66052
25000	1.80218



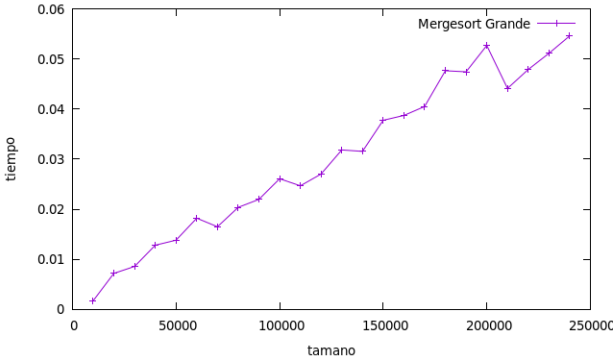
I.II Algoritmo **Mergesort** de Ordenación

Es un algoritmo mucho más eficiente que el de Burbuja con una eficiencia O(nlog(n)) y se aprecia enormemente en los resultados como veremos más adelante en la comparación. Para este algoritmo en lugar de un bucle interno y para maximizar el uso del trabajo ya realizado se ha utilizado un script que llama al programa mergesort con parámetros ascendentes que el programa utilizará como tamaños de vector. La redirección de la salida a un fichero de datos se realiza desde el propio script utilizando el comando (>>). Cabe decir que puesto que la diferencia de velocidades es demasiado grande he realizado dos tomas de datos con valores distintos para apreciarlo mejor.
La tabla contiene a la izquierda tamaños equivalentes a los del algoritmo Burbuja y a la derecha tamaños más apropiados para estimar su eficiencia.

Tamaños	Tiempo	Tamaños Grandes	Tiempo
1000	0.000119657	25000	0.010078
2000	0.000264132	30000	0.008588
3000	0.000516155	40000	0.012825
4000	0.00058905	50000	0.013818
5000	0.000828587	60000	0.018207
6000	0.00111413	70000	0.016493
7000	0.00104766	80000	0.020371
8000	0.00125165	90000	0.02196
9000	0.00148783	100000	0.026077
10000	0.00177333	110000	0.024664
11000	0.002784	120000	0.027015
12000	0.004747	130000	0.03183
13000	0.003784	140000	0.031542
14000	0.004411	150000	0.037755
15000	0.005063	160000	0.038716
16000	0.005454	170000	0.0405
17000	0.006015	180000	0.047634
18000	0.006551	190000	0.047422
19000	0.006851	200000	0.052683
20000	0.00766	210000	0.044111
21000	0.00749	220000	0.047889
22000	0.00728	230000	0.051118
23000	0.007736	240000	0.054589
24000	0.008669		



Gráfica tamaños pequeños

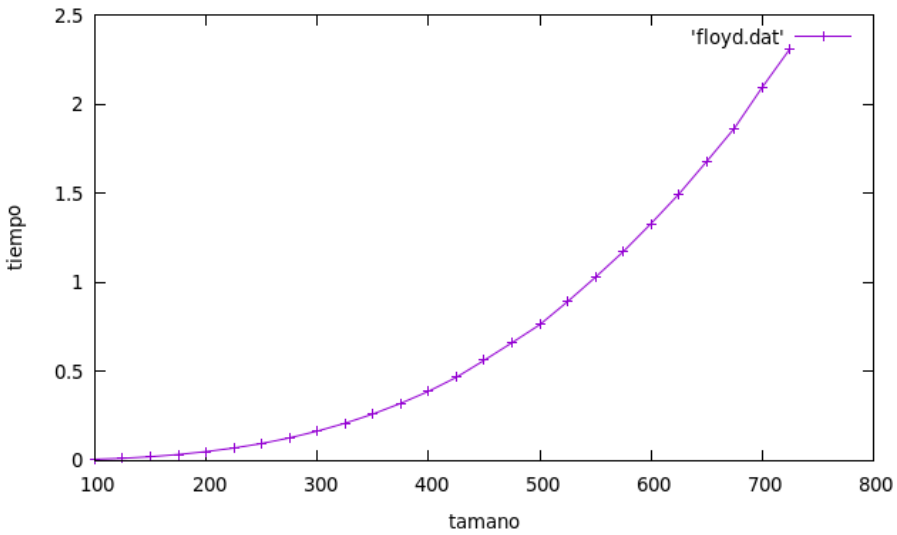


Gráfica tamaños grandes

En este caso la gráfica no queda tan uniforme como la de burbuja, pero esto es de esperar ya que los tiempos son tan inferiores que los valores se ven afectados por factores ajenos al algoritmo de forma más notoria. Se podría haber pulido un poco los valores más excéntricos realizando el cálculo de tiempo dentro de otro bucle y luego dividiendo entre el número de iteraciones para conseguir estimaciones de tiempo más precisas que no dependan tanto de los posibles casos (mejor, peor, intermedios).

I.III Algoritmo de **Floyd** para análisis de grafos

Este algoritmo que tiene por finalidad encontrar el camino mínimo/óptimo entre dos nodos de un grafo ponderado y tiene una eficiencia de $O(n^3)$, lo cual implica una curva muy pronunciada que vuelve el algoritmo prácticamente inservible para tamaños de caso considerables. En este caso cuando hablamos de tamaño nos referimos al tamaño de la matriz generada para estudiar un grafo asociado.

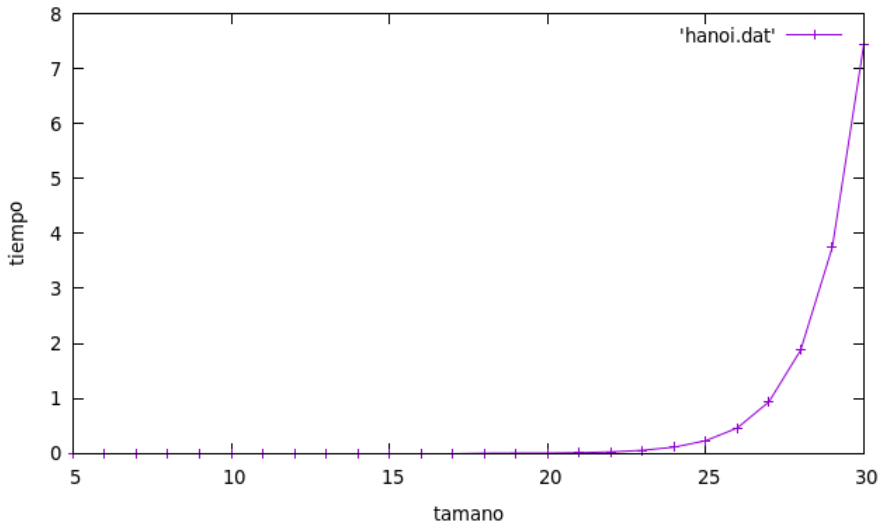


Tamaño	Tiempo
100	0.006639
125	0.012304
150	0.021264
175	0.033723
200	0.050117
225	0.07025
250	0.095535
275	0.127015
300	0.164945
325	0.209464
350	0.261064
375	0.321008
400	0.388598
425	0.466594
450	0.56221
475	0.660962
500	0.762128
525	0.89141
550	1.02816
575	1.17179
600	1.32929
625	1.49457
650	1.67772
675	1.86419
700	2.09393
725	2.31052

I.IV Algoritmo de **Hanoi**

El algoritmo de Hanoi para resolver el juego de las torres de Hanoi asciende hasta un Orden de complejidad de O(2^n). Es el orden menos eficiente de los que hemos estudiado y los datos se han ajustado a dicha característica. Intentar resolver el problema de las torres de hanoi con 100 discos sería imposible de realizar a tiempo antes de la fecha límite de este trabajo.

N.º Discos	Tiempo
5	3,00E-06
6	3,00E-06
7	3,00E-06
8	5,00E-06
9	8,00E-06
10	1,50E-05
11	3,00E-05
12	5,80E-05
13	0.000115
14	0.000229
15	0.000457
16	0.000929
17	0.001824
18	0.003653
19	0.006629
20	0.007157
21	0.014422
22	0.028759
23	0.057183
24	0.114717
25	0.232315
26	0.465925
27	0.940982
28	1.87874
29	3.74869
30	7.44434

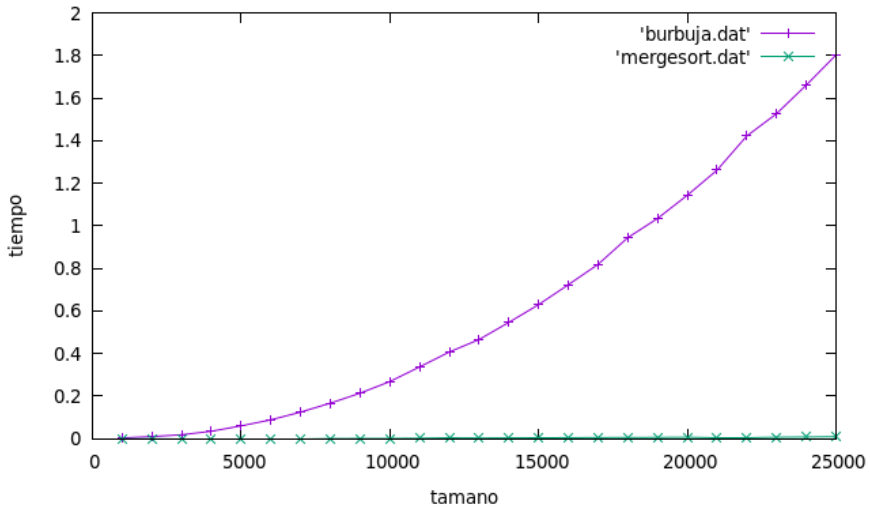


En la gráfica se aprecia perfectamente el ángulo que toma conforme los valores aumentan ligeramente, es fácil hacerse a la idea de los tiempos que podríamos necesitar para valores superiores.

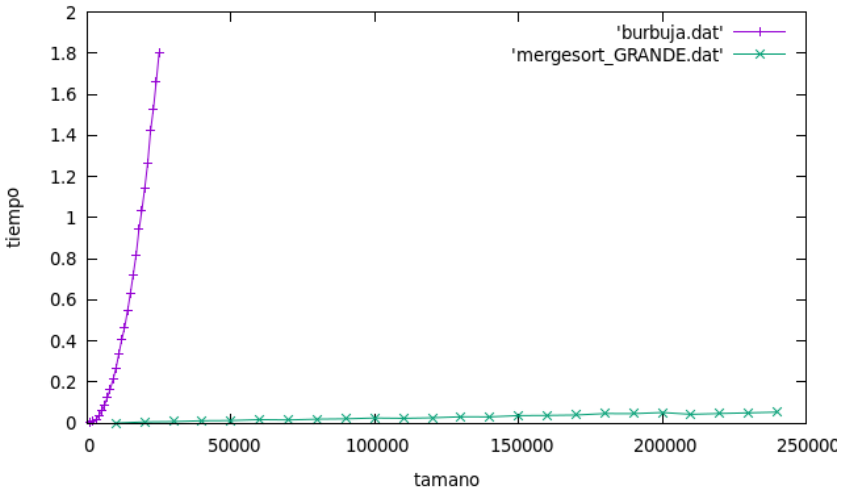
II Comparación entre **Burbuja** y **Mergesort**

Puesto que éstos han sido los algoritmos elegidos para cada uno de los órdenes, procederemos a la única comparación que tiene sentido, ya que son dos algoritmos que realizan la misma tarea. Aquí es donde nos será útil el haber tomado dos valores para Mergesort, para que se pueda apreciar bien la diferencia de velocidad de ambos algoritmos.

Comparación con valores pequeños:

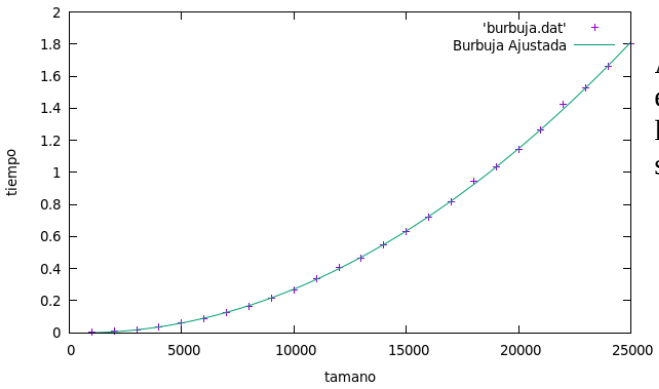


Comparación con valores grandes:



III Cálculo de la Eficiencia Híbrida

III.I Burbuja

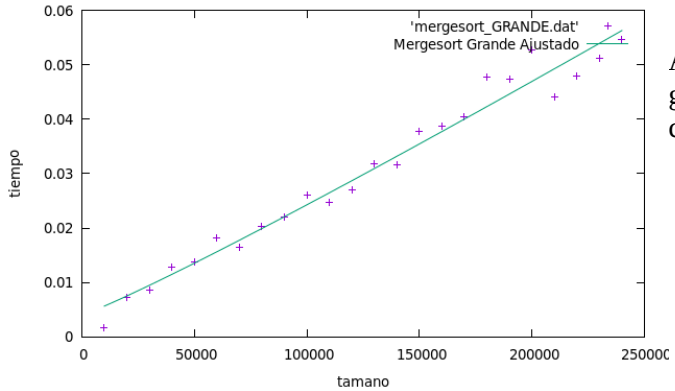


Ajustamos la gráfica a una curva cuadrática de ecuación general ax^2+bx+c, con lo que obtenemos los siguientes valores de constantes ocultas y la siguiente gráfica ajustada.

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

a	= 3.01487e-09	+/- 3.617e-11 (1.2%)
b	= -2.93199e-06	+/- 9.687e-07 (33.04%)
c	= 0.000821772	+/- 0.005466 (665.1%)

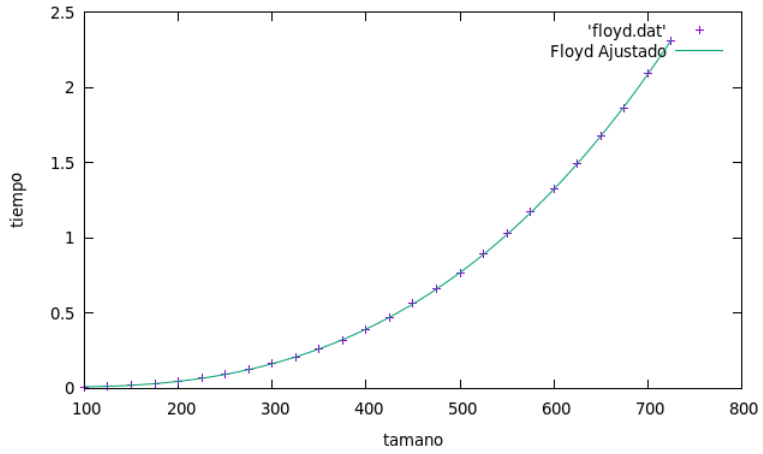
III.II Mergesort



Ajustamos a una curva logarítmica de ecuación general ax*log(x)+b y obtenemos las siguientes constantes ocultas.

Final set of parameters Asymptotic Standard Error
========== ==========
a = 1.7568e-08 +/- 6.427e-10 (3.658%)
b = 0.00404171 +/- 0.001109 (27.45%)

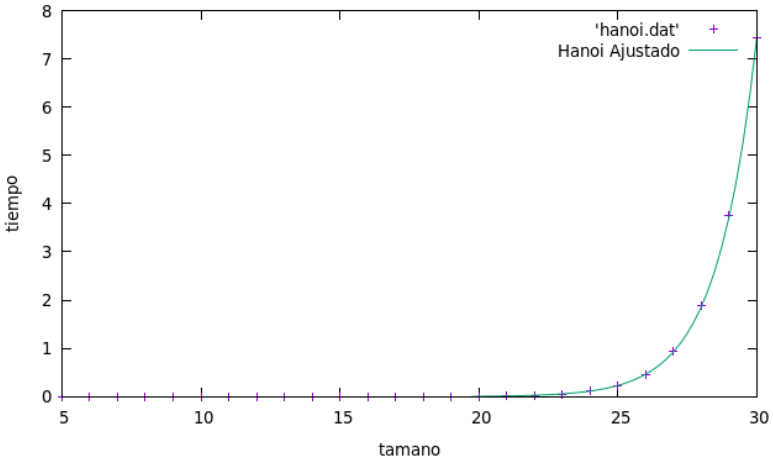
III.III Floyd



Ajustamos los datos del algoritmo de Floyd a una función a*x^3+b*x^2+c*x+d y obtenemos un muy buen ajuste con las siguientes constantes ocultas.

Final set of parameters Asymptotic Standard Error
========== ==========
a = 5.39136e-09 +/- 1.769e-10 (3.281%)
b = 8.07227e-07 +/- 2.208e-07 (27.35%)
c = -0.000258206 +/- 8.279e-05 (32.06%)
d = 0.0230115 +/- 0.009002 (39.12%)

III.IV Hanoi



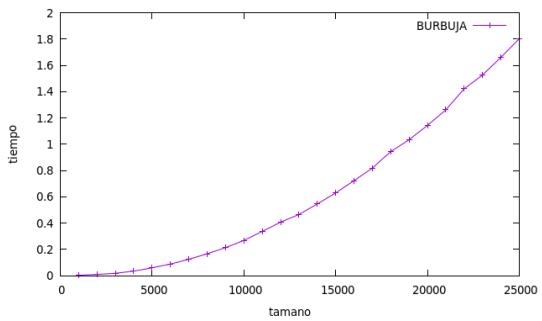
Ajustamos la función a una función a*2^n y obtenemos la siguiente constante oculta.

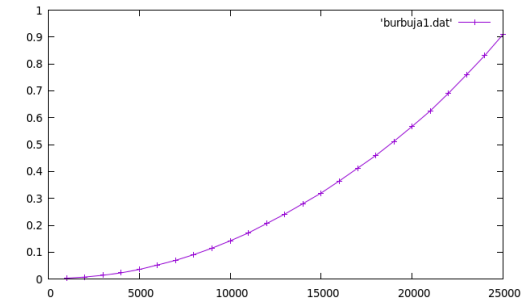
Final set of parameters Asymptotic Standard Error
========== ==========
a = 6.94634e-09 +/- 4.762e-12 (0.06856%)

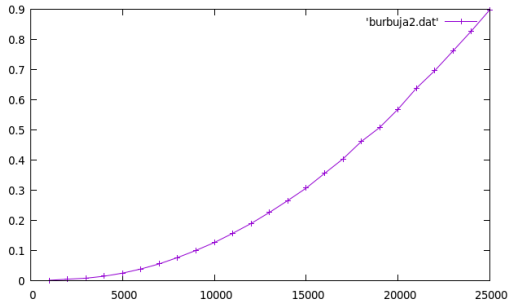
IV Factores externos: Optimización de Código

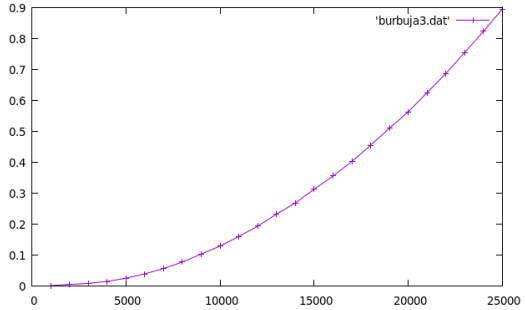
Todas las mediciones se han realizado empleando en la compilación del código la directiva de Optimización O0, O1, O2 y O3.
Las conclusiones obvias mirando las tablas de datos es que las optimizaciones afectan muchísimo a la velocidad del algoritmo y por tanto a su eficiencia empírica (a nivel de constantes ocultas). Un estudio un poco más detenido nos muestra como el gran cambio se produce en el paso de O0 a O1, donde el tiempo de ejecución en la mayoria de casos se reduce a la mitad aproximadamente.
Las otras dos Optimizaciones se mantienen aproximadamente al mismo nivel que O1, aun así, a continuación se mostrarán las gráficas generadas por cada algoritmo con cada una de las optimizaciones.
Las ficheros de datos (.dat) se encuentran en la carpeta del algoritmo asociado dentro de una subcarpeta con el nombre de la Optimización aplicada.

IV.I Burbuja

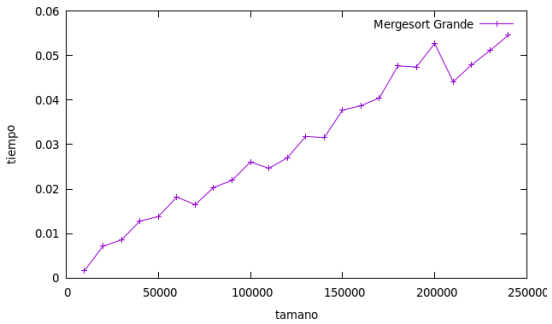


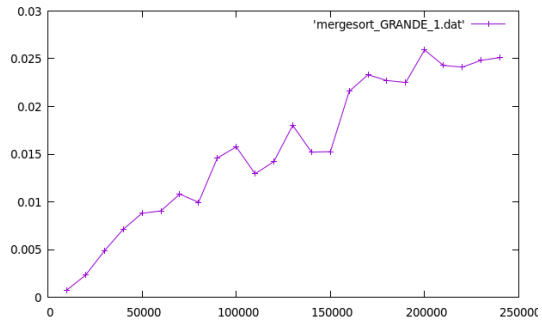


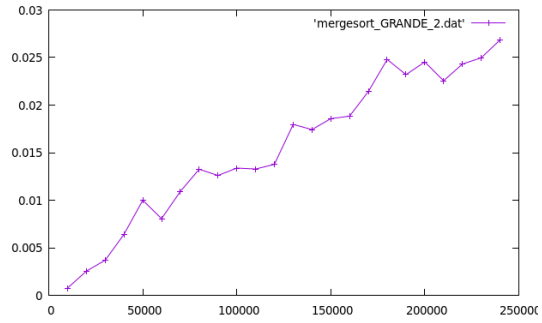


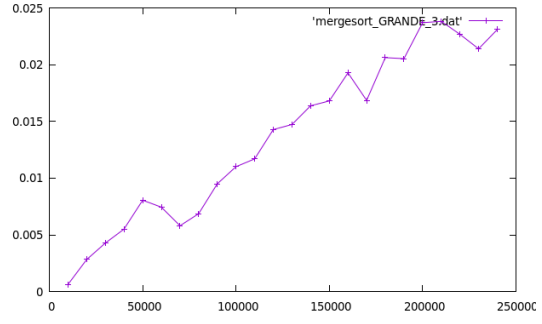


IV.II Mergesort

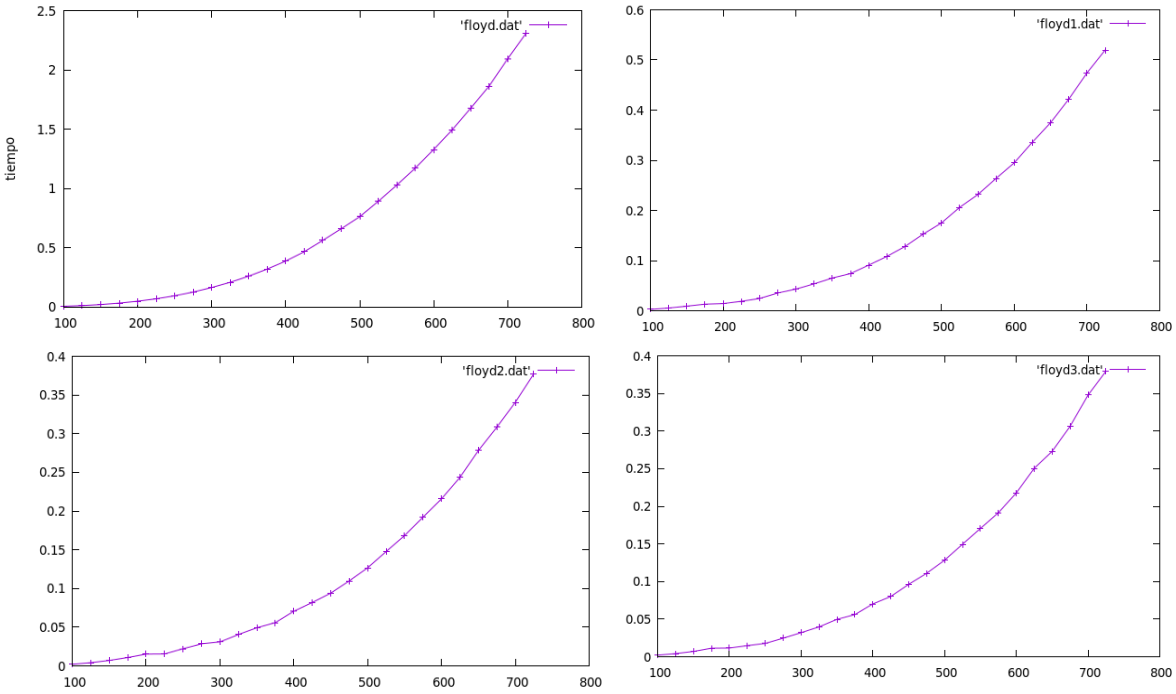




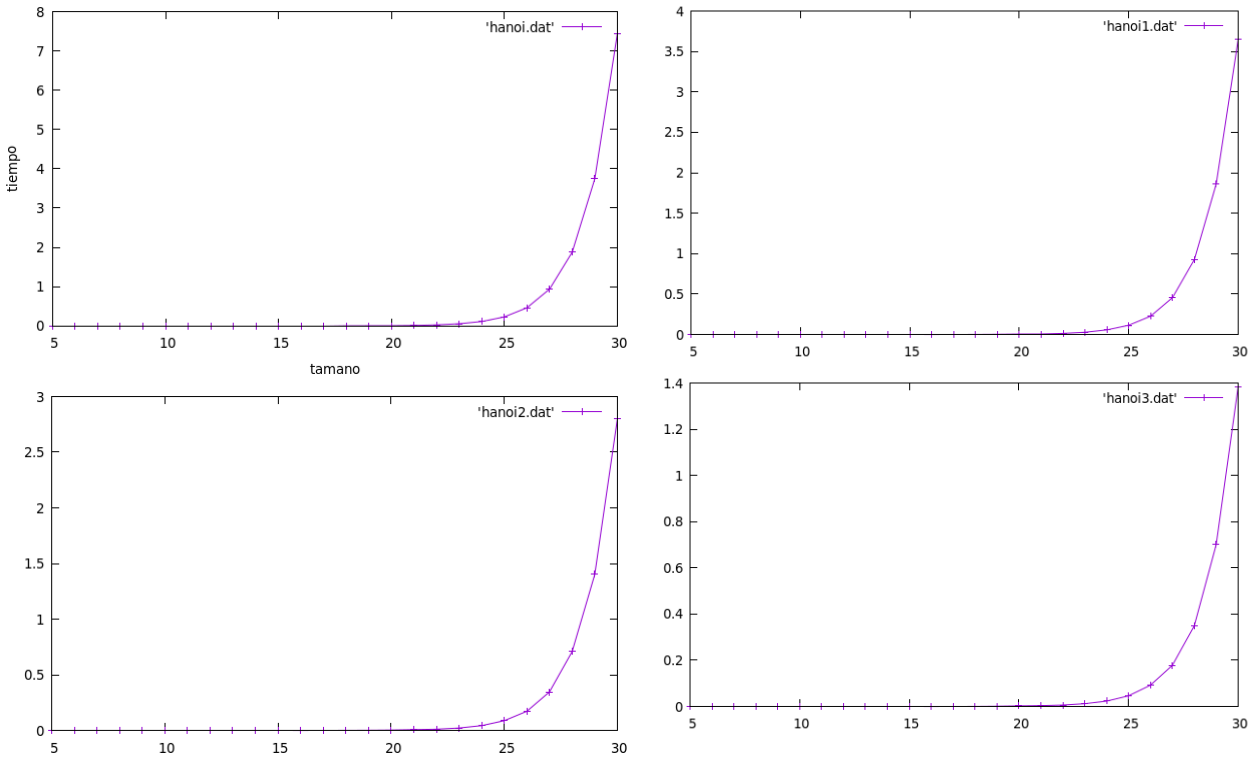




IV.III Floyd



IV.IV Hanoi



Como podemos observar, si bien el cambio es notorio, tampoco es suficiente como para que podamos llegar a considerar un algoritmo que previamente considerábamos más eficiente, menos.
El Orden de eficiencia prevalece enormemente sobre todo con algoritmos de Órdenes tan diferentes.