PRÁCTICA 1

ANÁLISIS DE EFICIENCIA DE ALGORITMOS

Algorítmica
2016-2017

Componentes del Grupo:
Daniel Bolaños Martínez
José María Borrás Serrano
Santiago De Diego De Diego
Fernando De la Hoz Moreno

Índice:

Introducción:

Hemos realizado las ejecuciones de los algoritmos dándole 25 valores a través de un script y los hemos representado en una tabla para cada orden de eficiencia.

Además hemos hecho cambios en la escala del tamaño de los datos para cada tipo de problema para que sean más apreciables los resultados a la hora de plasmarlos gráficamente.

A continuación, presentaremos los datos obtenidos en la ejecución del conjunto de algoritmos con el mismo orden de eficiencia, junto con una breve conclusión de los resultados, las especificaciones de las *CPUs* utilizadas en cada caso y las gráficas obtenidas.

Como una memoria con todos los resultados de cada componente del grupo sería bastante confusa y extensa, hemos decidido plasmar los datos más representativos de cada componente y dejar las comparaciones de los resultados para la exposición oral y de diapositivas.

Ejercicio 1:
Resultados Ejecuciones:
Gráficas individuales:

Se presentan en este sección por un lado las gráficas asociadas a las distintas ejecuciones y por otro las tablas con los valores que han generado dichas gráficas.

Resultados Algoritmos O(n²)

El equipo empleado tiene las prestaciones:

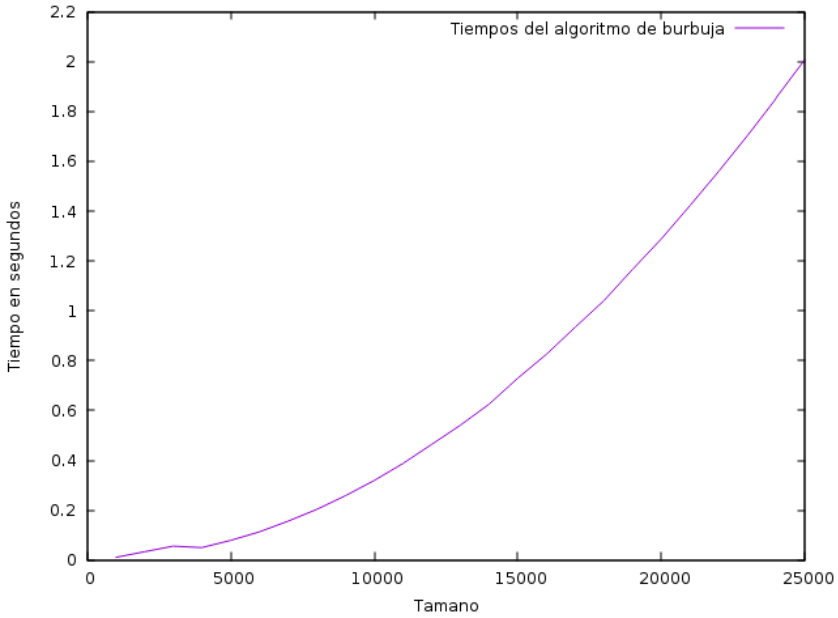
Procesador : Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
RAM: 8GB

Como podemos observar, el algoritmo más lento es del de burbuja con tiempos que superan el doble que los anteriores a medida que el tamaño aumenta. Entre inserción y selección no se aprecian diferencias significativas.

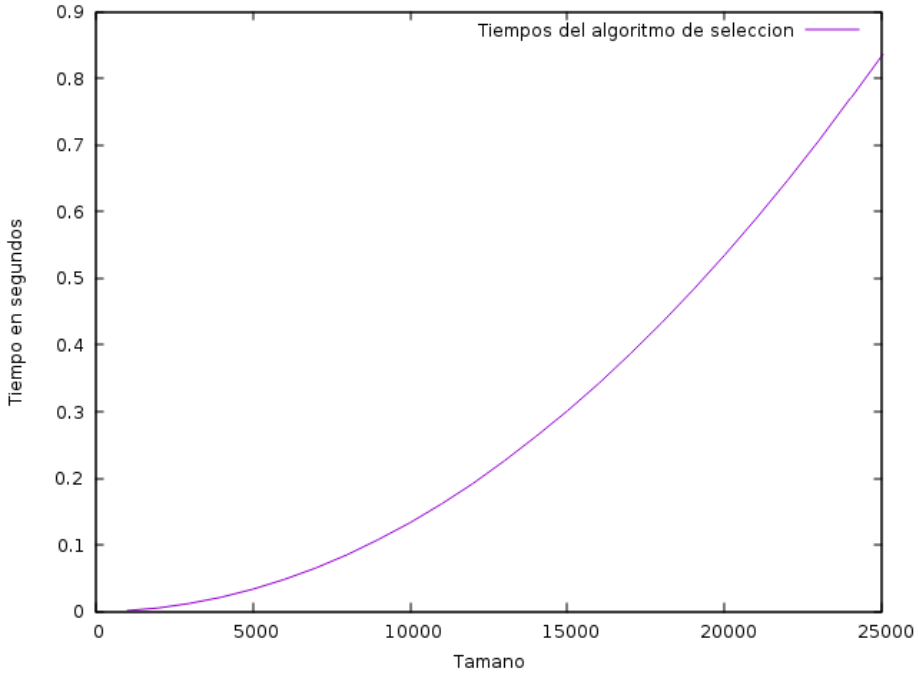
Resultado de las ejecuciones:

Tamaño	Burbuja	Selección	Inserción
1000	0.009918	0.00141602	0.001259
2000	0.024856	0.00542193	0.006108
3000	0.032287	0.012142	0.010358
4000	0.050257	0.02147	0.018124
5000	0.079186	0.033433	0.027975
6000	0.113281	0.048073	0.040309
7000	0.156463	0.065319	0.055311
8000	0.204092	0.085219	0.071996
9000	0.258562	0.108276	0.092417
10000	0.32005	0.133356	0.112756
11000	0.384988	0.161458	0.136253
12000	0.465959	0.192131	0.162078
13000	0.545303	0.226143	0.189631
14000	0.629492	0.262543	0.22141
15000	0.725537	0.301038	0.253554
16000	0.839554	0.342276	0.287533
17000	0.929566	0.386806	0.324087
18000	1.05121	0.43345	0.362635
19000	1.16358	0.482941	0.405348
20000	1.2925	0.535169	0.446799
21000	1.43041	0.58983	0.493072
22000	1.56164	0.647236	0.539583
23000	1.70039	0.707328	0.588799
24000	1.85597	0.770895	0.638557
25000	2.01983	0.835793	0.702854

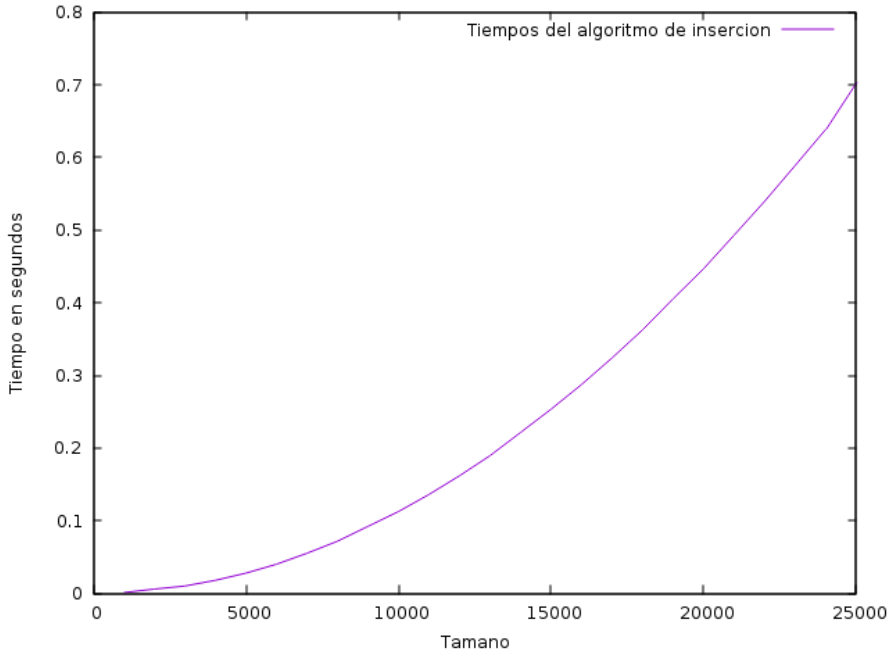
Gráfica Algoritmo Burbuja:



Gráfica Algoritmo Selección:



Gráfica Algoritmo Inserción:



Resultados Algoritmos O(nlog(n))

El equipo empleado tiene las prestaciones:

Procesador: Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz
RAM: 8 GB

Vemos una clara ventaja de utilizar Quicksort frente a los otros dos. Tanto Mergesort como Heapsort proporcionan tiempos similares, pero Quicksort es más rápido.

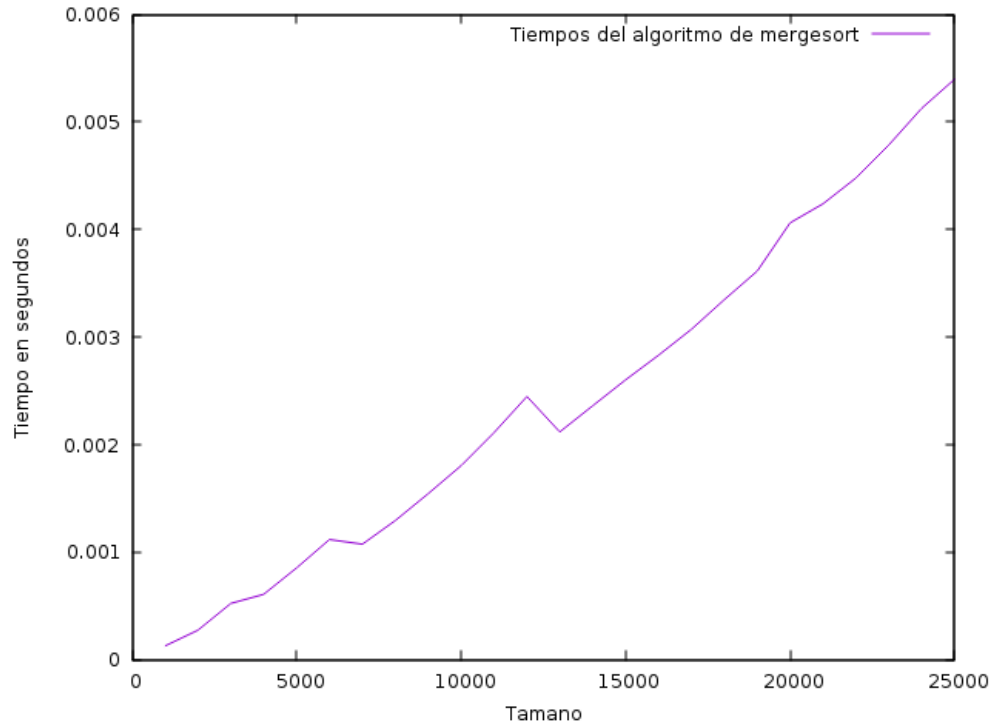
Por supuesto, los tres son mucho más rápidos que los tres anteriores, de hecho, como se puede ver, en este gráfico estamos trabajando con un tamaño mucho mayor y aun así conseguimos tiempos mucho mejores.

Resultado de las ejecuciones:

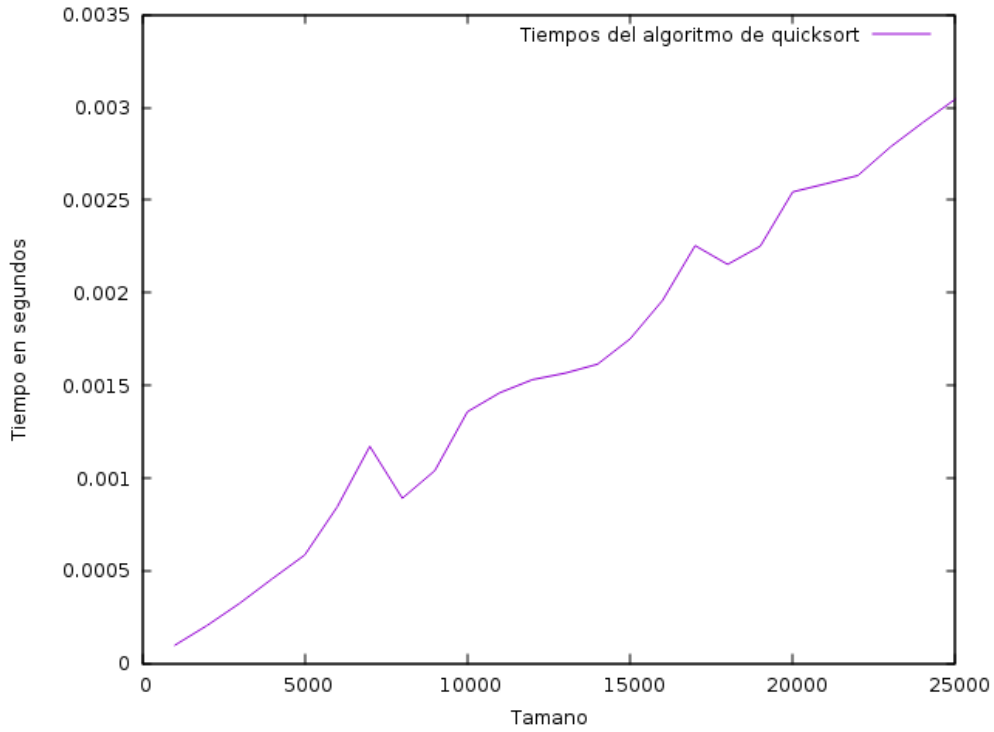
Tamaño	Mergesort	Quicksort	Heapsort
1000	0.000128245	9.7e-05	0.000131
2000	0.000274738	0.000205	0.000286
3000	0.000523958	0.000324	0.00044
4000	0.000607748	0.000456	0.000623
5000	0.000854426	0.000585	0.000832
6000	0.00111811	0.000844	0.00103
7000	0.00107502	0.001171	0.001426
8000	0.00129427	0.000891	0.001201
9000	0.00154438	0.00104	0.001362
10000	0.00180574	0.001358	0.001609
11000	0.002111	0.001461	0.001791
12000	0.002448	0.001531	0.002013
13000	0.002119	0.001566	0.002078
14000	0.002362	0.001614	0.002252
15000	0.002603	0.00175	0.002411
16000	0.002832	0.001959	0.002837
17000	0.003074	0.002254	0.00278
18000	0.003349	0.002153	0.002984
19000	0.003615	0.002251	0.003144
20000	0.004063	0.002544	0.003328
21000	0.004241	0.002587	0.003489

22000	0.00448	0.002632	0.003703
23000	0.004788	0.002786	0.003872
24000	0.005128	0.002919	0.004059
25000	0.005399	0.003045	0.004241

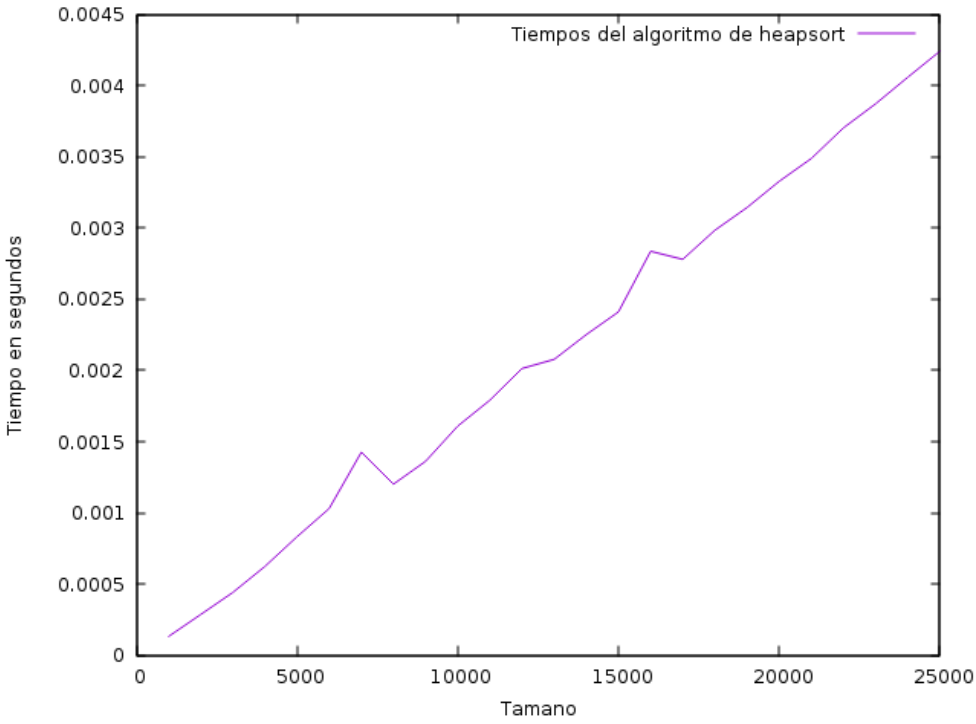
Gráfica Algoritmo Mergesort:



Gráfica Algoritmo Quicksort:



Gráfica Algoritmo Heapsort:



Resultados Algoritmos O(n³)

El equipo empleado tiene las prestaciones:

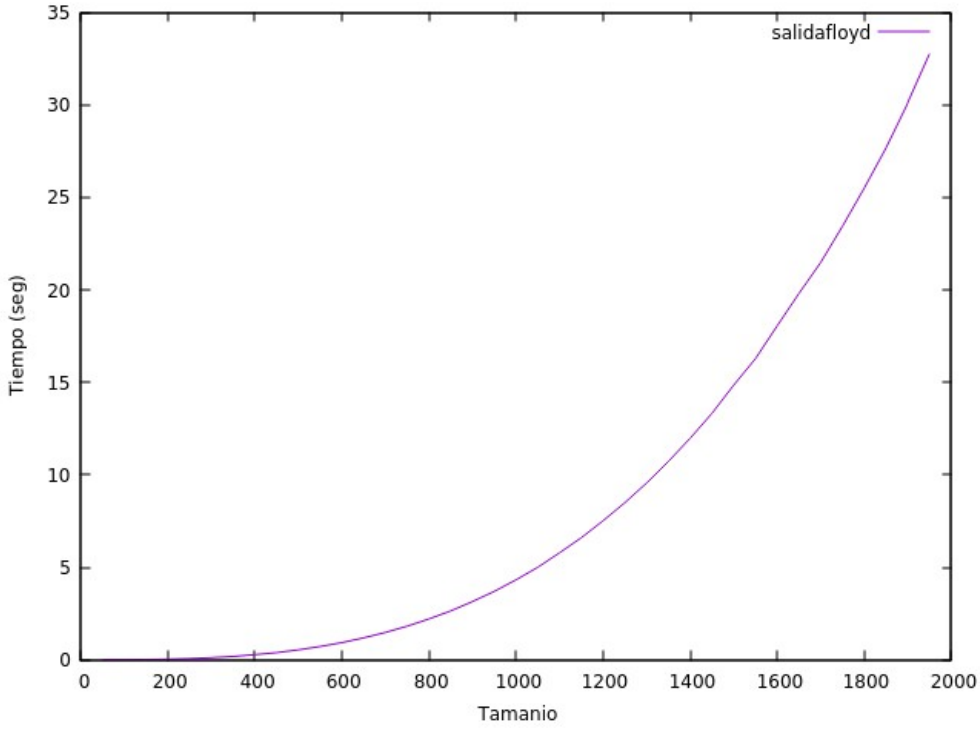
Procesador: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
RAM: 16 GB

Como podemos apreciar es muchísimo más lento que los algoritmos vistos hasta ahora, incluso para un tamaño mucho menor.

Resultado de las ejecuciones:

Tamaño	Floyd
40	0.001656
80	0.011078
120	0.028261
160	0.034968
200	0.045051
240	0.071313
280	0.113485

320	0.167839
360	0.23765
400	0.324802
440	0.437196
480	0.559942
520	0.716048
560	0.887893
600	1.09473
640	1.32449
680	1.5848
720	1.90991
760	2.21144
800	2.57918
840	2.98543
880	3.43734
920	3.92441
960	4.4626
1000	5.03838



Resultados Algoritmos $O(2^n)$

El equipo empleado tiene las prestaciones:

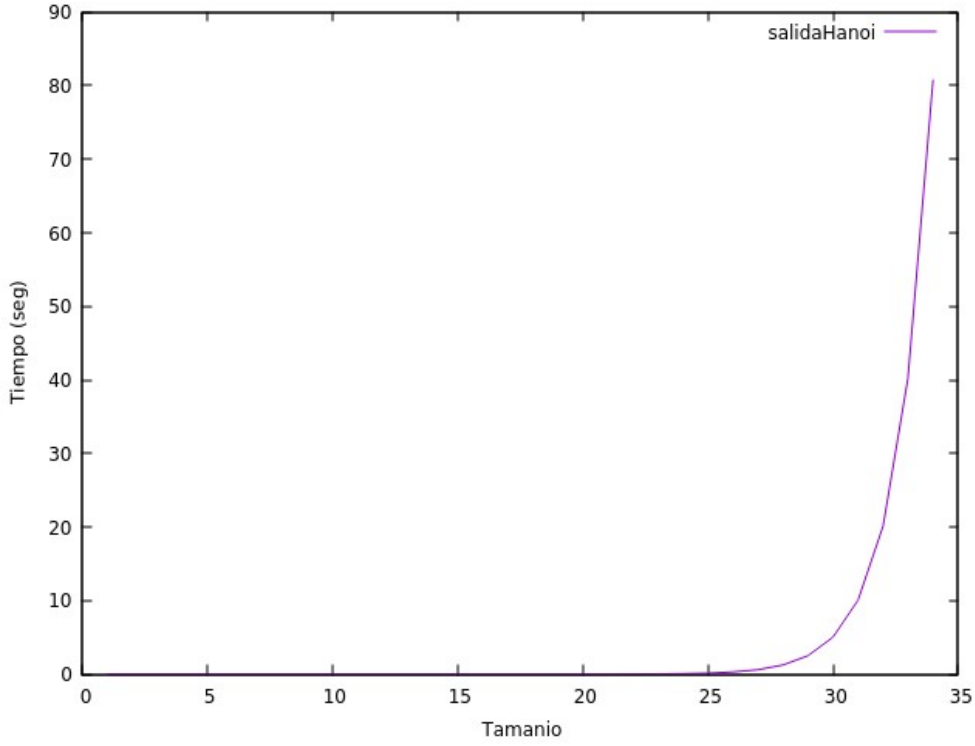
Procesador: Intel Core i5-7200U CPU @ 2.50 Ghz x 4
RAM: 7,7 GiB

En este caso estamos ante el algoritmo más lento de todos. Se nota su orden exponencial, que además se puede apreciar en la gráfica. Pasando de un determinado tamaño el tiempo se dispara.

Resultado de las ejecuciones:

Tamaño	**Hanoi**
10	3e-06
11	6e-06
12	1.1e-05
13	2.1e-05
14	4.2e-05
15	8e-05
16	0.000158
17	0.000328
18	0.000629
19	0.001297
20	0.002567
21	0.005341
22	0.009641
23	0.017469
24	0.034855
25	0.071372
26	0.141154
27	0.281754
28	0.553998
29	1.10018
30	2.20384
31	4.40097
32	8.8083
33	17.6036

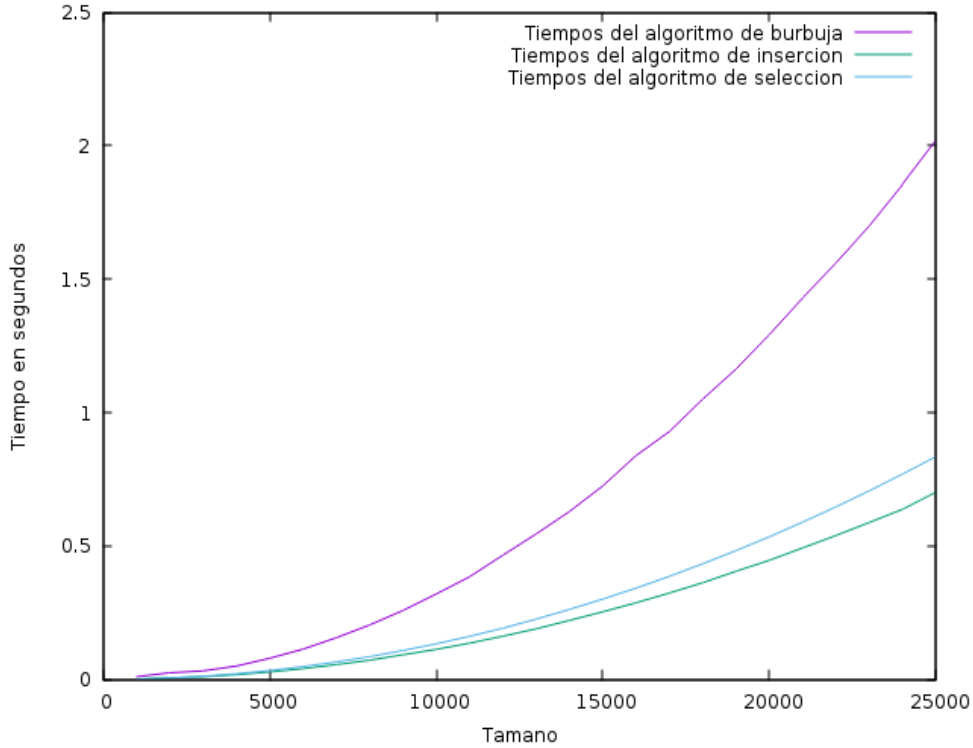
34	35.1961
35	70.3265
36	140.648



Ejercicio 2:

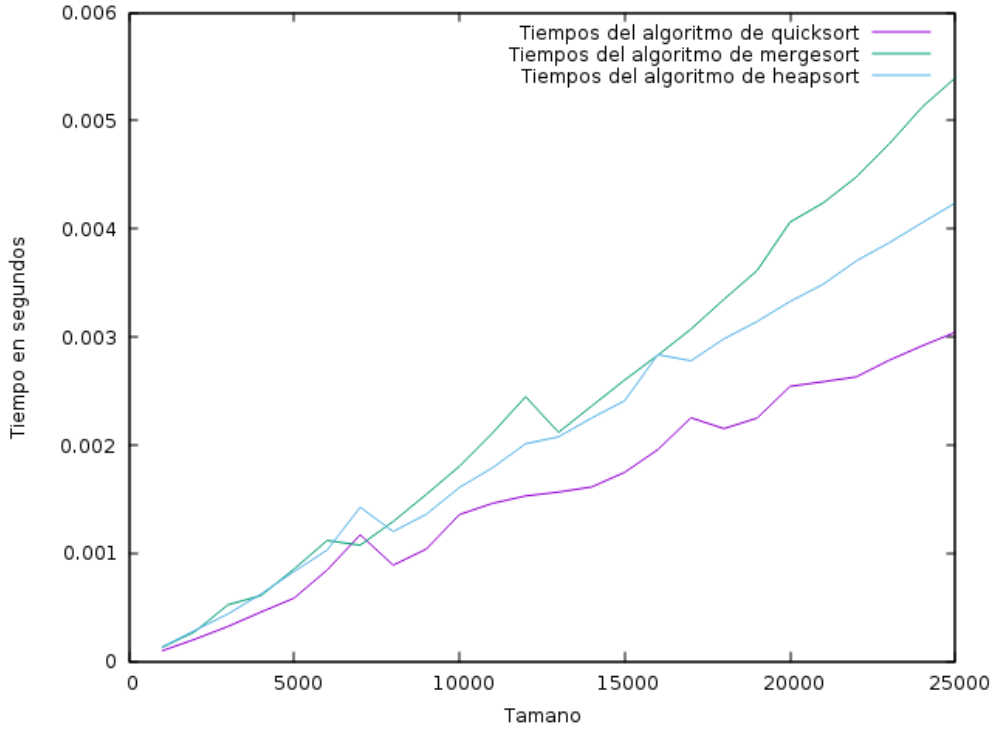
Gráficas conjuntas:

Gráfica Algoritmos O(n²)

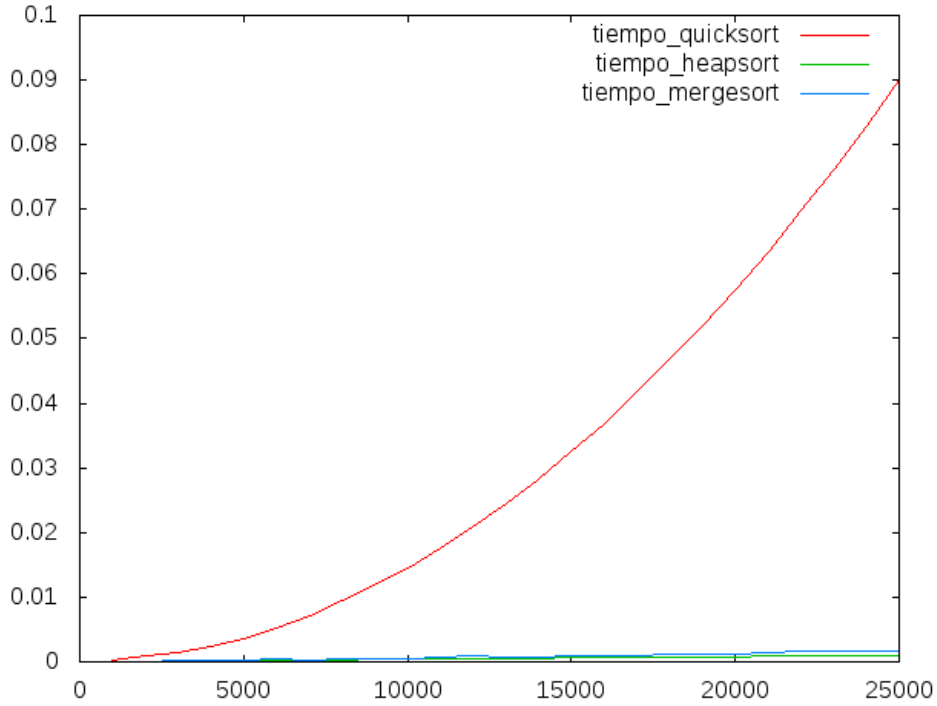


Gráfica Algoritmos O(nlog(n))

Caso general:



Peor de los casos:



Como podemos observar, cuando se da el peor de los casos para el algoritmo Quicksort, que es cuando la lista ya está ordenada, entonces su eficiencia es peor que Mergesort y Heapsort. Ésto es debido a que aunque en general Quicksort es el más rápido de los algoritmos de ordenación presentados, en el peor de los casos su eficiencia se aproxima más a $\boldsymbol{O}(N^2)$ que a $nlog(n)$.

Ejercicio 3:
Eficiencia Híbrida:

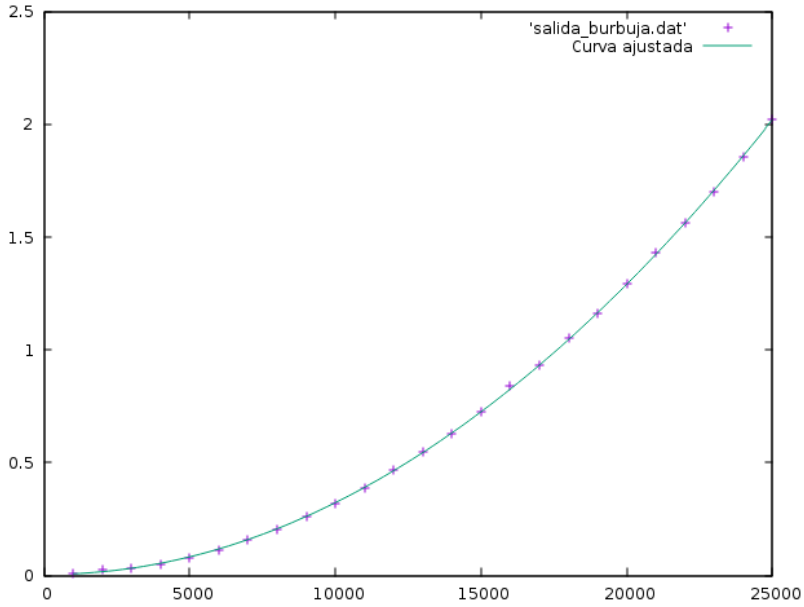
A la hora de hacer la regresión, lo que nos interesa son los valores de los coeficientes de regresión, así como la bondad del ajuste realizado. En principio, hemos utilizado la misma función de la eficiencia teórica para ajustar cada algoritmo, aunque también hemos probado otros para ver cómo de bueno es cada ajuste.

A continuación las gráficas junto con la función a la cual se ha ajustado y las constantes ocultas obtenidas por el programa *gnuplot* para cada ajuste:

Burbuja: función utilizada

f(x) = a0*x*x+a1*x+a2

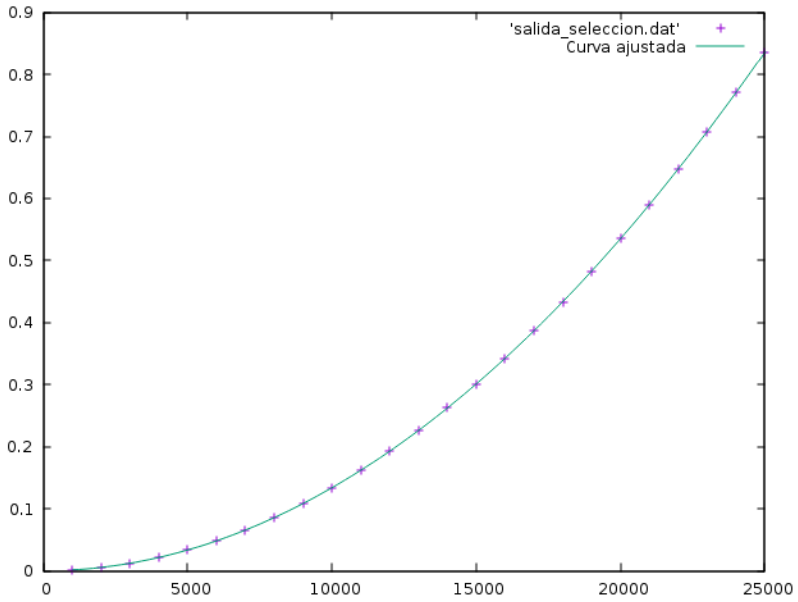
a0 = 3.25431e-09
a1 = -7.85455e-0
a2 = 0.00480453



Selección: función utilizada

f(x) = a0*x*x+a1*x+a2

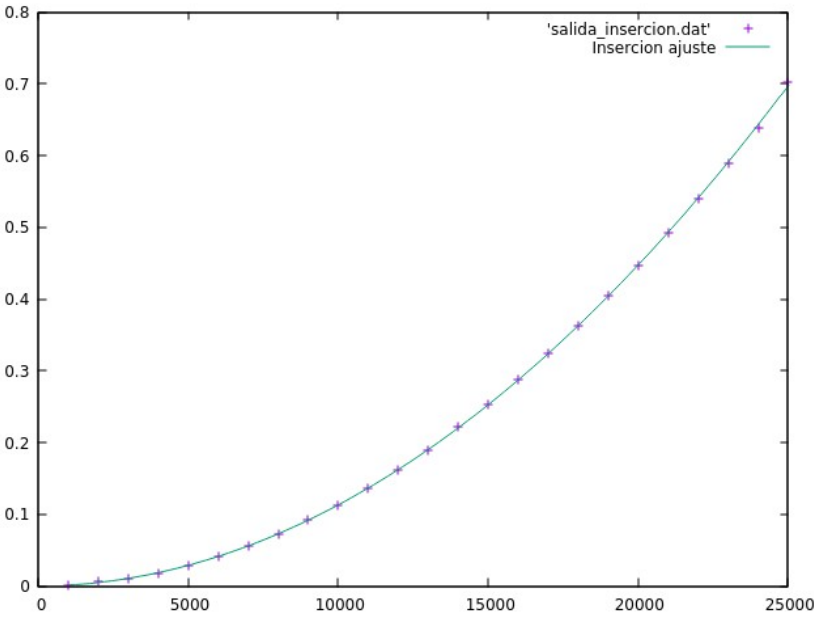
a0 = 1.33866e-09
a1 = -2.46031e-08
a2 = 4.2861e-05



Inserción: función utilizada

f(x) = a0*x*x+a1*x+a2

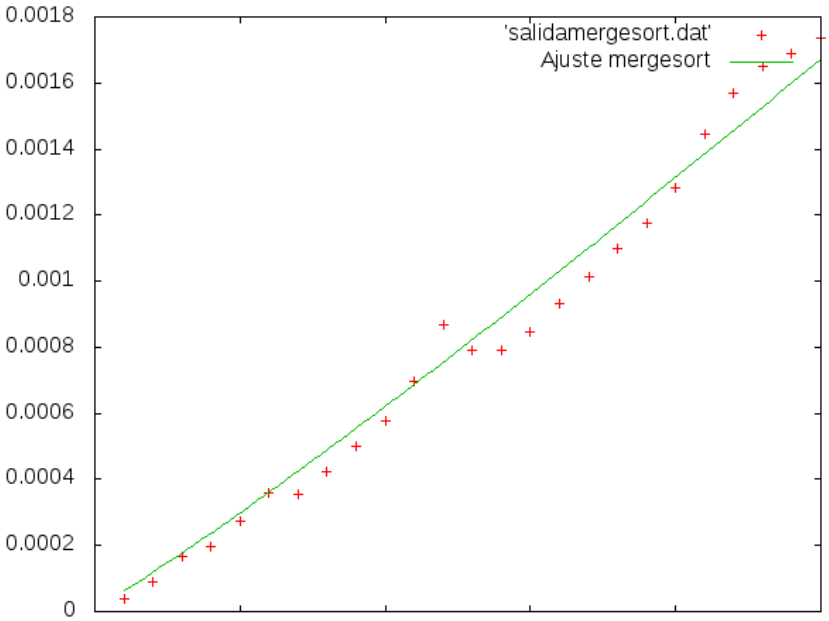
a0 = 1.10634e-09
a1 = 2.4927e-07
a2 = -0.000188397



Mergesort: función utilizada

f(x) = a0*x*log(x) + a1

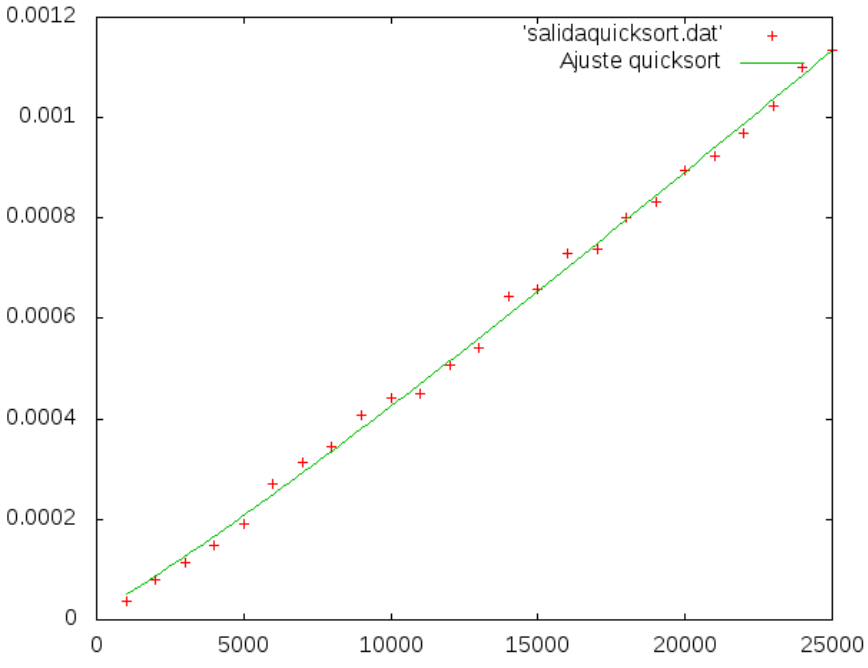
a0 = 6.5289e-09
a1 = 1.98674e-05



Quicksort: función utilizada

f(x) = a0*x*log(x) + a1

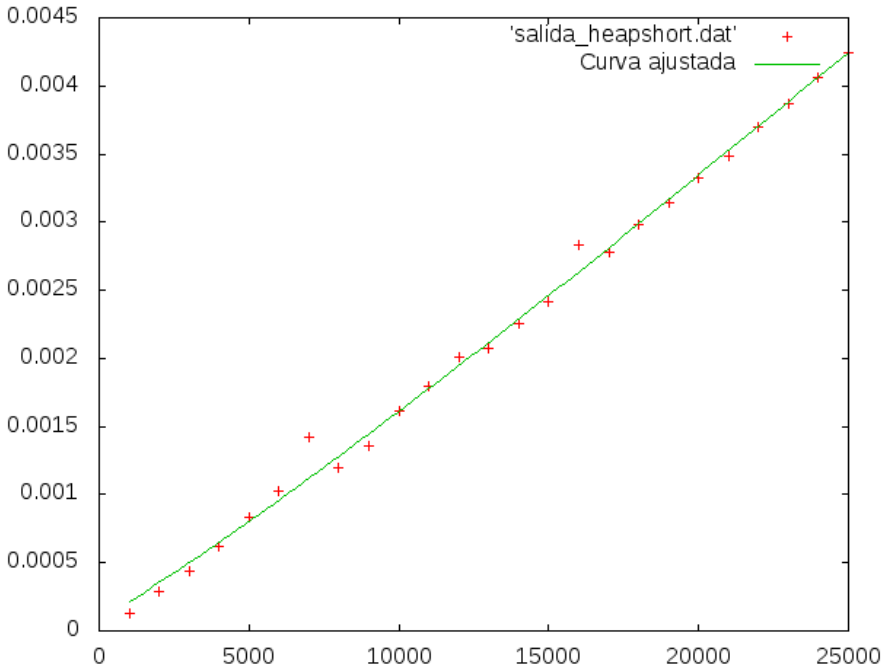
a0 = 4.40174e-09
a1 = 1.98678e-05



Heapsort: función utilizada

f(x) = a0*x*log(x) + a1

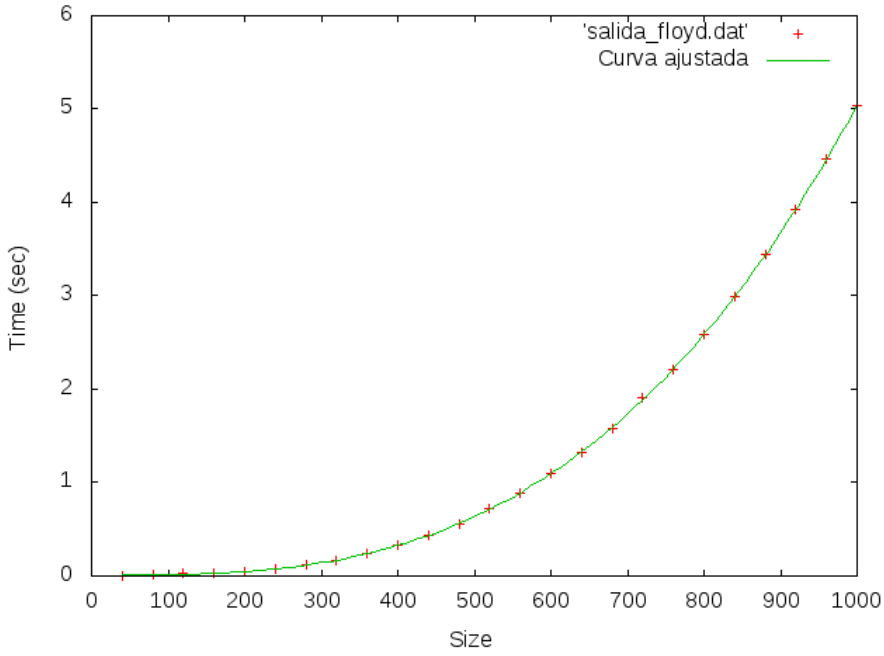
a0 = 1.63609e-08
a1 = 0.000104941



Floyd: función utilizada

f(x) = a0*x*x*x+a1*x*x+a2*x+a3

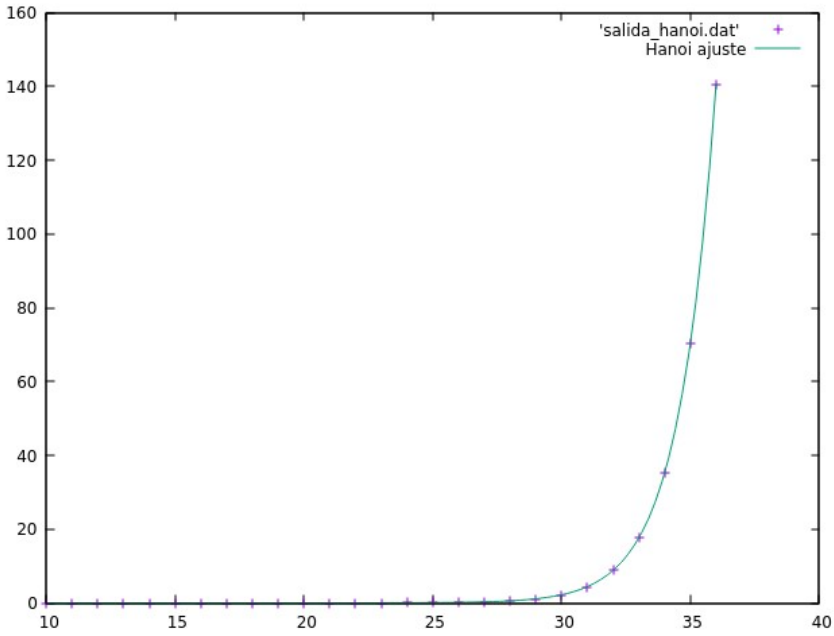
a0 = 4.98387e-09
a1 = 8.4389e-08
a2 = -4.09129e-05
a3 = 0.0113909



Hanoi: función utilizada

f(x) = 2**(a0*x+a1)

a0 = 0.999508
a1 = -28.8464



Como podemos observar la bondad de los ajustes en cada caso es bastante buena, la única en la que no coinciden exactamente todos los puntos con la función ajustada es en el caso *n*log(x)*, pero esto es debido a, como ya comentamos en los ejercicios anteriores, el bajo rango de los valores que al ser utilizados por un algoritmo con un orden de eficiencia tan rápido, genera picos en la gráfica.

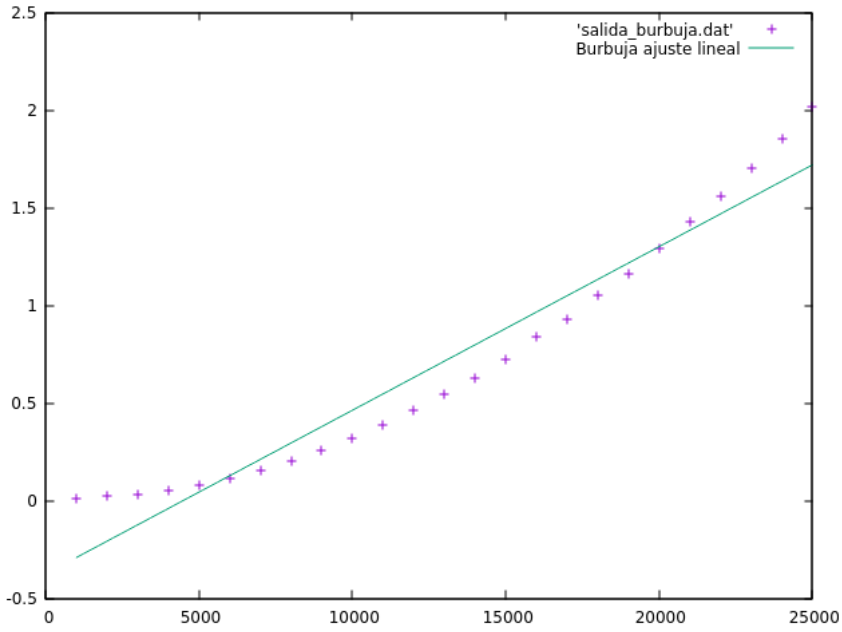
Ajustes distintos a los teóricos:

A continuación se presentan otros ajustes utilizados diferentes a los que se presentan teóricamente para algunos algoritmos determinados.

Burbuja: función utilizada

f(x) = a0*x+a1

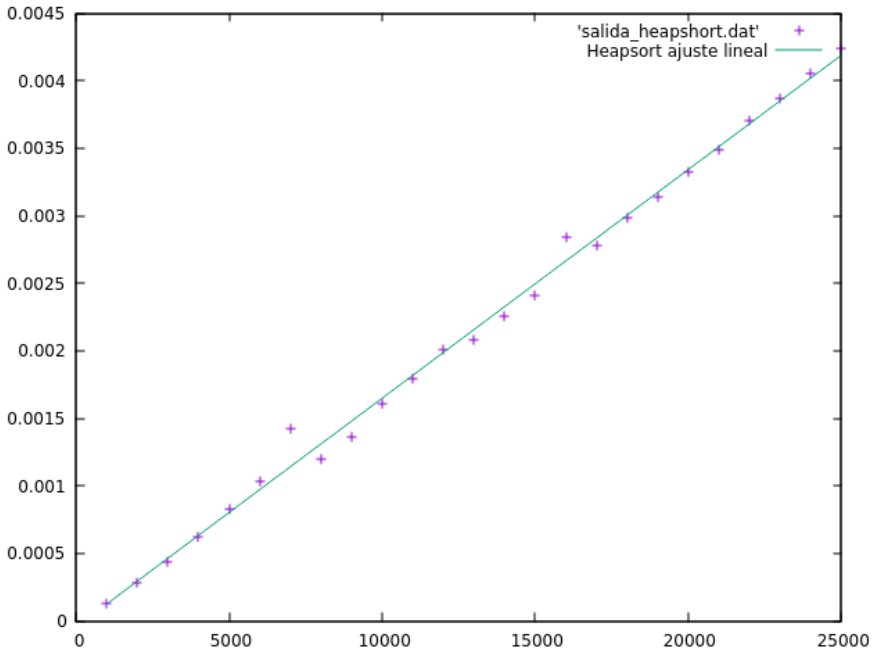
a0 = 8.38265e-05
a1 = -0.375949



Heapsort: función utilizada

f(x) = a0*x+a1

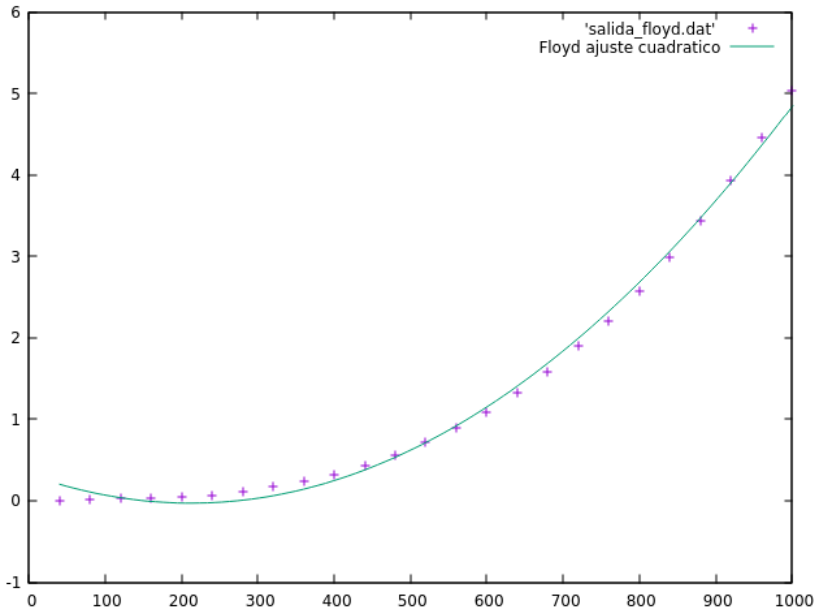
a0 = 1.69433e-07
a1 = -4.575e-05



Floyd: función utilizada

f(x) = a0*x*x+a1*x+a2

a0 = 7.85923e-06
a1 = -0.00333904
a2 = 0.324873



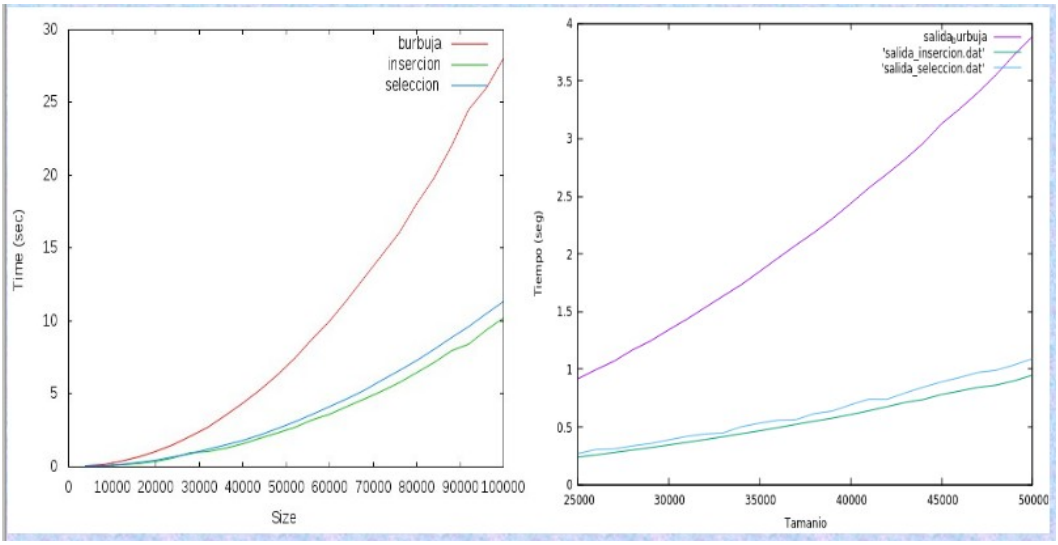
Podemos ver que efectivamente estos casos son mucho peores:

En el caso del algoritmo de burbuja, los datos no se ajustan casi nada utilizando una función de grado 1, aunque en el caso del Heapsort el ajuste conseguido con una función de grado 1, es muy similar a la obtenida con logaritmos. Seguramente sea porque la escala de datos utilizada ha sido demasiado pequeña para apreciar la tendencia de los datos a asemejarse a una función de la forma *n*log(n)* más que a una lineal.

Finalmente, en el caso de Floyd, aunque no se ajusta mal del todo, la aproximación con una función de grado 2 es mucho peor que la usada inicialmente de grado 3.

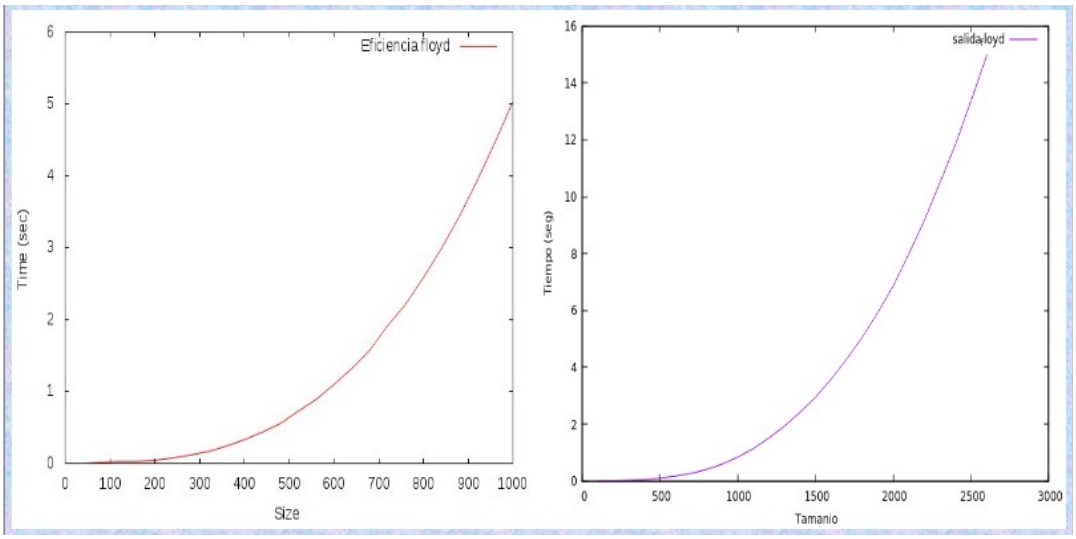
Ejercicio 3:
Comparaciones y optimización:

Ya mostramos en las diapositivas que las gráficas obtenidas para los valores de dos componentes del mismo grupo, dependen de muchos factores. Nunca vamos a obtener dos gráficas idénticas ni siquiera usando el mismo ordenador para dos ejecuciones distintas de un mismo algoritmo.



Procesador: Intel® Core™ i7-4720HQ CPU @ 2.60GHz x4 (Código sin optimizar)

Procesador: Intel® Core™ i5-7200U CPU @ 2.50GHz x4 (Optimización O2)



Procesador: Intel® Core™ i7-4720HQ CPU @ 2.60GHz x4 (Código sin optimizar)

Procesador: Intel® Core™ i5-7200U CPU @ 2.50GHz x4 (Optimización O2)

Como podemos ver la optimización del código influye notablemente en la eficiencia de nuestros programas, así como el procesador de cada máquina, ya que un i5 de 7th generación supera al i7 de 4th y por tanto, los resultados del primero serán mucho más rápidos que los del segundo.

A modo de conclusión, no solo es importante que el código de un algoritmo sea eficiente y lo más rápido posible, muchas veces tenemos que tener en cuenta el objetivo final de ese algoritmo, así como la máquina en la que se va a utilizar, la optimización con la que se va a compilar y un largo etc. de factores que influyen notablemente en los resultados de un programa.