Memoria de la práctica 1: Análisis de Eficiencia de Algoritmos

Grupo A1

Realizado por:

Alejandro Sánchez Molina
Jorge García Moreno
David López Maldonado
Jose Manuel Rodríguez Calvo
Jesús Baeza Álvarez

Índice

Tareas realizadas:

1.-Introducción:

En esta práctica hemos realizado los ejercicios del final del guión a partir de los códigos burbuja, inserción, selección, mergesort, quicksort, heapsort ,fibonacci y Floyd proporcionados en la plataforma decsai.

Para el ejercicio 1 hemos llevado a cabo un análisis de eficiencia empírica, para ello medimos el tiempo para cada tamaño dado en las entradas que en el caso de algoritmos de ordenación viene dado por el número de componentes del vector y en el caso del algoritmo de Floyd el tamaño ha sido el número de nodos del grafo. (Apartados 2.1 y 2.2)

En el ejercicio 2 gracias al software gnuplot generamos un gráfico a partir de las tablas comparando los tiempos de los algoritmos. Incluimos una gráfica específica para los de ordenación. (Apartados 3.1, 3.2, 3.3, 3.4, 3.5)

En el ejercicio 3 proporcionamos la eficiencia híbrida gracias de nuevo al software gnuplot. Para ello necesitamos conocer el valor de las constantes asociadas a cada término de la expresión obtenida al calcular el tiempo de ejecución de un algoritmo. (Apartado 4)

Por último para el ejercicio 4 hemos realizado todos los análisis de cada uno de los algoritmos en dos ordenadores distintos con el objetivo de comparar cada resultado dependiendo de las características de los ordenadores. (Apartado 5)

2.-Cálculo de la eficiencia empírica

2.1.- Cálculo de los datos

Para el cálculo de datos hemos medido los recursos empleados (el tiempo) para cada tamaño dado de las entradas. Modificamos el fichero definiendo dos variables “tantes” y “tdespues” y añadiendo el código necesario para la obtención del tiempo según la entrada. Hacemos uso del macro proporcionado en el guión para ayudarnos a obtener los datos necesarios para el análisis. Por último para generar las tablas y los gráficos, a partir de los ficheros de salida en los que la primera columna corresponde con el tamaño del problema y la segunda al tiempo, hacemos uso del software gnuplot.

2.2.-Tablas de datos de distinto orden

→ Algoritmos orden n^2 (Burbuja, inserción, selección):

ALGORITMOS ORDEN O(n^2)			
TAMAÑO	BURBUJA	INSERCIÓN	SELECCIÓN
1000	0.003474	0.002273	0.002269
2000	0.013642	0.007475	0.008776
3000	0.030865	0.01713	0.019746
4000	0.06077	0.030512	0.034904
5000	0.094498	0.051514	0.054466
6000	0.143243	0.070034	0.078158
7000	0.198503	0.093721	0.105868
8000	0.266597	0.116877	0.137996
9000	0.344918	0.148621	0.174752
10000	0.42949	0.182247	0.215889
11000	0.530335	0.217741	0.260781
12000	0.648409	0.257772	0.311337
13000	0.762505	0.301912	0.366014
14000	0.895291	0.352296	0.424018
15000	1.03	0.402478	0.48658
16000	1.18292	0.458153	0.554026
17000	1.34286	0.52831	0.625542
18000	1.51497	0.592809	0.715723
19000	1.69277	0.650443	0.783128
20000	1.89772	0.716637	0.874388
21000	2.11058	0.790636	0.956913
22000	2.31255	0.866258	1.05346
23000	2.57535	0.995187	1.14626
24000	2.76332	1.02884	1.2481
25000	3.00934	1.11482	1.3532

Como podemos observar en la tabla, el algoritmo de inserción es el más eficiente con respecto a los demás seguidos del de selección y a continuación por el de burbuja. Los tamaños de entrada para estos algoritmos son mucho menores que los de los algoritmos de orden nlogn ya que el tiempo de ejecución comenzaba a ser demasiado largo.

→ Algoritmos orden nlogn (mergesort, heapsort, quicksort)

ALGORITMOS ORDEN O(n log n)			
TAMAÑO	MERGESORT	QUICKSORT	HEAPSORT
400000	0.166124	0.097076	0.15
800000	0.345536	0.205298	0.319394
1200000	0.487227	0.309781	0.494658
1600000	0.719383	0.429227	0.69876
2000000	0.806097	0.538046	0.90567
2400000	1.01498	0.651421	1.11594
2800000	1.24496	0.765841	1.33647
3200000	1.5106	0.883142	1.56027
3600000	1.46806	1.00774	1.7613
4000000	1.67631	1.13079	1.99007
4400000	1.91485	1.23534	2.22281
4800000	2.11166	1.35673	2.4737
5200000	2.34108	1.48331	2.68743
5600000	2.58193	1.60899	2.96379
6000000	2.82843	1.73237	3.18594
6400000	3.10898	1.85956	3.45199
6800000	2.85215	1.98569	3.71626
7200000	3.06476	2.0898	3.93041
7600000	3.27799	2.22835	4.16972
8000000	3.49011	2.32794	4.45239
8400000	3.73093	2.45234	4.61333
8800000	3.92282	2.58512	4.89892
9200000	4.14908	2.71357	5.24467
9600000	4.37556	2.86652	5.43264
10000000	4.66431	2.95001	5.6845

El algoritmo quicksort es más eficiente que el mergesort seguido por el heapsort. Estos algoritmos de orden nlogn , como podemos comprobar, son mucho más eficientes que los algoritmos de orden n^2.

→ Algoritmos orden n^3 (floyd)

ALGORITMOS ORDEN O(n^3)	

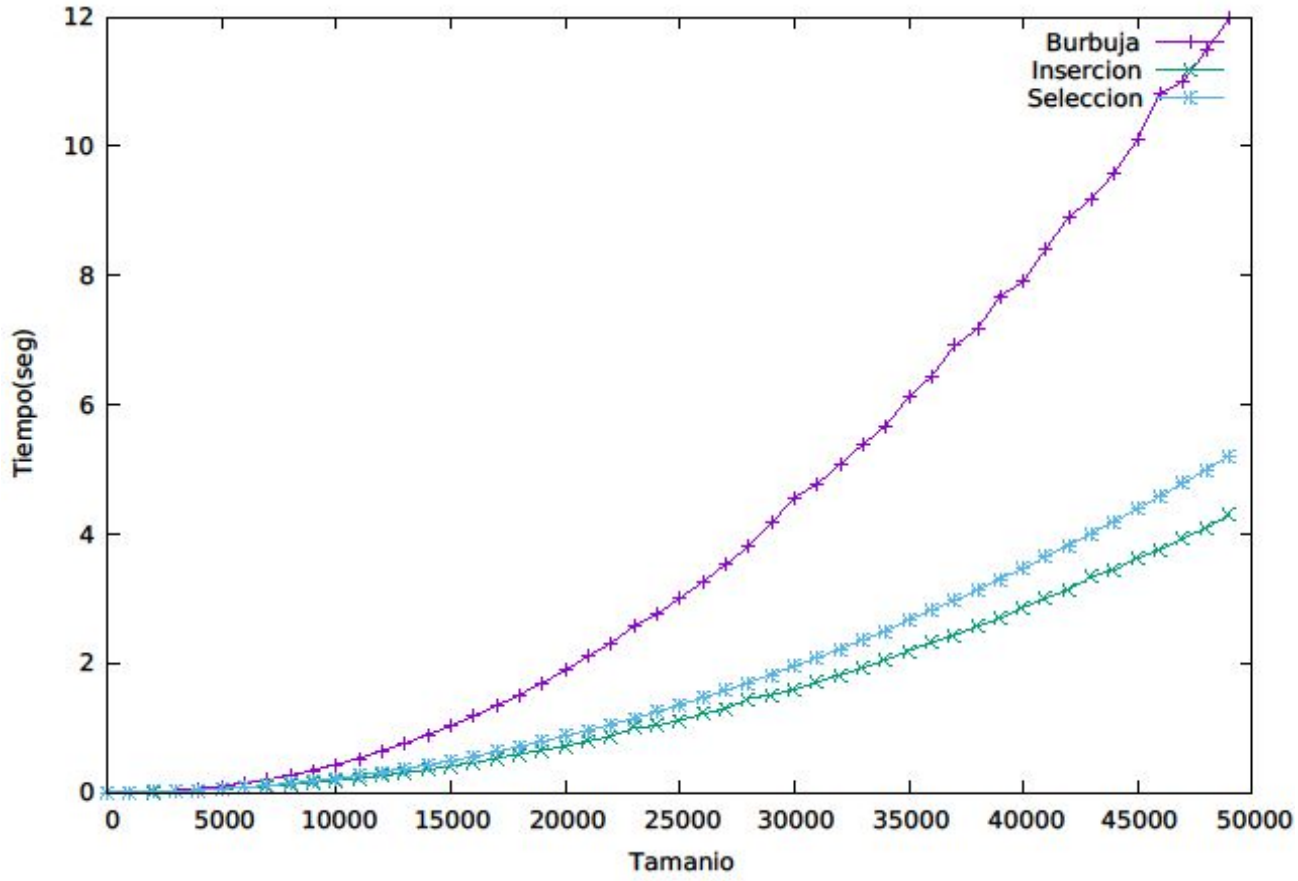
TAMAÑO	FLOYD
0	0.000338
32	0.000338
64	0.002549
96	0.008875
128	0.020695
160	0.038876
192	0.066284
224	0.104604
256	0.15652
288	0.219608
320	0.298932
352	0.399858
384	0.519754
416	0.661122
448	0.821598
480	1.01143
512	1.22582
544	1.47021
576	1.74266
608	2.05254
640	2.38865
672	2.77026
704	3.18981
736	3.18981
768	4.14594

→ Algoritmos orden (((1+ √5/2)^n)) (fibonacci)

ALGORITMOS ORDEN O((1+√5/2)^n)	
TAMAÑO	FIBONACCI
21	0.000124
22	0.0002
23	0.000344
24	0.00052
25	0.00084
26	0.001629
27	0.002291
28	0.00357
29	0.005771
30	0.009327
31	0.016051
32	0.024939
33	0.039914
34	0.064544
35	0.103813
36	0.167294
37	0.269919
38	0.43716
39	0.706828
40	1.14355
41	1.84923
42	2.99506
43	4.84251
44	7.92531
45	12.6945

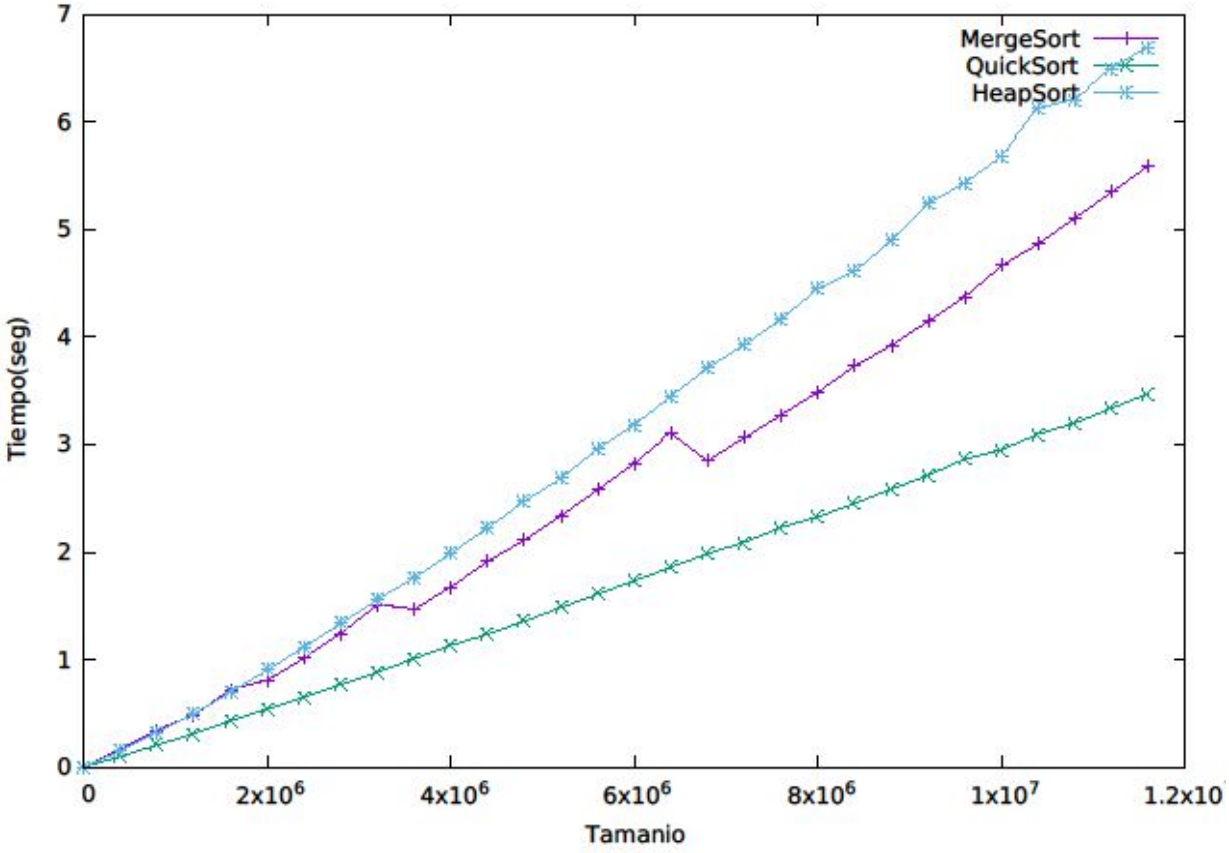
3-.Gráficas comparativas de algoritmos

3.1.- Gráfica orden O(n²)



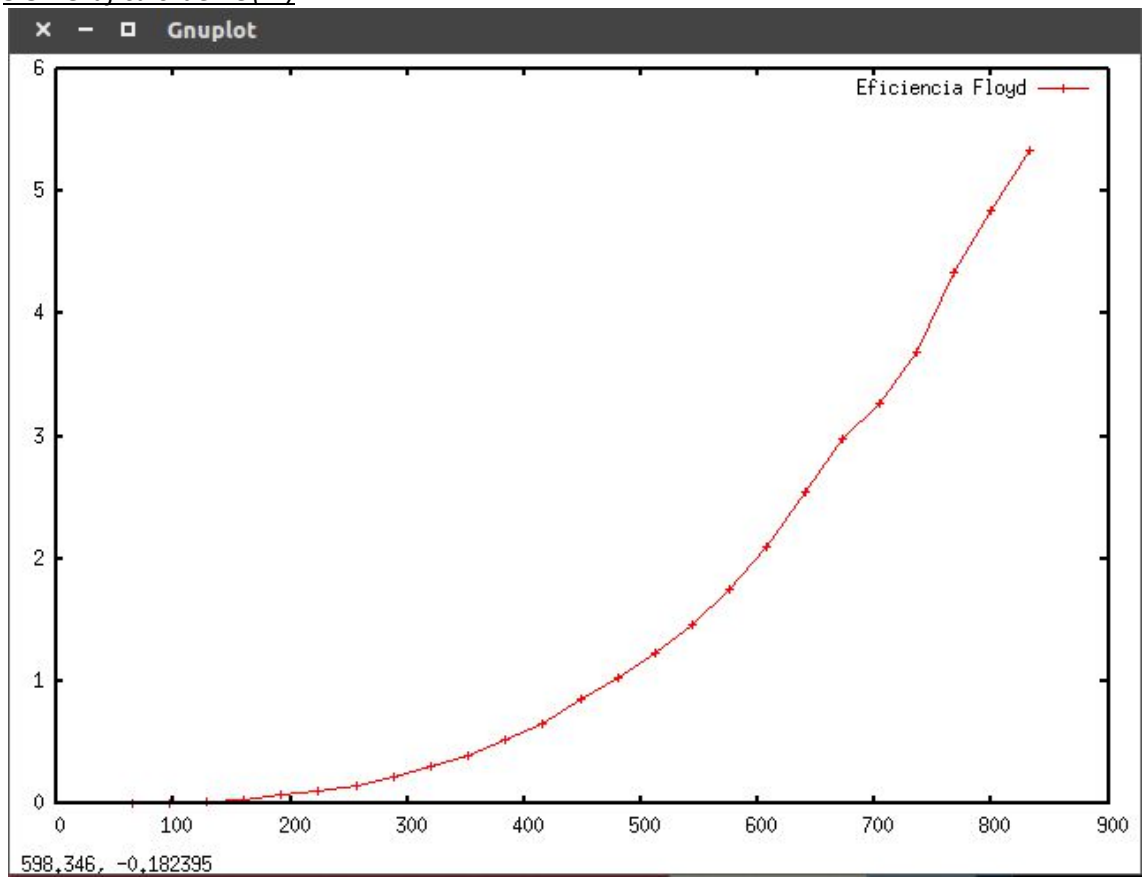
Esta gráfica muestra los algoritmos de ordenación n² teniendo en cuenta el tiempo(seg) y el tamaño de la entrada. Como podemos comprobar el algoritmo de ordenación inserción es el más eficiente seguido por el de selección y por último el burbuja en este caso.

3.2.- Gráfica orden O(nlogn)



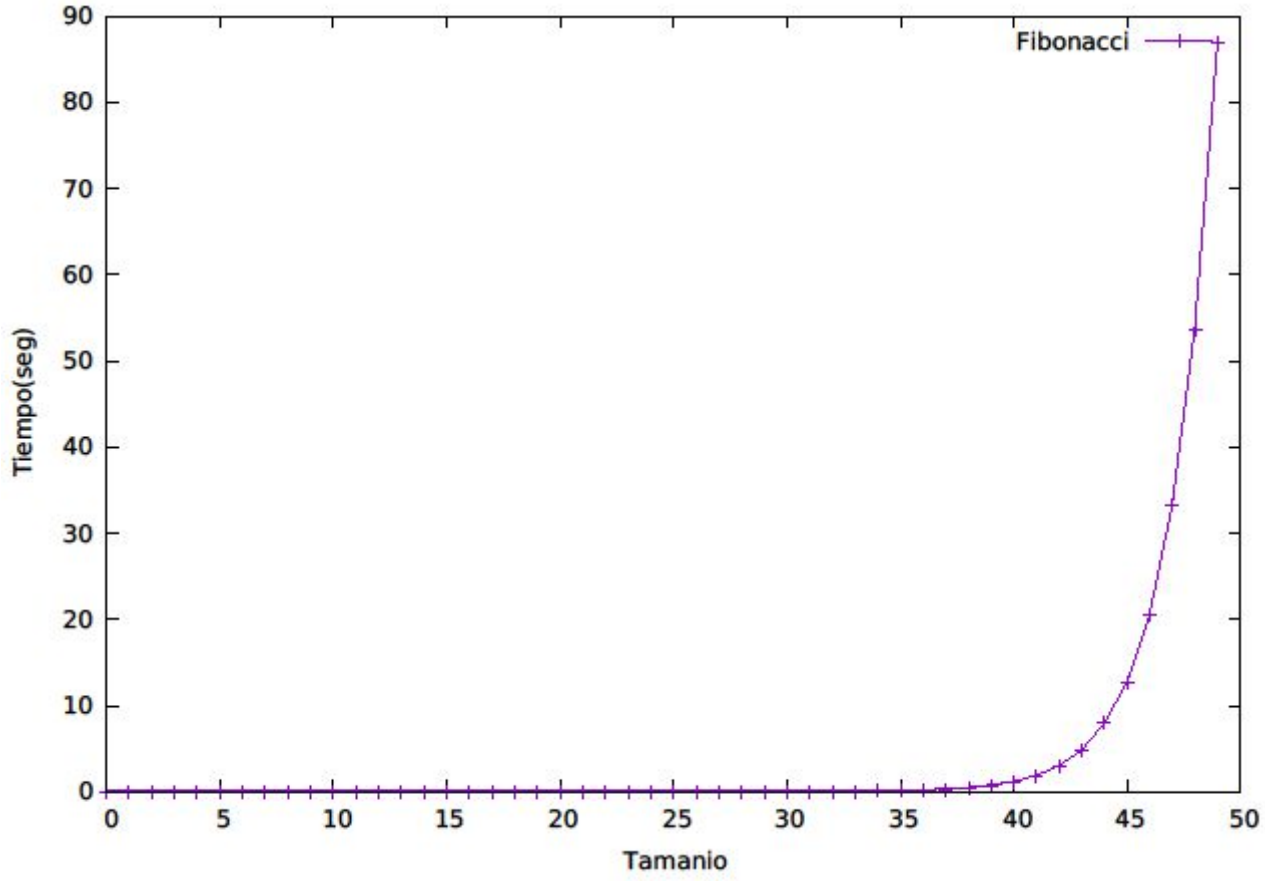
Esta gráfica muestra los algoritmos de ordenación de orden nlogn según el tiempo (seg) y el tamaño de la entrada. El algoritmo quicksort es el más eficiente seguido del mergesort y por último el heapsort. Los picos que podemos observar en la gráfica del mergesort son a causa de intervenciones del sistema operativo durante la ejecución del algoritmo.

3.3.- Gráfica orden O(n³)



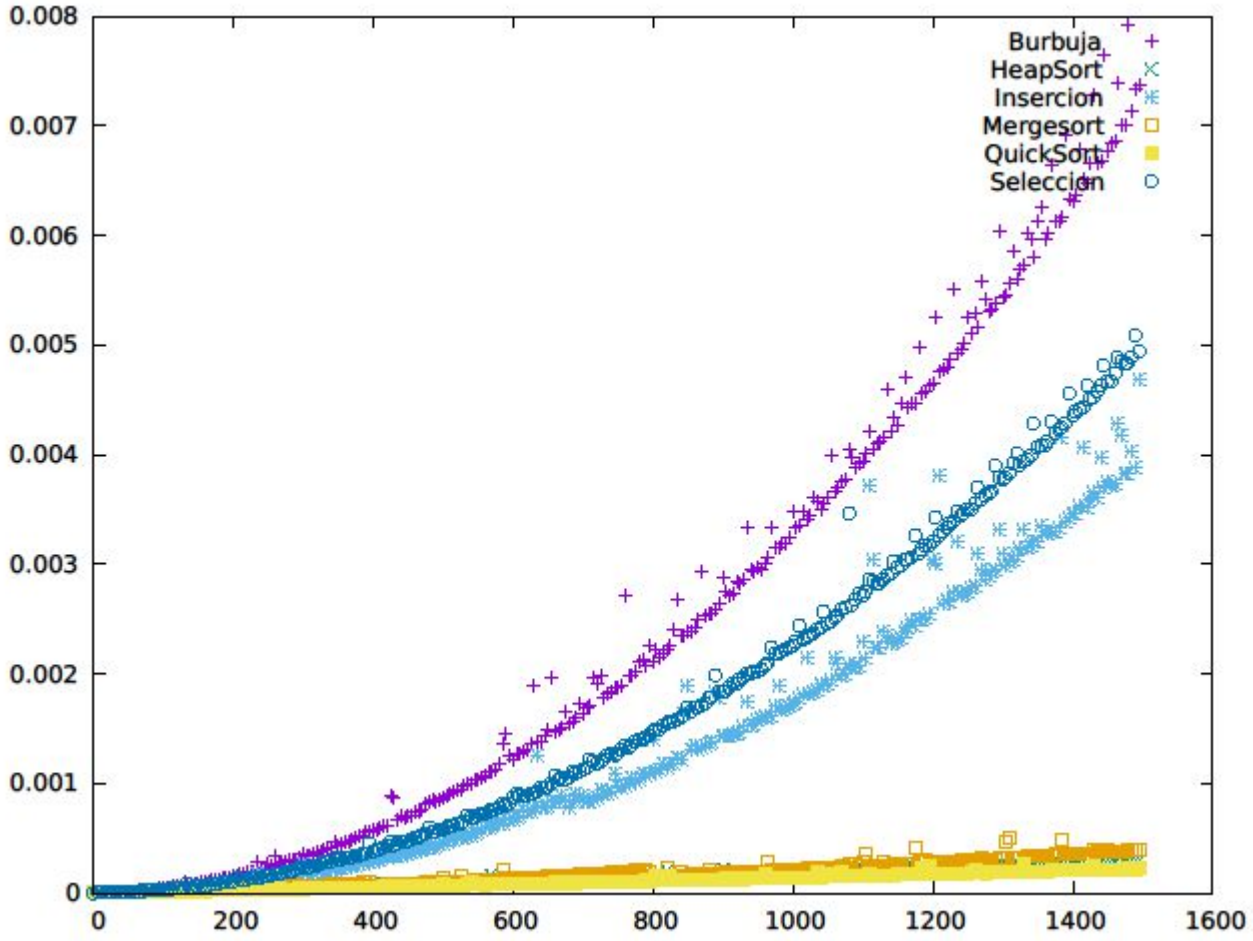
Esta gráfica muestra el algoritmo de ordenación Floyd según el tiempo (seg) y el tamaño de la entrada. Como podemos observar al ser un algoritmo de orden n³ el tiempo de ejecución comparado con el resto de algoritmos es mucho mayor.

3.4.- Gráfica orden (((1+ √5/2)^n))



Esta gráfica muestra el algoritmo de Fibonacci según el tiempo (seg) y el tamaño de la entrada.
Al ser un algoritmo exponencial su eficiencia es mala comparada con el resto de algoritmos.

3.5.- Gráfica de ordenación



Esta gráfica muestra los distintos algoritmos de ordenación a los que les hemos obtenido la eficiencia empírica. Como podemos observar los algoritmos de orden nlogn son los mas eficientes seguidos de los algoritmos de n^2).

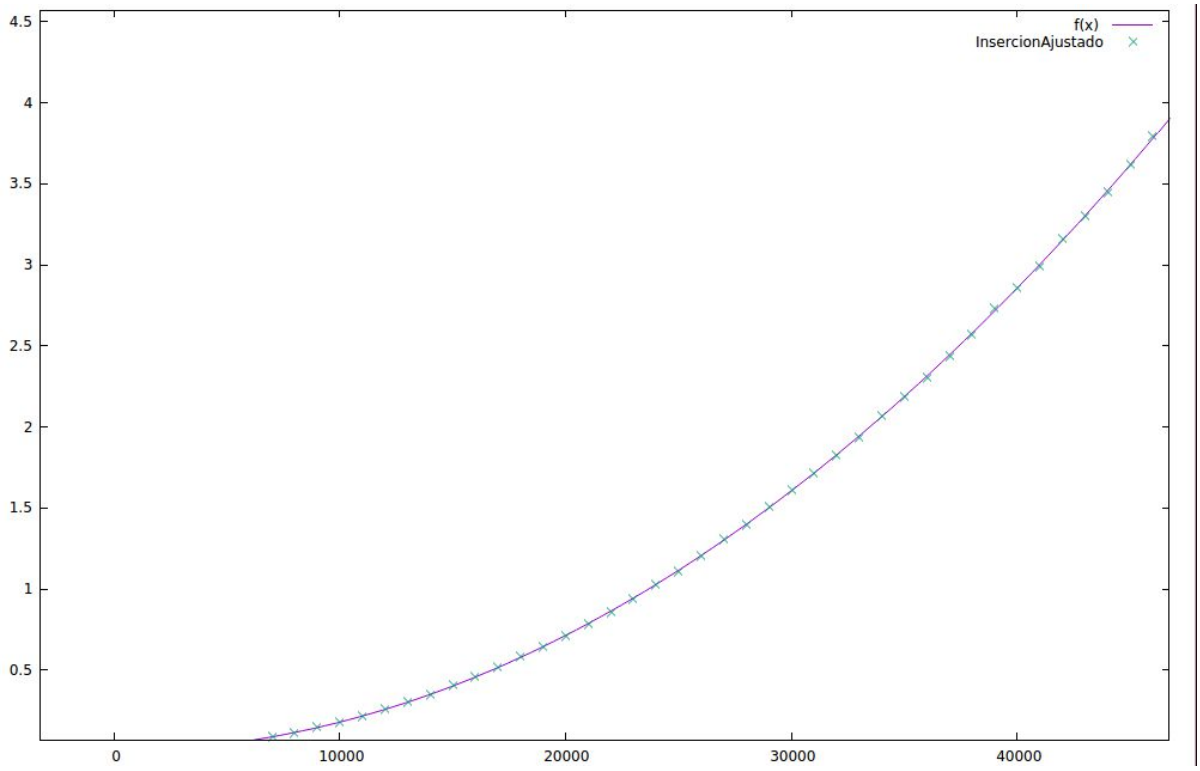
4.- Cálculo eficiencia híbrida

Para calcular la eficiencia híbrida necesitamos conocer el valor de las constantes asociadas a cada término de la expresión obtenida al calcular el tiempo de ejecución de un algoritmo. Para ello hacemos uso del gnuplot, en cada algoritmo vamos a mostrar el error estándar que es el valor que muestra la diferencia entre valores reales y estimados de la regresión.

-Inserción, Burbuja y Selección, los 3 algoritmos han sido ajustados por la función $f(x)=a*x^2+b*x+c$

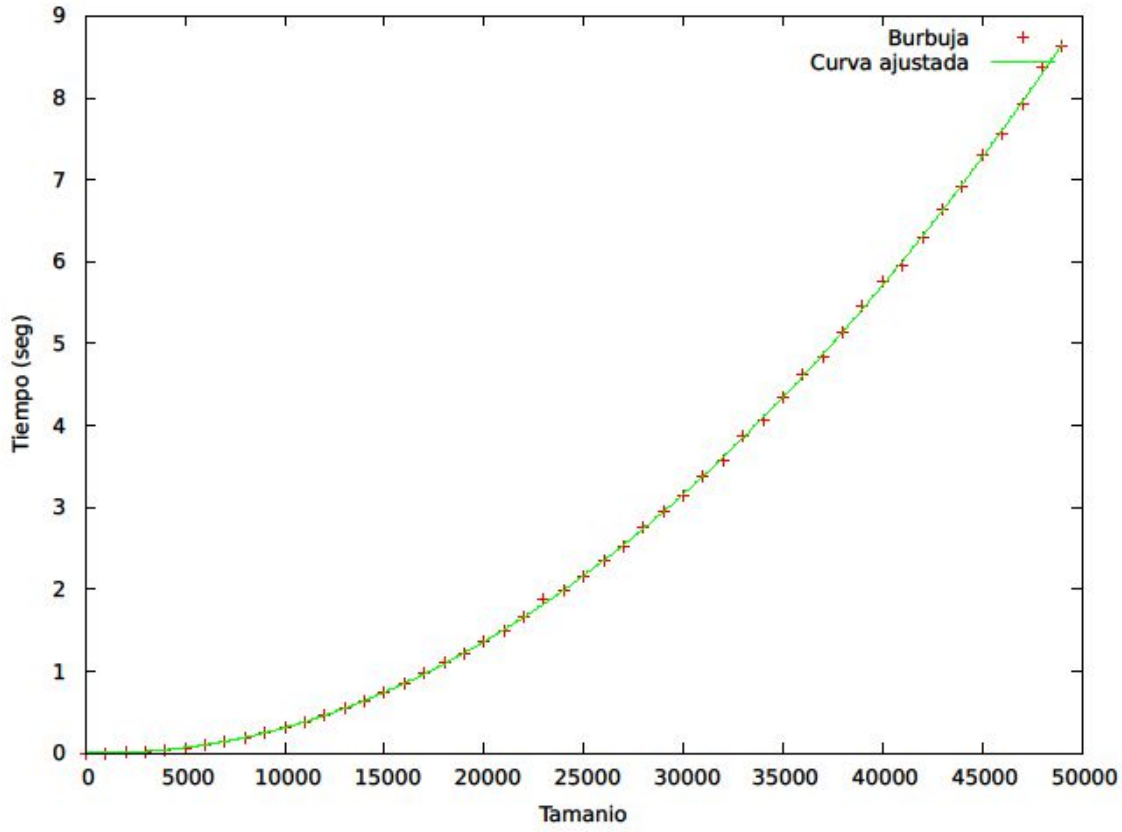
-Inserción

a = 1.79012e-09 +/- 4.433e-12 (0.2476%)
b = -2.75067e-07 +/- 2.286e-07 (83.12%)
c = 0.00259602 +/- 0.002478 (95.45%)



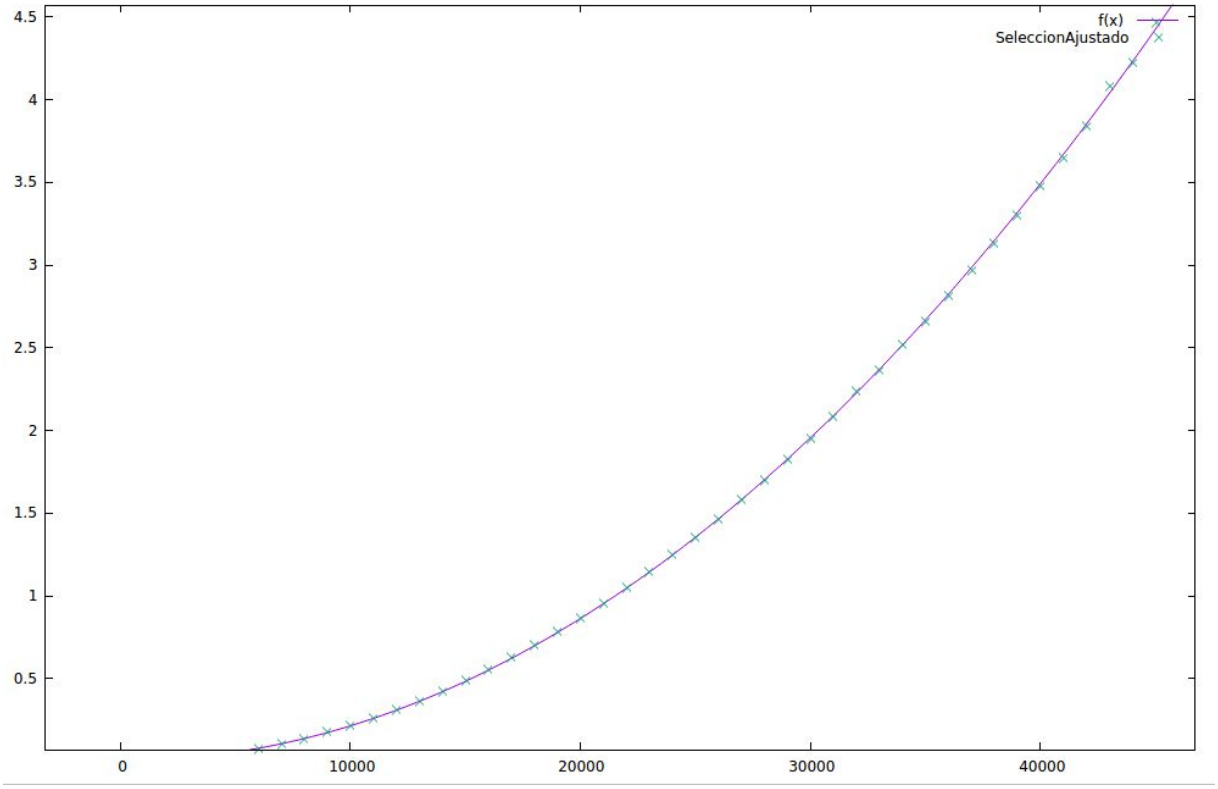
-Burbuja

a = 3.74848e-09 +/- 1.852e-11 (0.4939%)
b = -7.29409e-06 +/- 9.382e-07 (12.86%)
c = 0.00878894 +/- 0.00994 (113.1%)



-Selección

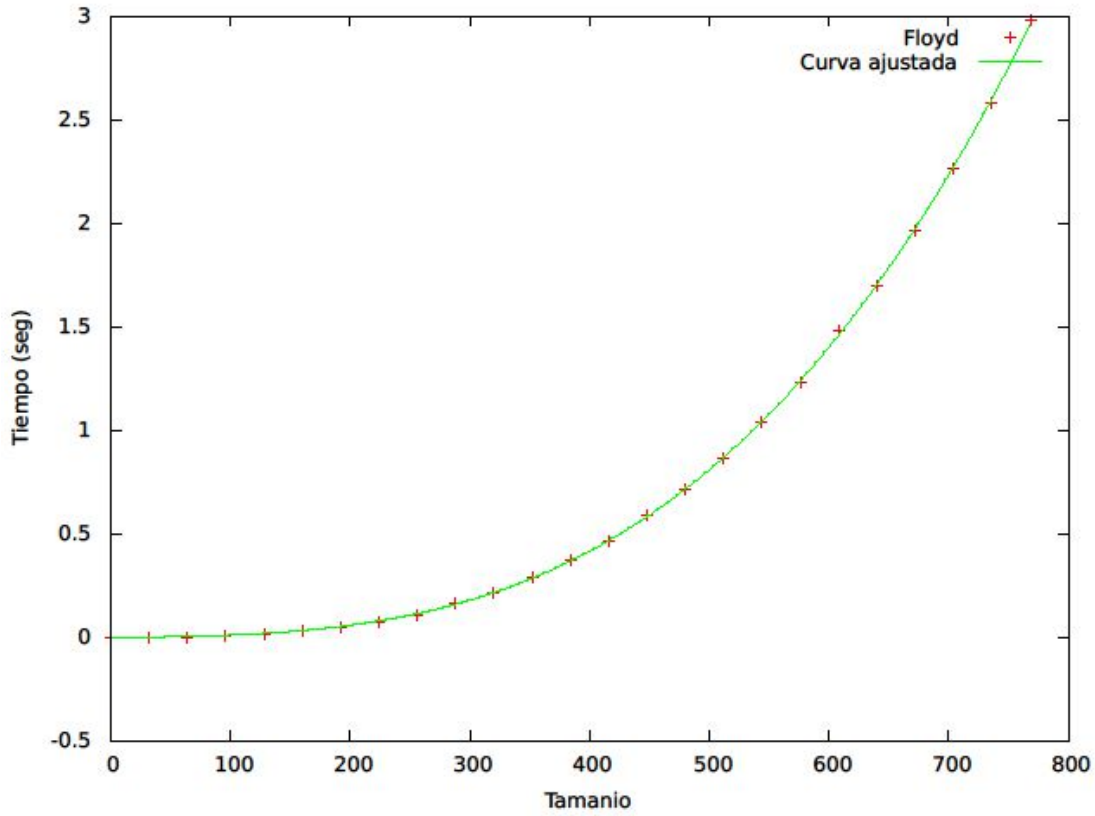
a = 2.21783e-09 +/- 1.115e-11 (0.5027%)
b = -1.70046e-06 +/- 5.404e-07 (31.78%)
c = 0.00922901 +/- 0.005506 (59.66%)



Se ve que el ajuste importante, el de la a tiene un error bastante bueno menor de 1%.

-Floyd ha sido ajustada con la función $f(x)=a*x^3+b*x^2+c*x+d$

a = 4.5651e-14 +/- 3.738e-15 (8.188%)
b = -1.52907e-12 +/- 1.217e-13 (7.961%)
c = 32 +/- 1.204e-12 (3.762e-12%)
d = -1.35692e-11 +/- 3.806e-12 (28.05%)

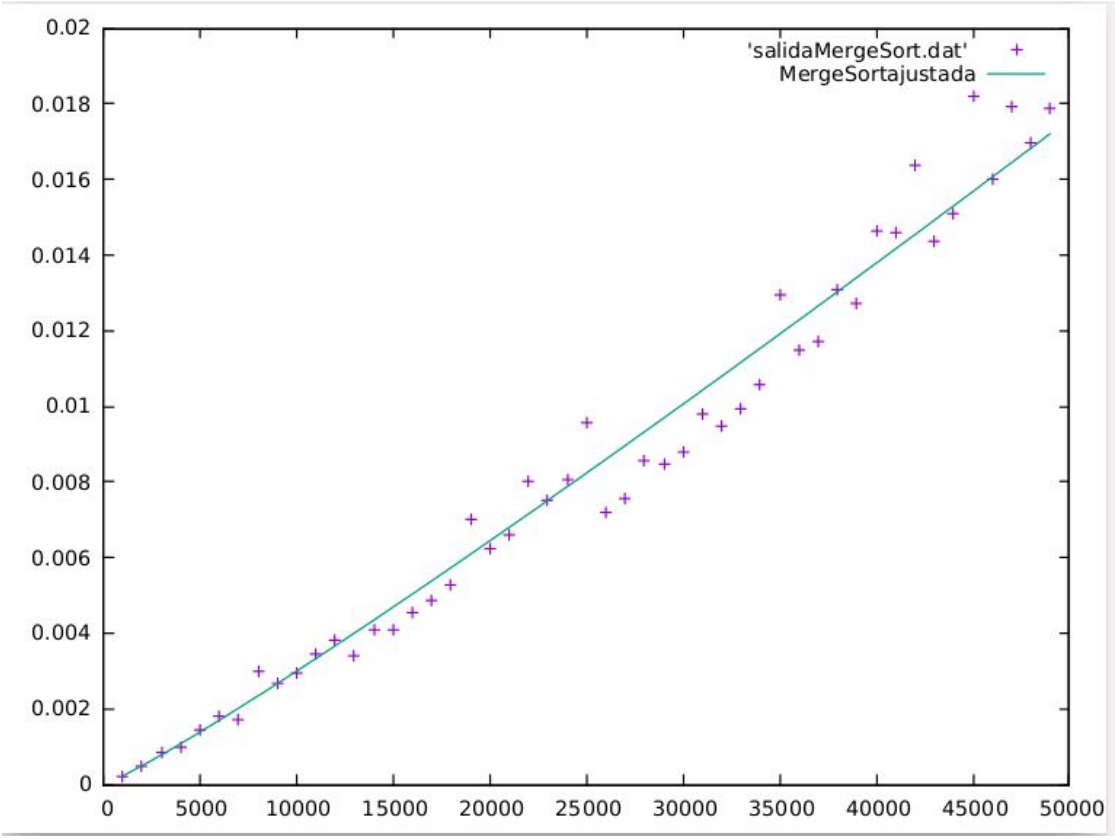


Aunque el error sea algo elevado, la gráfica parece bien ajustada.

-Heapsort, quicksort y mergesort han sido ajustado a la misma función, f(x)=a*x*log(x)

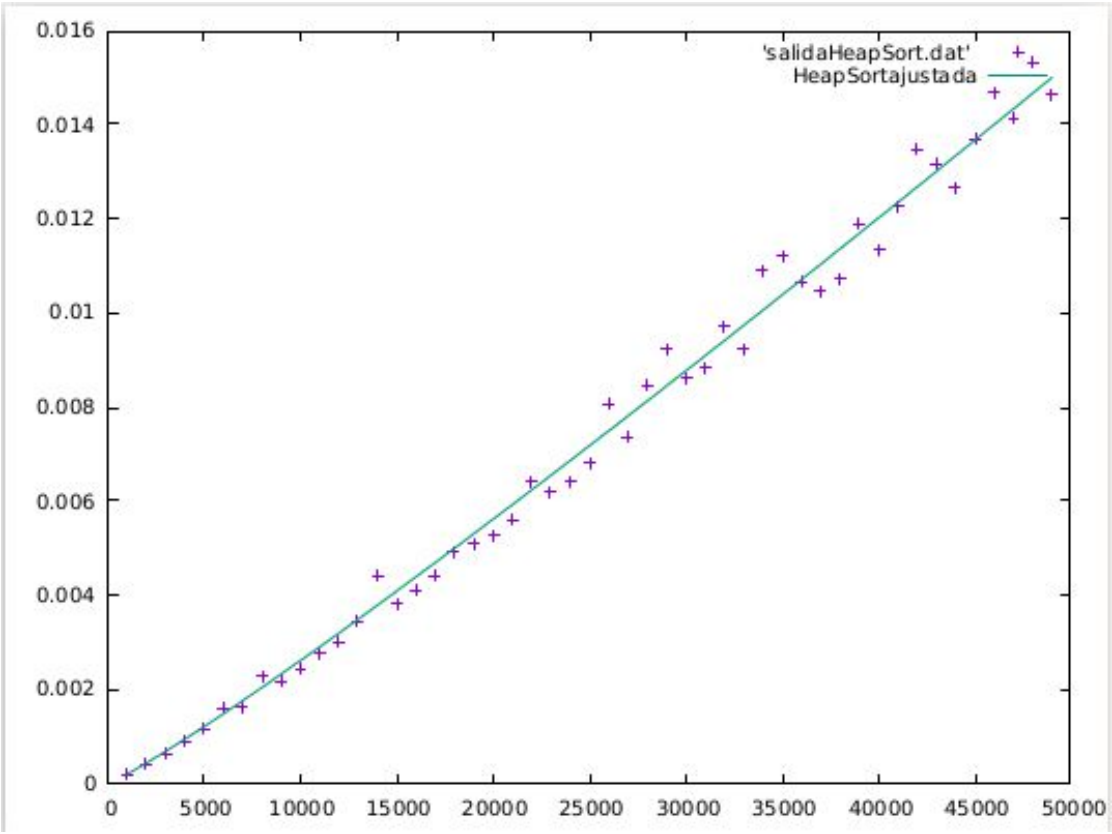
-Mergesort

a = 3.25199e-08 +/- 4.022e-10 (1.237%)



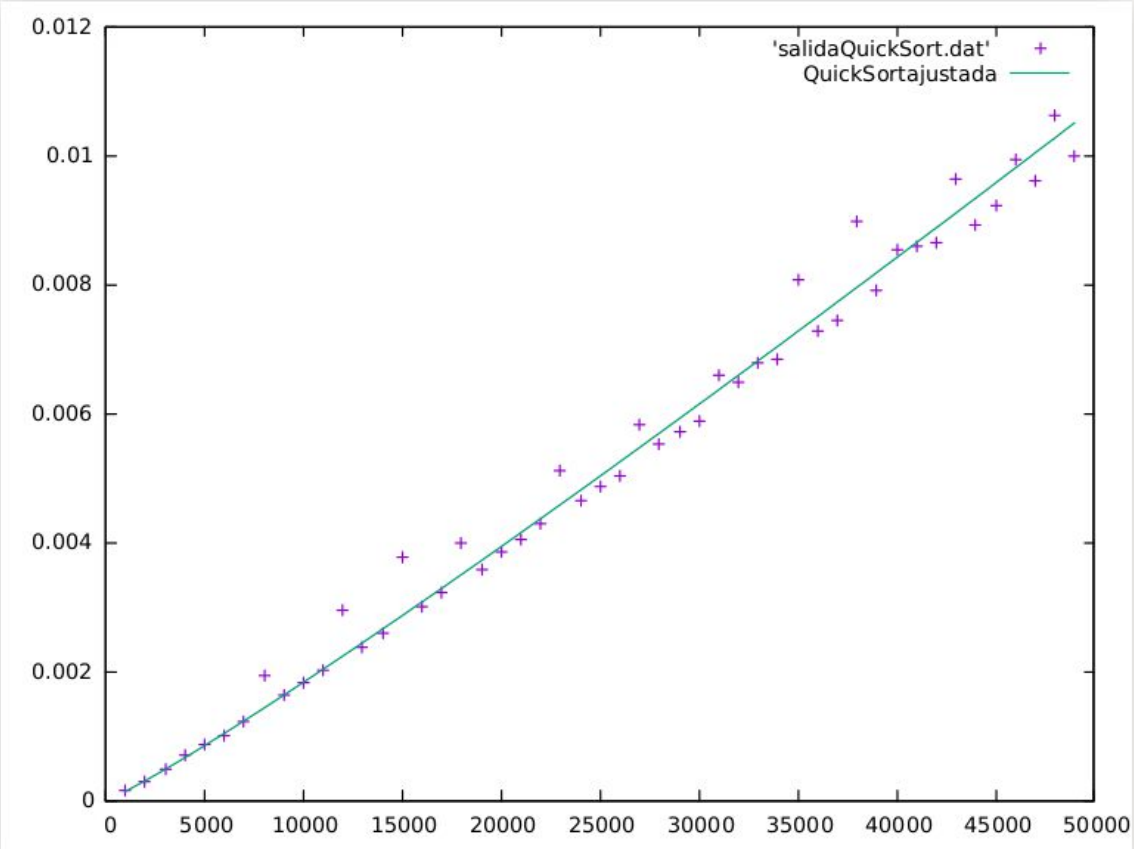
-Heapsort

a = 2.83396e-08 +/- 1.941e-10 (0.6849%)



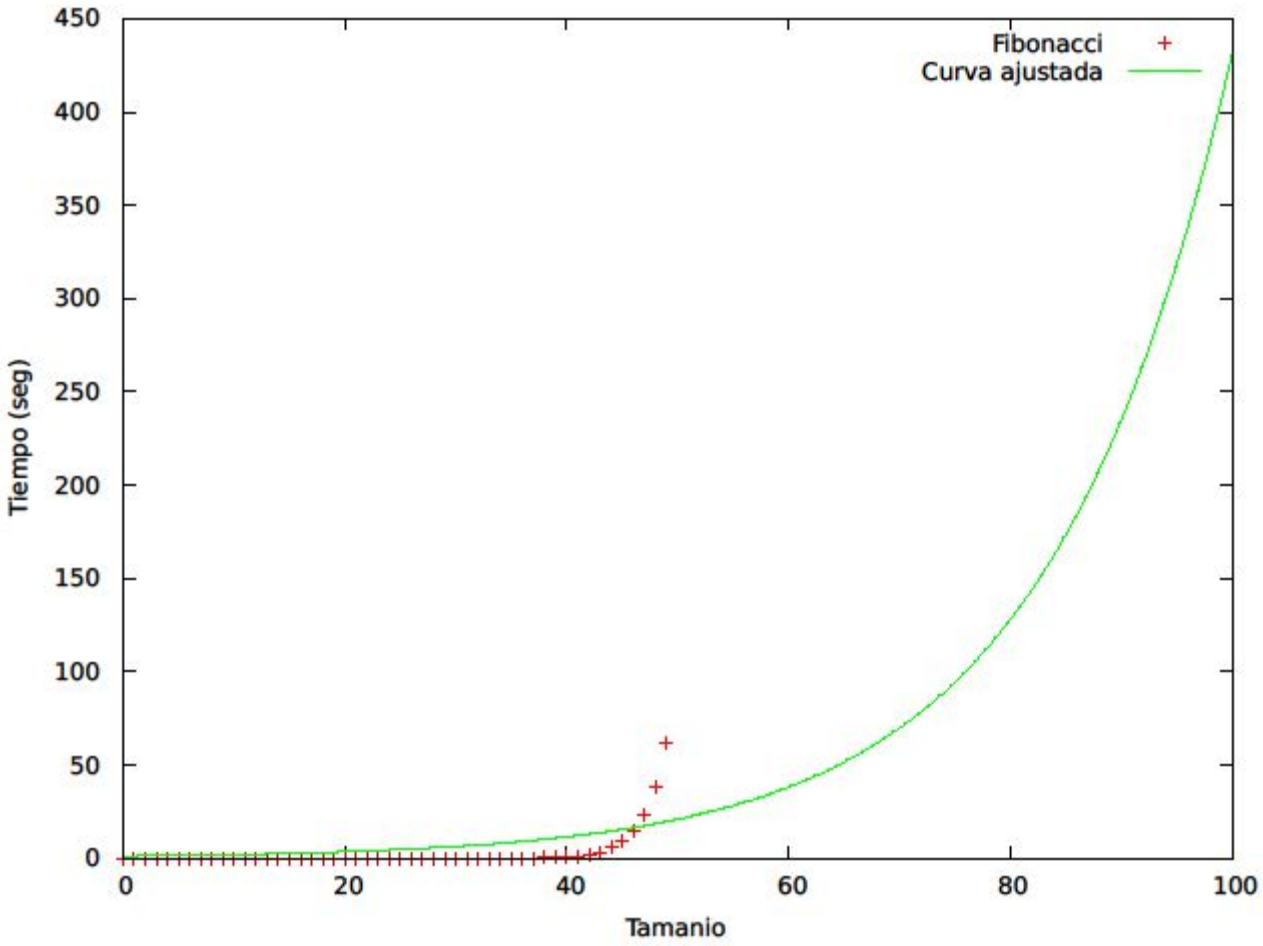
-Quicksort

a = 1.98491e-08 +/- 1.639e-10 (0.8258%)



-Fibonnacci, ha sido ajustada por la función f(x)=a*1,618^x

a = 5.03885e-09 +/- 2.085e-12 (0.04137%)

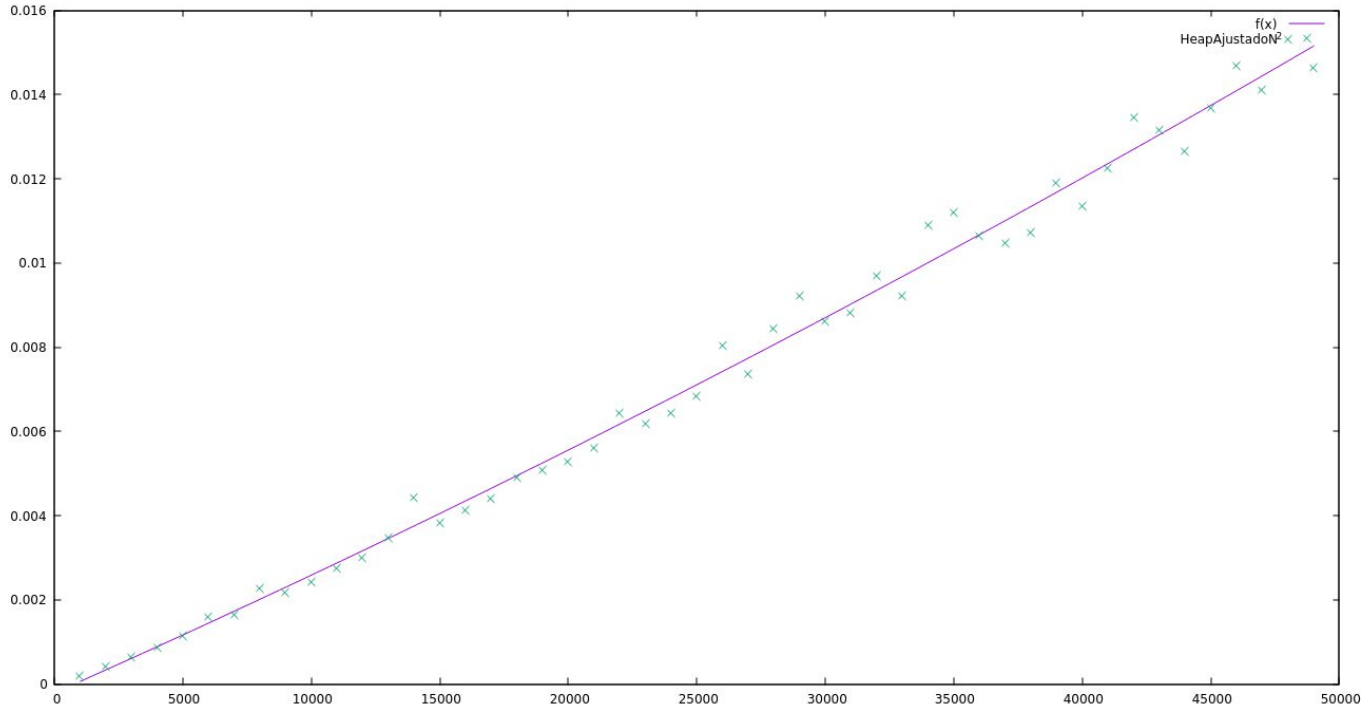


El error aquí es bajo, es un buen ajuste.

-Ahora Hemos probado a ajustar algunos algoritmos con funciones que no son adecuadas y podemos observar como el error crece y el ajuste es peor:

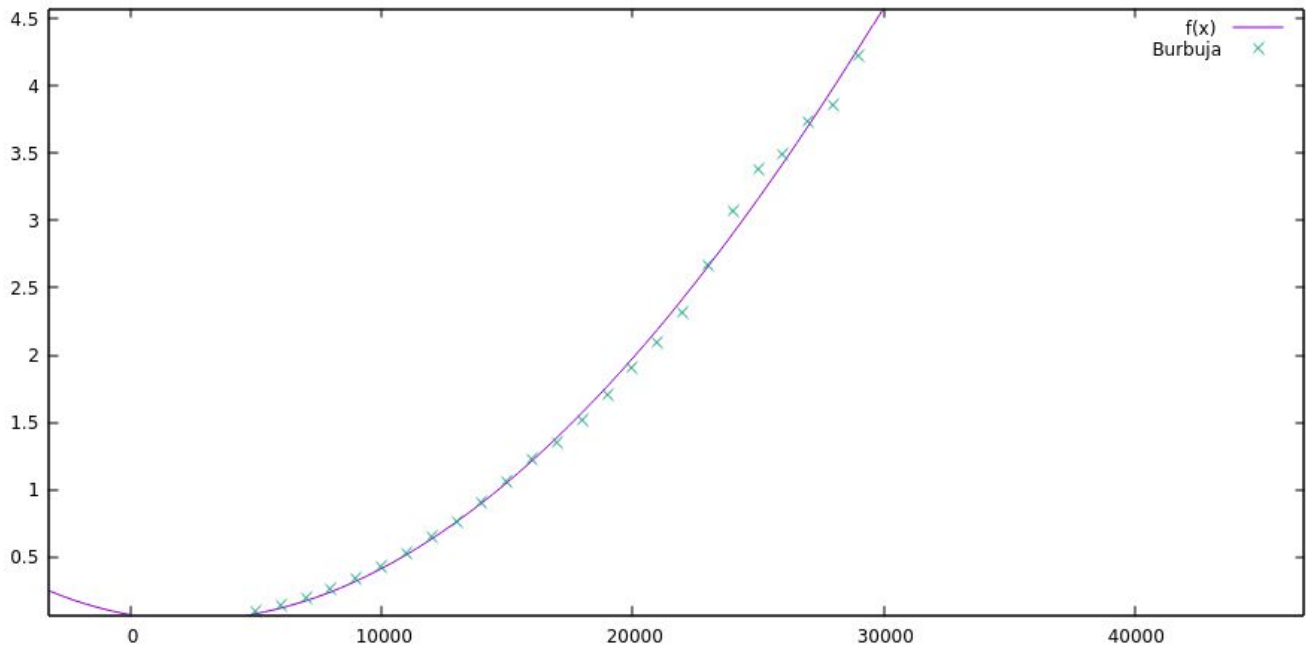
-para HeapSort con f(x)=x*x*a+x*b+c

a = 8.63408e-13 +/- 3.329e-13 (38.56%)
b = 2.713e-07 +/- 1.717e-08 (6.328%)
c = -0.000215156 +/- 0.0001861 (86.49%)



-para Burbuja con f(x)=a*x*x*x+b*x*x+c*x+c

a = 8.63408e-13 +/- 3.329e-13 (38.56%)
b = 2.713e-07 +/- 1.717e-08 (6.328%)
c = -0.000215156 +/- 0.0001861 (86.49%)



Como conclusión podemos extraer que en un buen ajuste se debe usar la función correcta, ya que de otra forma el error cometido será muy grande (>20%)

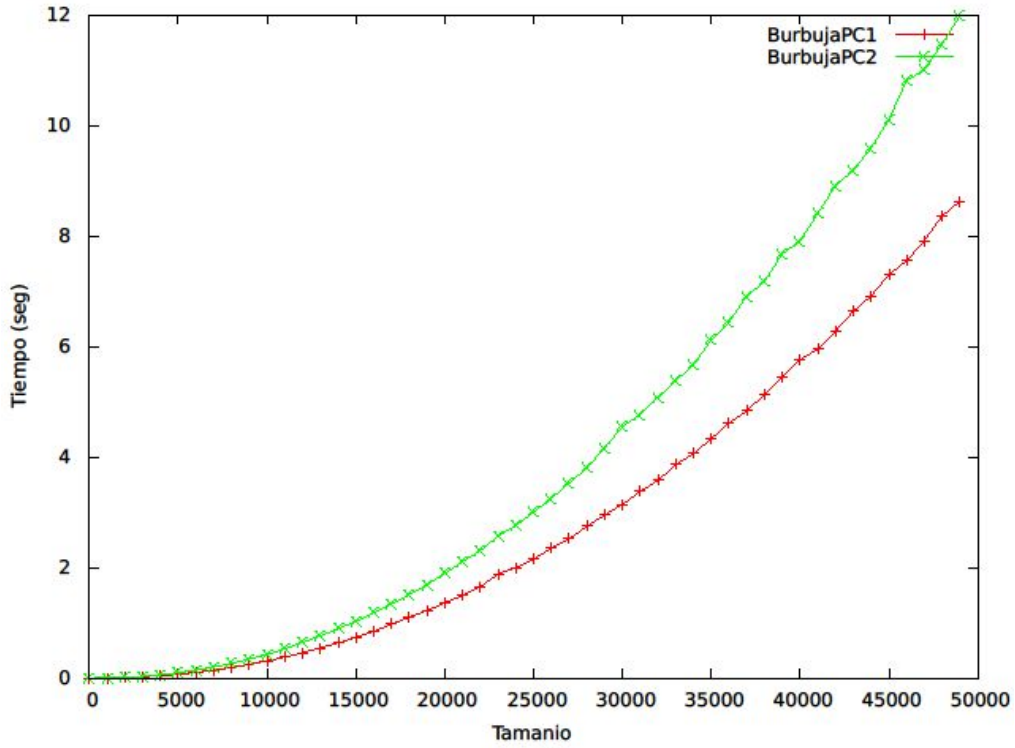
5.- Comparación eficiencia empírica

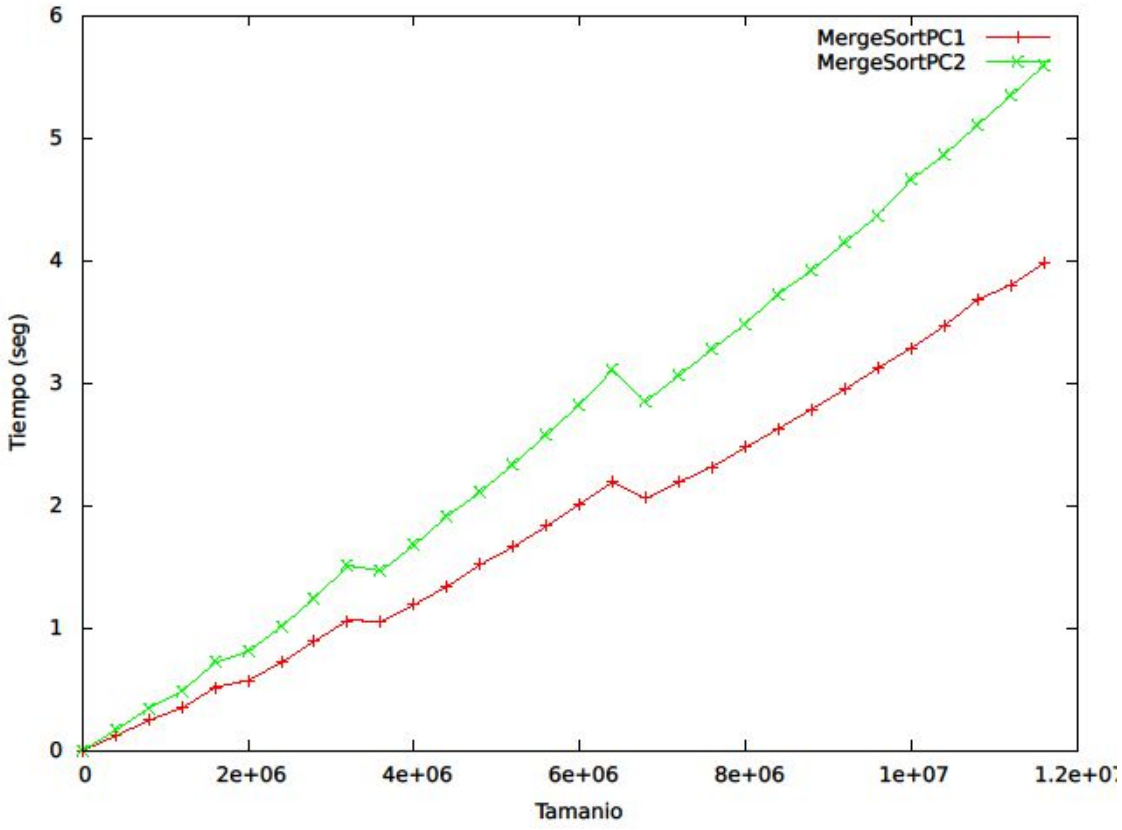
5.1.- Prestaciones PCs utilizados

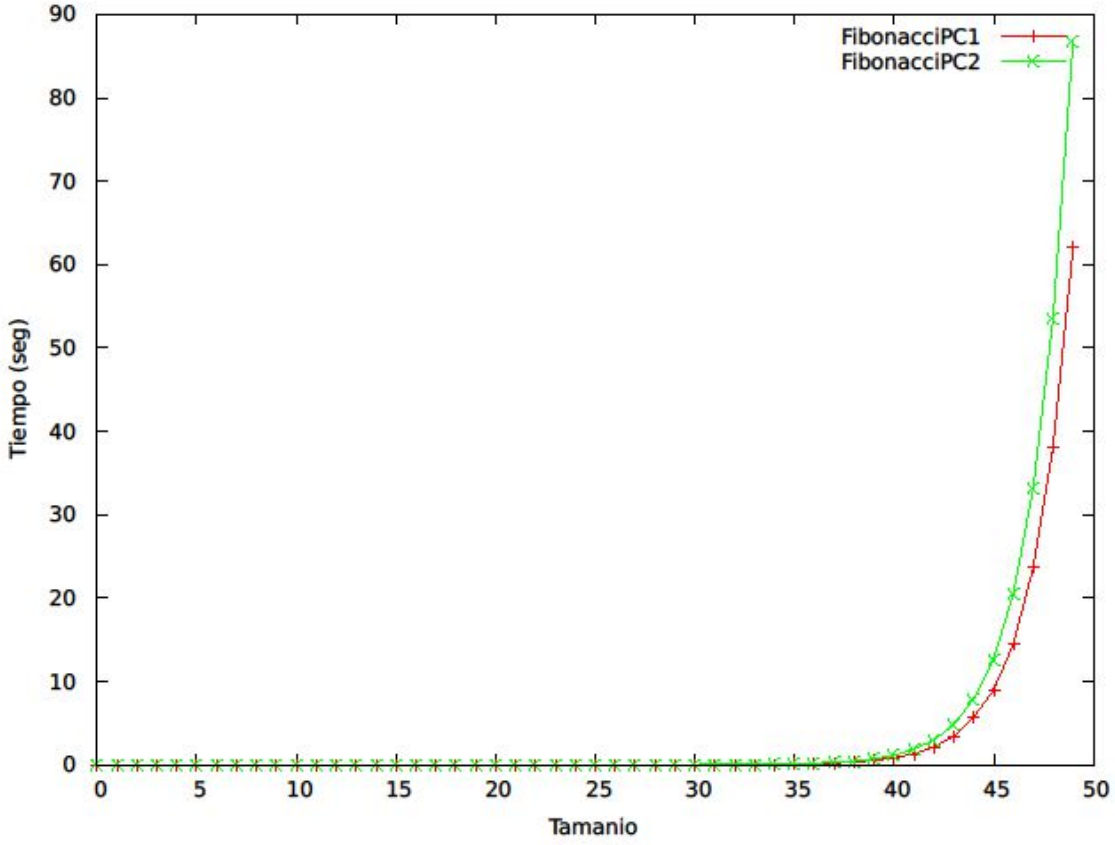
Intel Core i5-5200U 2.2GHz
Intel Core i3-5005U 2.0GHz

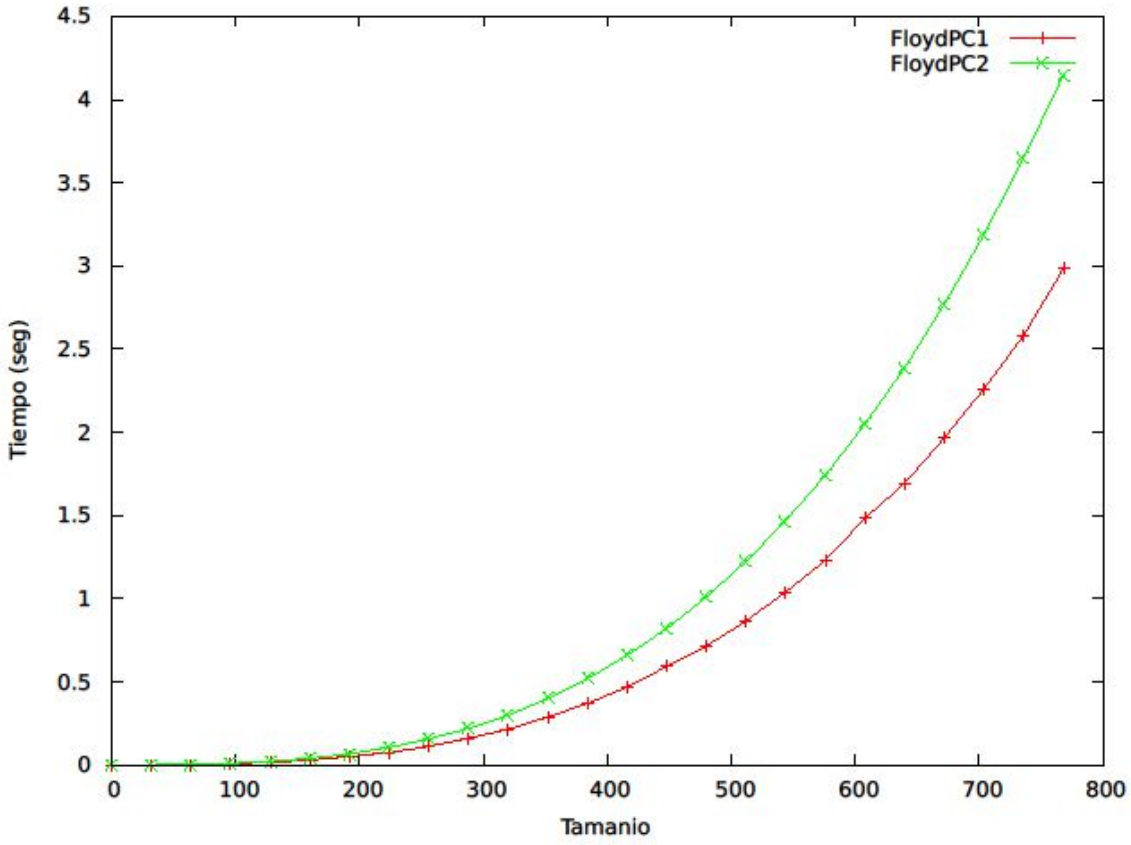
Ambos PC tienen los mismos sistemas operativos (ubuntu 16.04) y no ejecutaban ninguna tarea extra.
Por tanto a igualdad de condiciones las gráficas extraídas son las siguientes:

5.2.- Tiempos obtenidos para los distintos PCs




MergeSortPC1
MergeSortPC2
Tiempo (seg)
Tamanio
0
1
2
3
4
5
6
0
2e+06
4e+06
6e+06
8e+06
1e+07
1.2e+07


FibonacciPC1
FibonacciPC2
Tiempo (seg)
Tamanio
0
10
20
30
40
50
60
70
80
90
0
5
10
15
20
25
30
35
40
45
50



5.3.- Conclusiones de la comparación

Se sabe por tanto que un mejor procesador puede ganar bastante velocidad a la hora de ejecutar algoritmos. Esto es debido a diferentes factores internos que hacen que el i5 supere al i3.