

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.....pág 3

Ejercicio 1:

Resultados ejecuciones.....pág 3-11

Gráficas individuales.....pág 3-11

Ejercicio 2:

Gráficas conjuntas.....pág 11-13

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 esta 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 $O(n^2)$

El equipo empleado tiene las prestaciones:

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

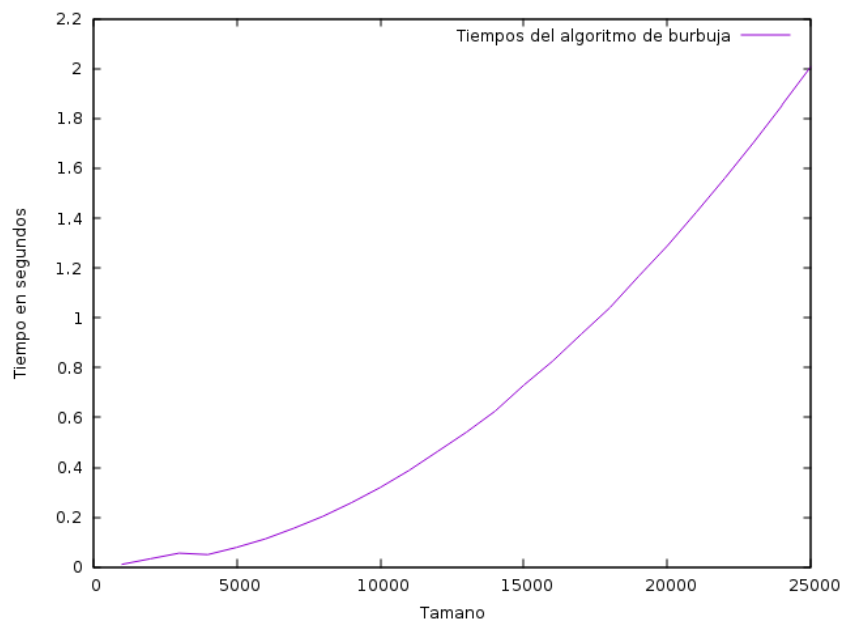
RAM: 8GB

Como podemos observar, el algoritmo más lento es el 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.

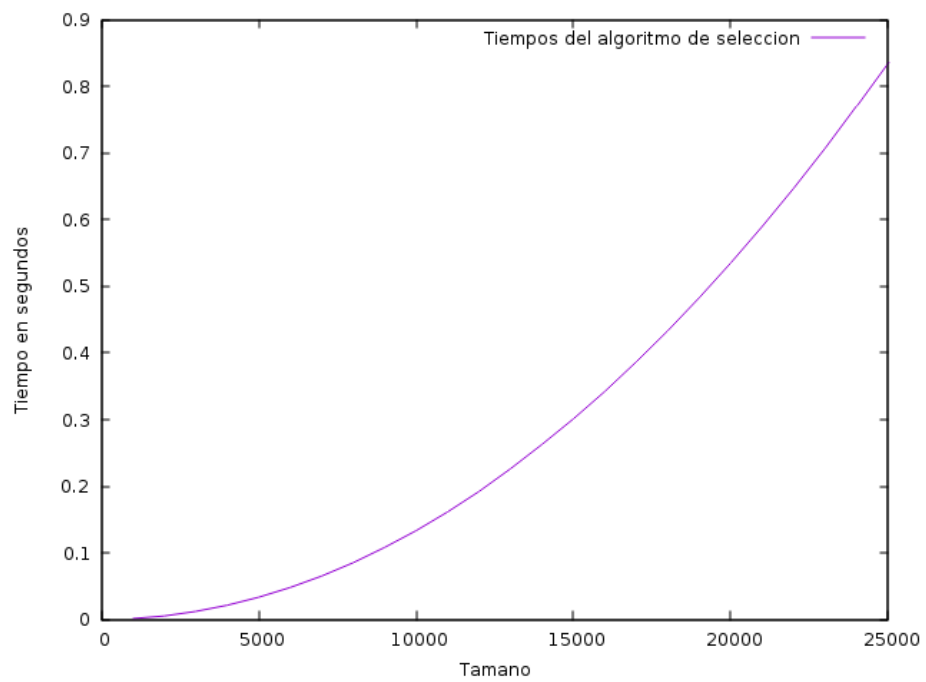
Resultado de las ejecuciones:

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

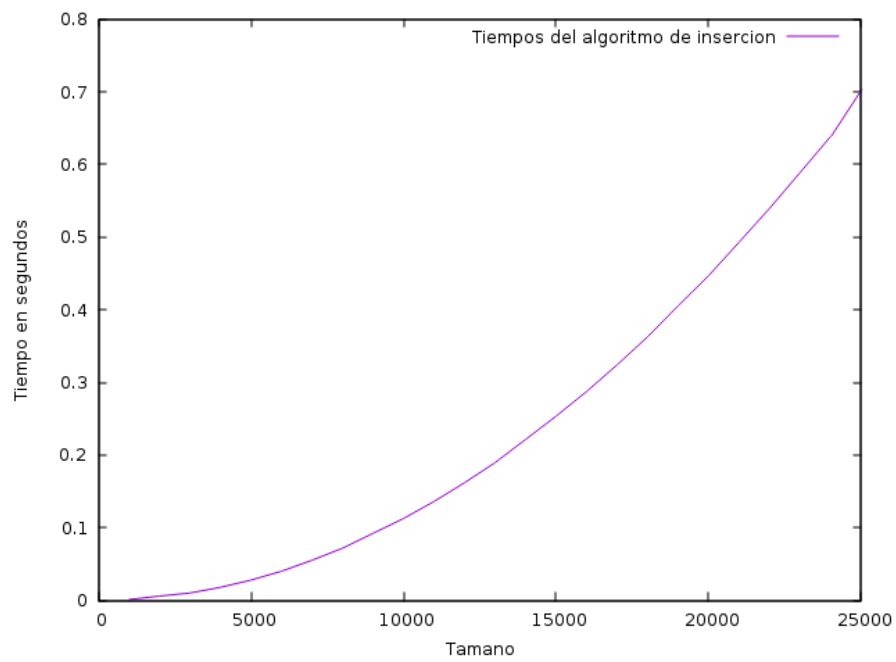
Gráfica Algoritmo Burbuja:



Gráfica Algoritmo Selección:



Gráfica Algoritmo Inserción:



Resultados Algoritmos $O(n \log(n))$

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.

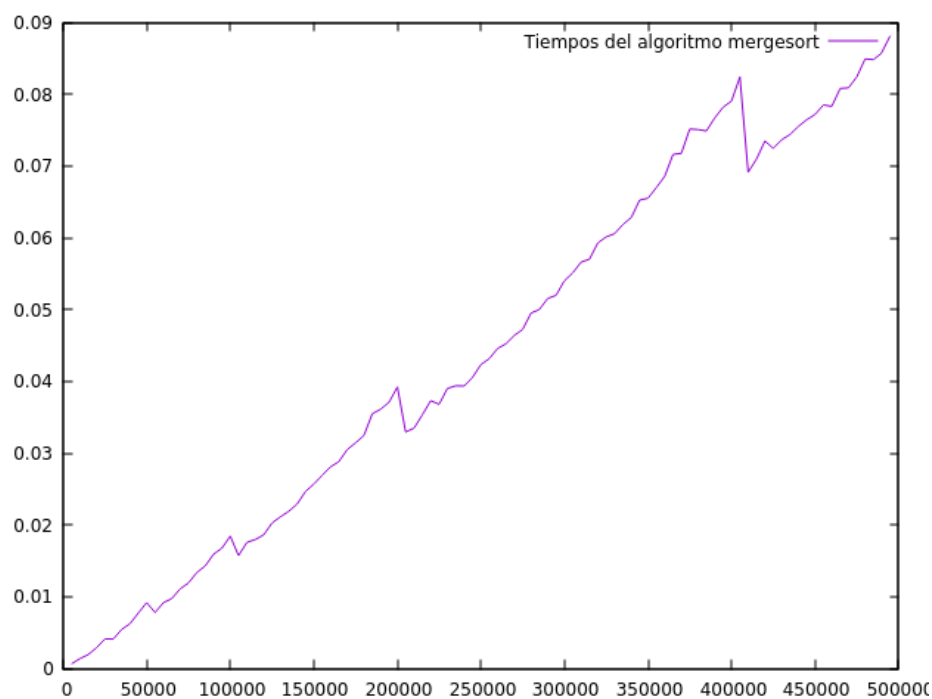
Por supuesto, los tres son mucho más rápidos que los tres anteriores, de hecho, como se puede ver, en este gráfico estamos trabajando con un tamaño mucho mayor y aun así conseguimos tiempos mucho mejores.

Resultado de las ejecuciones:

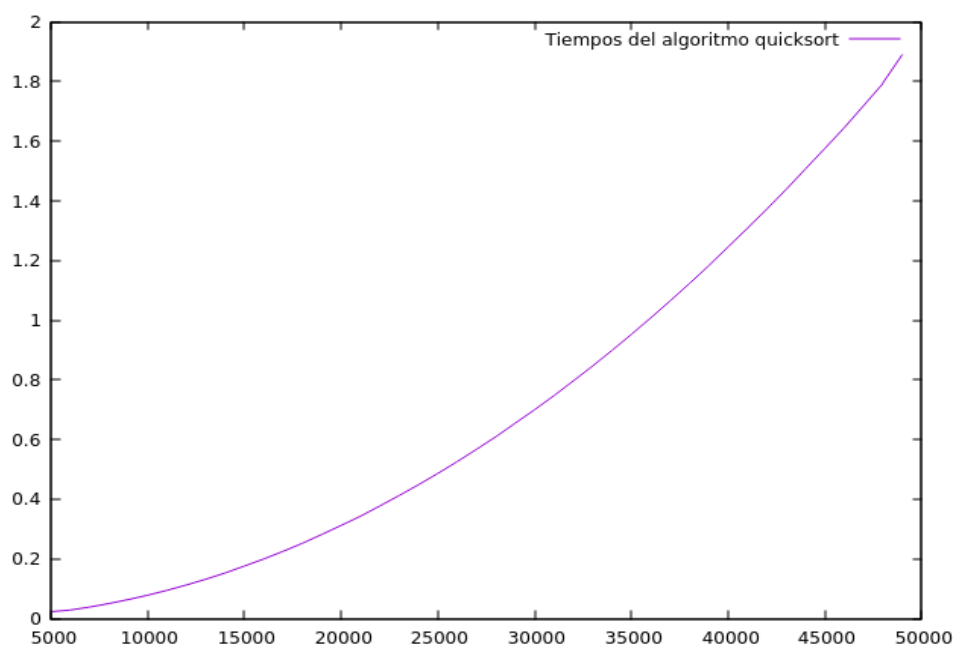
Tamaño	Mergesort	Quicksort	Heapsort
1000	3.8878e-05	0.000292	3.64e-05
2000	8.8723e-05	0.0010498	7.69e-05
3000	0.000167864	0.0013523	0.0001636
4000	0.000194886	0.0023551	0.0003
5000	0.000272133	0.0036459	0.0003136
6000	0.000357258	0.0052523	0.0002378
7000	0.000353615	0.0071011	0.0002808
8000	0.000423233	0.0094129	0.0003249
9000	0.000498435	0.0117933	0.0003693
10000	0.000578286	0.0146035	0.0003991
11000	0.000698	0.0174775	0.0004835
12000	0.000869	0.0209144	0.0004805
13000	0.00079	0.0243907	0.0005247
14000	0.000789	0.0282944	0.000576
15000	0.000846	0.0325595	0.0006045
16000	0.000931	0.0368279	0.000648
17000	0.001014	0.041702	0.0006952
18000	0.001097	0.0469084	0.0007284
19000	0.001175	0.0520349	0.0007744
20000	0.001283	0.0575794	0.0008225
21000	0.001447	0.0635084	0.0008642

22000	0.001567	0.0699285	0.0009109
23000	0.001651	0.0761016	0.0009425
24000	0.001688	0.0828021	0.0009938
25000	0.001738	0.090069	0.0010341

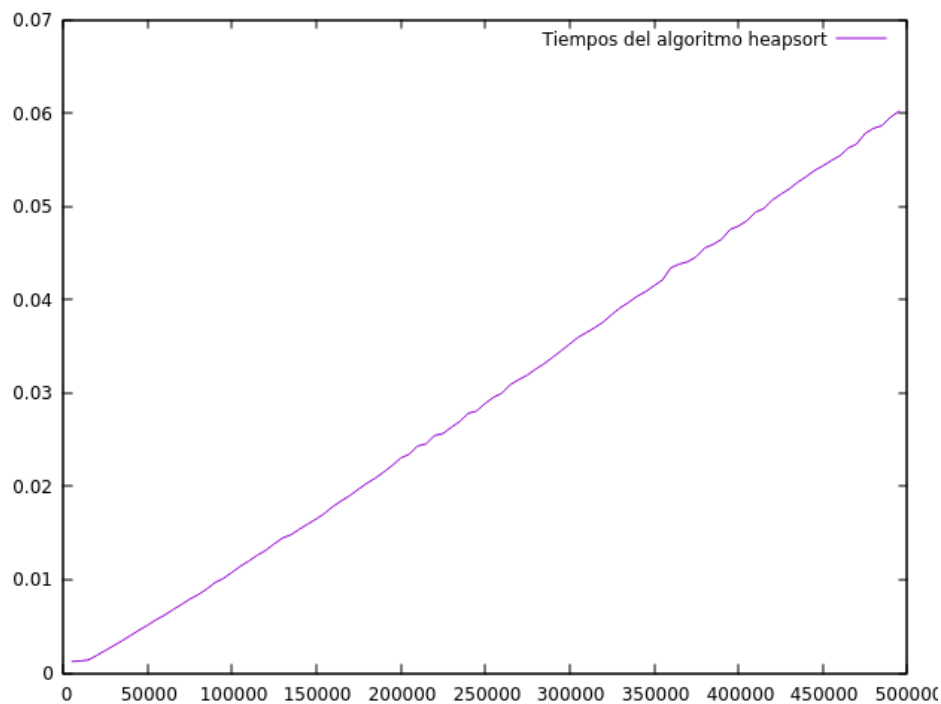
Gráfica Algoritmo Mergesort:



Gráfica Algoritmo Quicksort:



Gráfica Algoritmo Heapsort:



Resultados Algoritmos $O(n^3)$

El equipo empleado tiene las prestaciones:

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

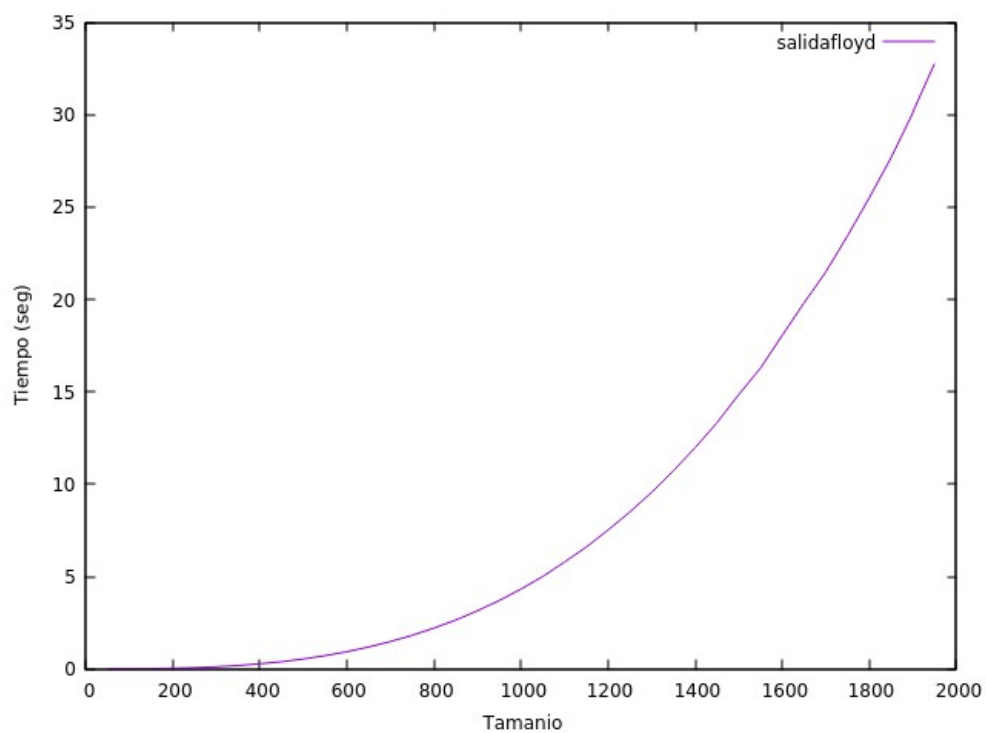
RAM: 16 GB

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

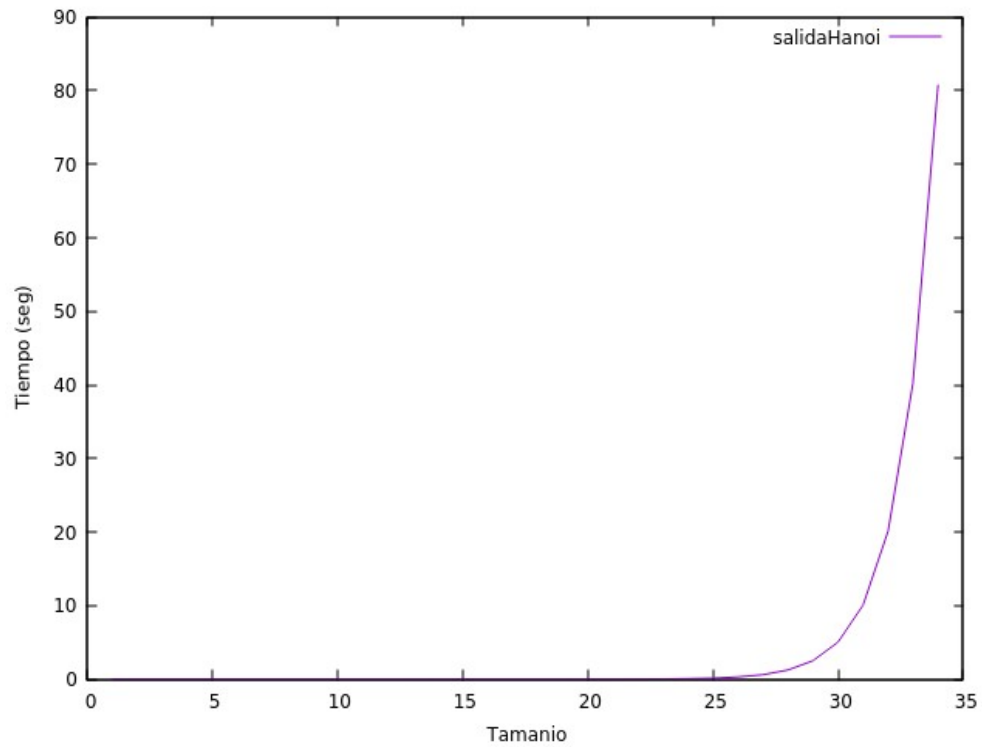
RAM: 7,7 GiB

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

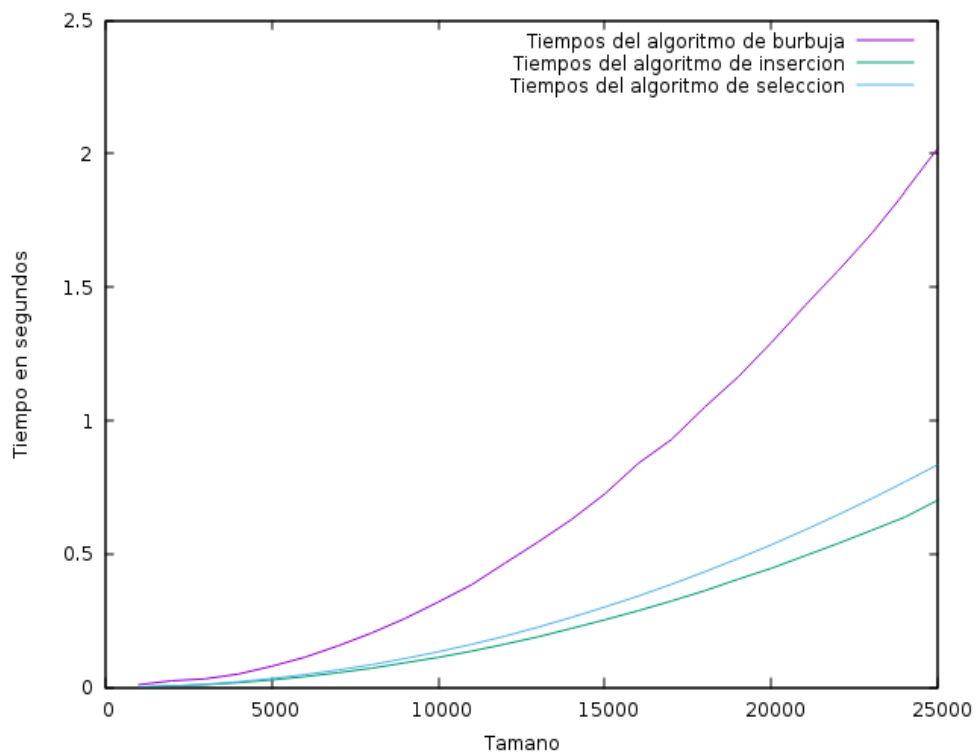
34	35.1961
35	70.3265
36	140.648



Ejercicio 2:

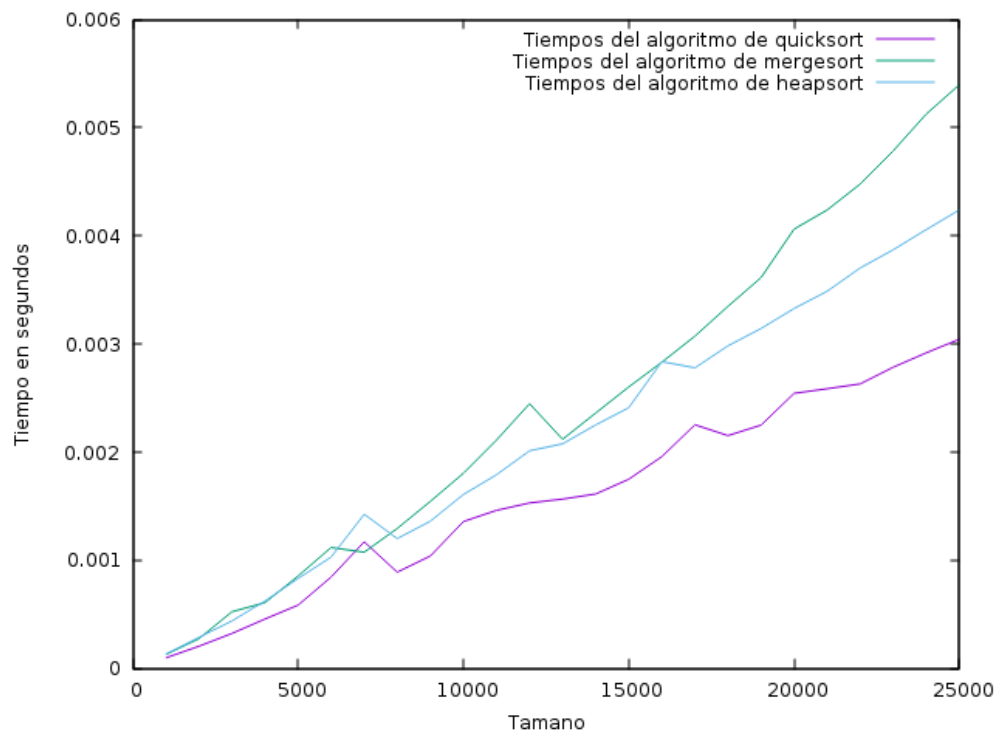
Gráficas conjuntas:

Gráfica Algoritmos $O(n^2)$

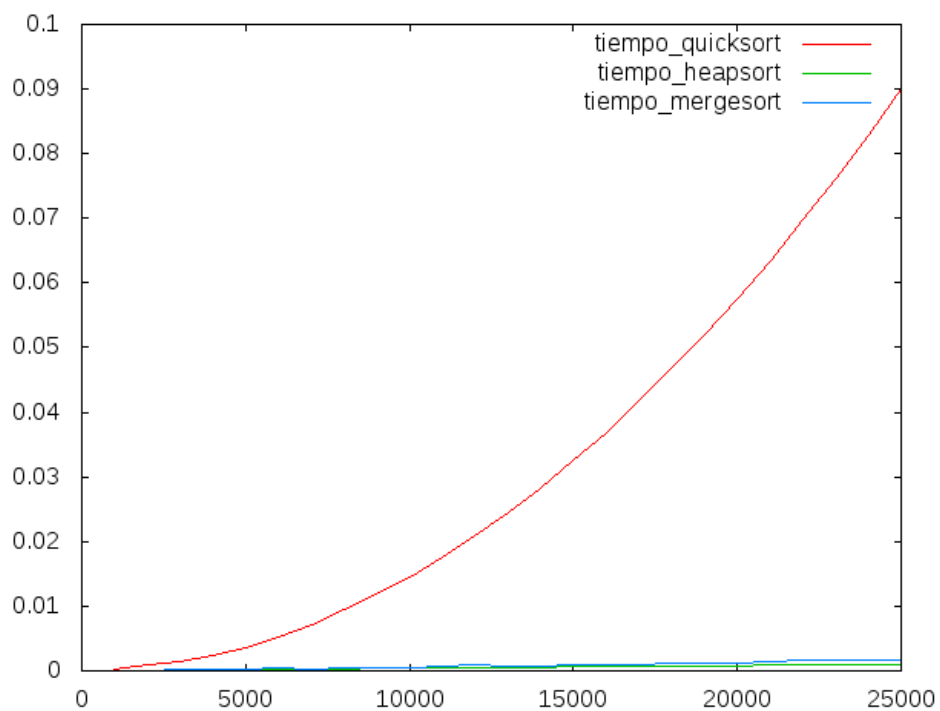


Gráfica Algoritmos $O(n \log(n))$

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 $O(N^2)$ que a $n\log(n)$.