Algoritmica
Practica 1. Eficiencia
2018-2019

UNIVERSIDAD
DE GRANADA

Félix Ramírez García
Raúl Del Pozo Moreno
Cristian Piles Ruiz
Oleksandr Kudryavtsev
Sixto Coca Cruz

Indice

1. Calculo de la eficiencia empírica

    1.1 Burbuja

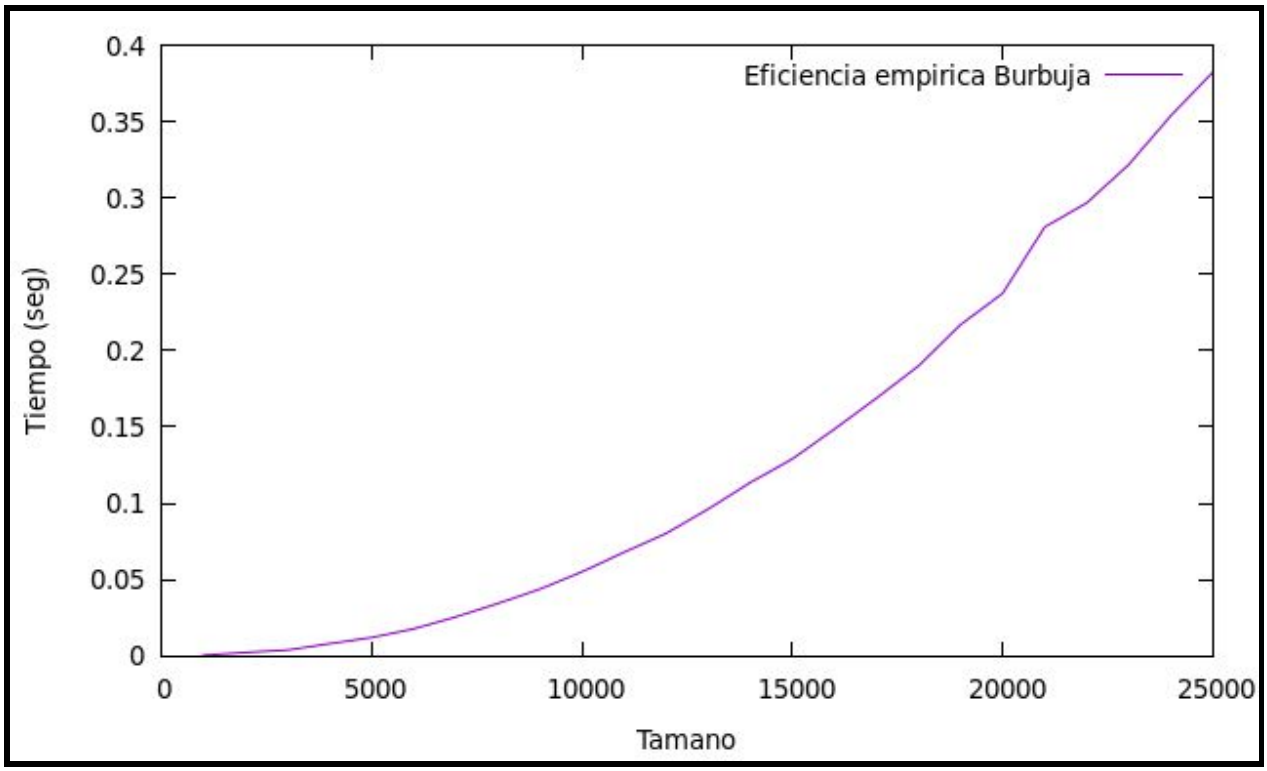
Para calcular la eficiencia empírica del algoritmo burbuja se va medir tiempos antes y después de la llamada al método Burbuja, de esta forma podremos centrarnos en el tiempo del algoritmo y no del programa.

Para obtener un valor lo más real posible, se ha ejecutado la llamada del método 5 veces, en la cual, para cada llamada se reordena el vector inicial. Una vez que se ha ejecutado 5 veces, se obtienen 5 tiempos de ejecución del algoritmo, de entre los cuales nos quedamos el que mayor valor tenga (peor caso).

Este procedimiento se repite para 25 tamaños del vector, desde 1000 hasta 25000, aumentando el tamaño de 1000 en 1000. Así se obtienen los tiempos siguientes:

Tamaño	Tiempo	Tamaño	Tiempo	Tamaño	Tiempo
1000	0.0004804	10000	0.0551738	19000	0.217003
2000	0.0023782	11000	0.0679514	20000	0.23754
3000	0.0039438	12000	0.080295	21000	0.280946
4000	0.081268	13000	0.0962724	22000	0.296733
5000	0.0121242	14000	0.113597	23000	0.321959
6000	0.017839	15000	0.129094	24000	0.35956
7000	0.0256186	16000	0.148562	25000	0.382414
8000	0.0344332	17000	0.168871		
9000	0.0438832	18000	0.19002		

Una vez que se han obtenido los tiempos, se procede a obtener una gráfica, la cual tiene la siguiente forma:



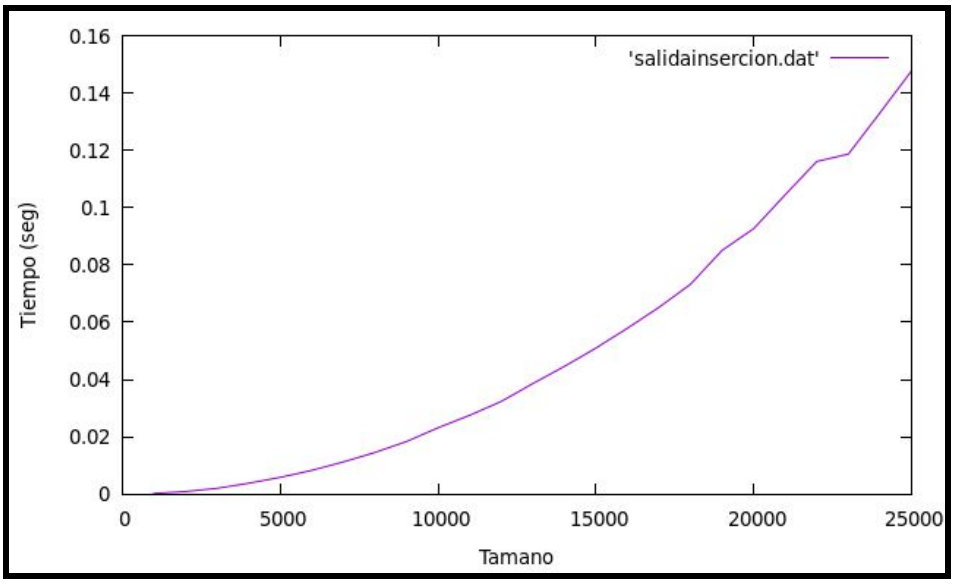
Como se puede ver, la curva obtenida es ascendente, la cual concuerda con una eficiencia $O(n^2)$, la cual coincide con la eficiencia del algoritmo burbuja.

1.2 Insercion

Al igual que en el ejercicio anterior, se ha ejecutado la llamada al método que ejecuta el algoritmo inserción 5 veces, desordenando para cada ejecución el vector. De esta forma se obtienen los tiempos para el peor caso para 25 tamaños diferentes.

Tamaño	Tiempo	Tamaño	Tiempo	Tamaño	Tiempo
1000	0.0002746	10000	0.0231924	19000	0.0851052
2000	0.0009476	11000	0.0275332	20000	0.0926616
3000	0.0020622	12000	0.0323718	21000	0.104502
4000	0.0038262	13000	0.0386124	22000	0.116105
5000	0.0059172	14000	0.0445938	23000	0.118703
6000	0.0083288	15000	0.0509878	24000	0.133071
7000	0.0112868	16000	0.0579388	25000	0.147696
8000	0.0145764	17000	0.0652208		
9000	0.0184044	18000	0.0733034		

El algoritmo inserción también es un algoritmo cuadrático O(n²), por lo que la gráfica obtenida deberá ser parecida a la burbuja.



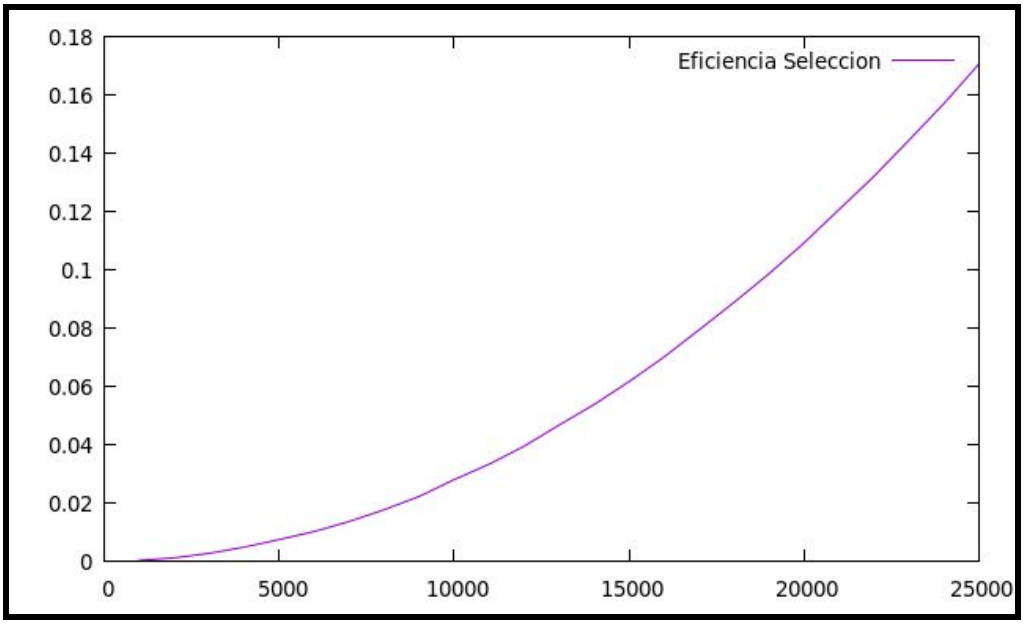
Como se puede observar, también se obtiene una curva ascendente propia de un algoritmo cuadrático.

1.3 Seleccion

Repitiendo 5 veces el método con vectores desordenados, se han obtenido los tiempos siguientes:

Tamaño	Tiempo	Tamaño	Tiempo	Tamaño	Tiempo
1000	0.0003198	10000	0.028039	19000	0.0986498
2000	0.0011808	11000	0.033305	20000	0.109214
3000	0.0027534	12000	0.0394982	21000	0.120674
4000	0.0048866	13000	0.0467548	22000	0.132078
5000	0.0074726	14000	0.0537928	23000	0.144439
6000	0.0102348	15000	0.0616594	24000	0.157045
7000	0.013673	16000	0.0700812	25000	0.17056
8000	0.0177172	17000	0.0793692		
9000	0.022254	18000	0.0888308		

Con una grafica:

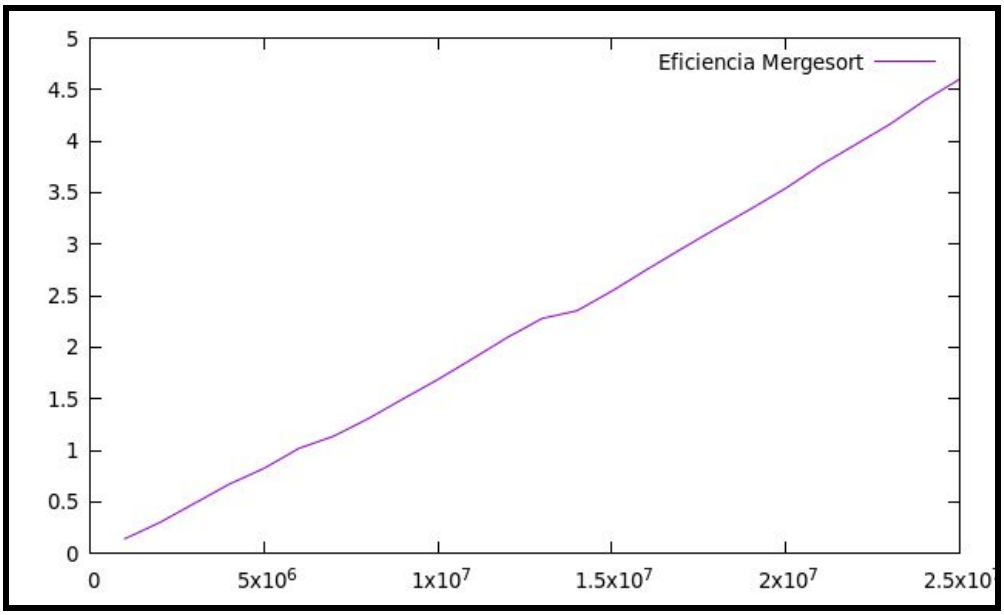


1.4 Mergesort

Este algoritmo ya tiene una eficiencia logarítmica, en este caso, de O(n*log(n)).

Tamaño	Tiempo	Tamaño	Tiempo	Tamaño	Tiempo
1000000	0.140843	10000000	1.68701	19000000	3.34279
2000000	0.298001	11000000	1.88932	20000000	3.54245
3000000	0.484057	12000000	2.09513	21000000	3.76694
4000000	0.670765	13000000	2.27971	22000000	3.96556
5000000	0.824142	14000000	2.35446	23000000	4.16505
6000000	1.0179	15000000	2.54441	24000000	4.39634
7000000	1.13646	16000000	2.75191	25000000	4.60142
8000000	1.30641	17000000	2.95437		
9000000	1.49878	18000000	3.15141		

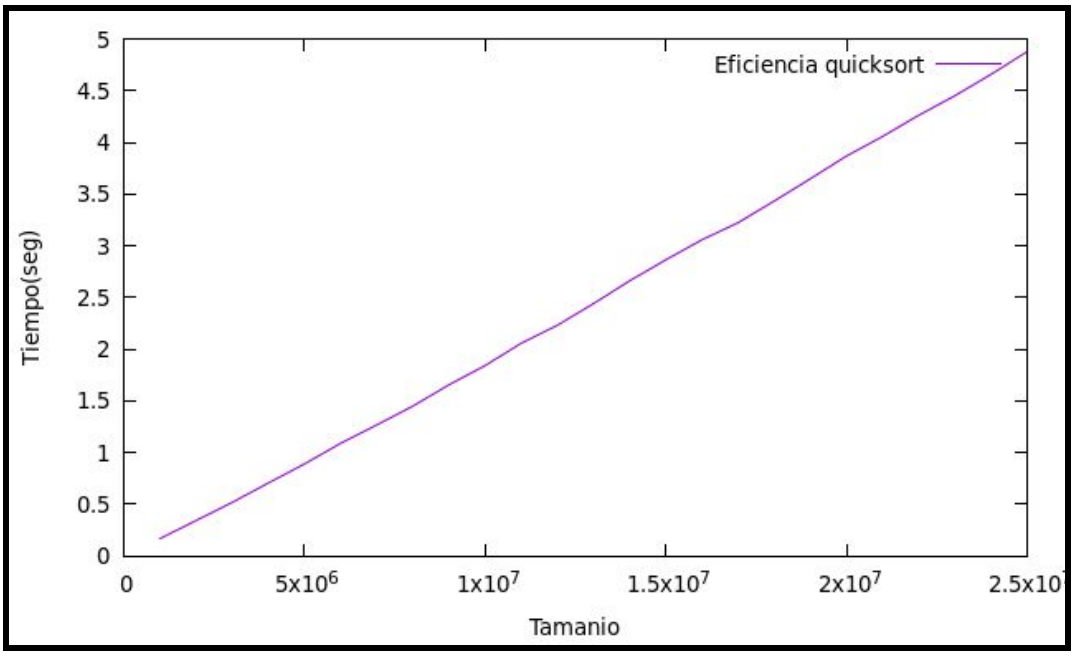
Como se puede ver respecto a los algoritmos cuadráticos, los valores de tamaño se han aumentado considerablemente, ya que los algoritmos logarítmicos son más eficientes que los cuadráticos y podrán gestionar mayores tamaños de datos.



Aunque se espera una curva característica logarítmica, lo que se obtiene es otro tipo de curva, ya que está influenciado por el “n” previo al logaritmo.

1.5 Quicksort

El algoritmo quicksort se ha ejecutado desde tamaño 1000000 hasta tamaño 25000000 con incrementos de 1000000. La gráfica de la eficiencia empírica es la siguiente:

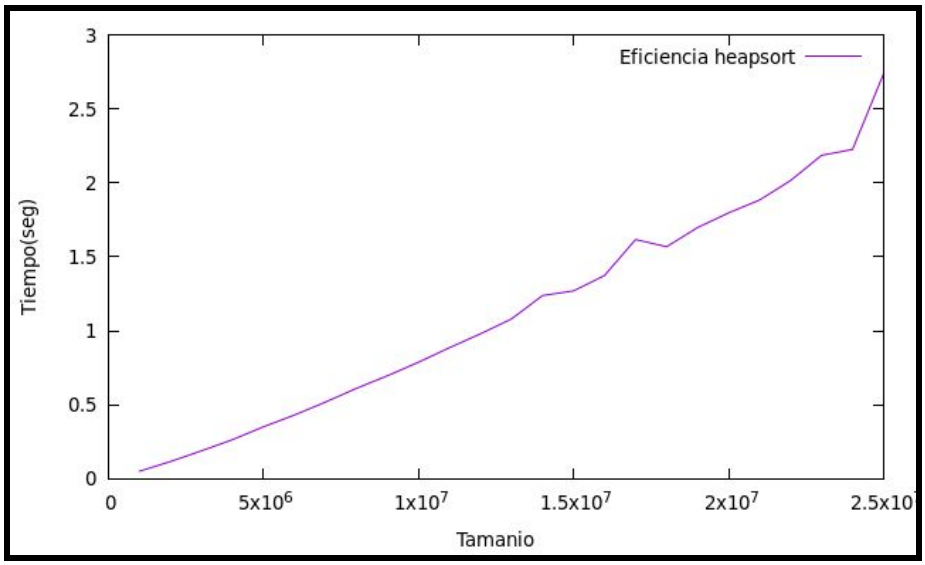


Y la tabla con los resultados obtenidos de la ejecución del algoritmo es:

Tamaño	Tiempo	Tamaño	Tiempo	Tamaño	Tiempo
1000000	0.169305	10000000	1.8423	19000000	3.64882
2000000	0.344614	11000000	2.05908	20000000	3.86758
3000000	0.51929	12000000	2.23227	21000000	4.05776
4000000	0.707517	13000000	2.44141	22000000	4.26158
5000000	0.89087	14000000	2.66107	23000000	4.45113
6000000	1.09104	15000000	2.86472	24000000	4.65702
7000000	1.26858	16000000	3.0586	25000000	4.8773
8000000	1.45083	17000000	3.22337		
9000000	1.6571	18000000	3.43288		

1.6 Heapsort

El algoritmo Heapsort se ha ejecutado desde tamaño 1000000 hasta tamaño 25000000 con incrementos de 1000000. La gráfica de la eficiencia empírica es la siguiente:

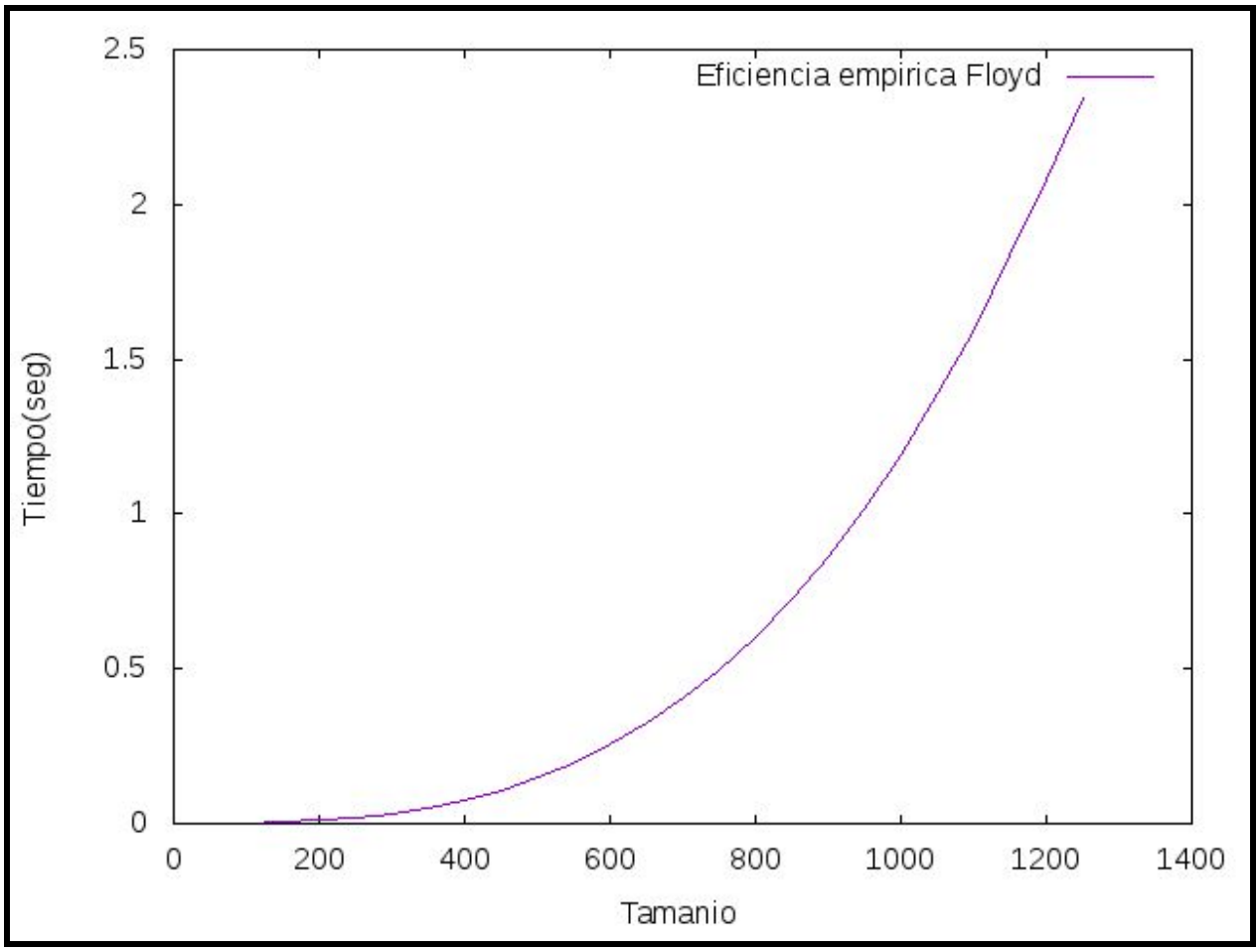


Y la tabla con los resultados obtenidos de la ejecución del algoritmo es:

Tamaño	Tiempo	Tamaño	Tiempo	Tamaño	Tiempo
1000000	0.05136	10000000	0.788029	19000000	1.69776
2000000	0.116546	11000000	0.885974	20000000	1.79699
3000000	0.188706	12000000	0.97959	21000000	1.88413
4000000	0.264342	13000000	1.08039	22000000	2.0155
5000000	0.351473	14000000	1.23794	23000000	2.18586
6000000	0.430926	15000000	1.26974	24000000	2.22568
7000000	0.518344	16000000	1.37381	25000000	2.73769
8000000	0.611572	17000000	1.61648		
9000000	0.696531	18000000	1.56826		

1.7 Floyd

Hemos ejecutado el algoritmo Floyd desde tamaño 0 hasta tamaño 1250 con un incremento de 50.La gráfica de la eficiencia empírica es la siguiente:

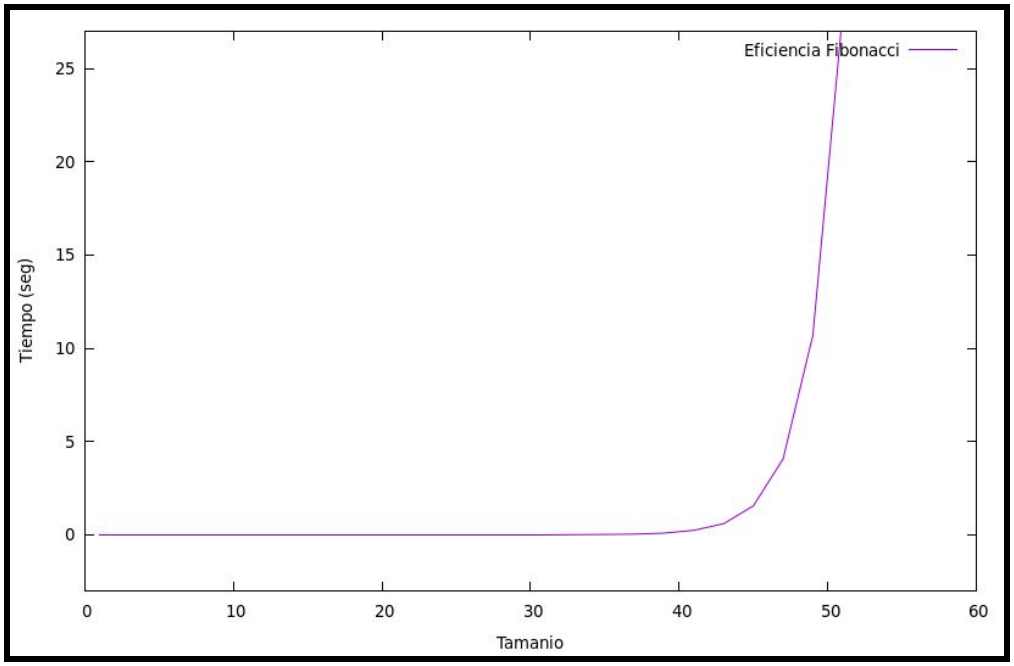


Su tabla correspondiente es:

Tamaño	Tiempo	Tamaño	Tiempo	Tamaño	Tiempo
50	0.000199	500	0.148444	950	1.0153
100	0.0012598	550	0.196276	1000	1191
150	0.0052864	600	0.253895	1050	1.38909
200	0.0100262	650	0.323513	1100	1.60048
250	0.0197854	700	0.404964	1150	1.83547
300	0.0330412	750	0.497985	1200	2.0804
350	0.0519988	800	0.601423	1250	2.34557
400	0.0768832	850	0.725005		
450	0.108187	900	0.864828		

1.8 Fibonacci

El algoritmo Fibonacci se ha ejecutado desde 1 hasta 51 con un incremento de 2 en 2, debido que para un número de dígitos mayor, tarda mucho tiempo ya que es de orden O(([1+ sqrt(5)] / 2)^n), que sería exponencial de n, algo que se va a comprobar con los tiempos obtenidos. La gráfica de la eficiencia empírica es:



Como se ve, para tamaños hasta 40 el tiempo es prácticamente 0, pero a partir de ahí hasta 50, el tiempo sube hasta los 25 segundos. Su tabla correspondiente es:

Tamaño	Tiempo
1	2,00E-07
3	2,00E-07
5	2,00E-07
7	2,00E-07
9	2,00E-07
11	4,00E-07
13	6,00E-07
15	1,20E-06
17	2,40E-06
19	6,00E-06
21	1,96E-05
23	3,92E-05
25	0,0001176

Tamaño	Tiempo
27	0,0003548
29	0,0008368
31	0,0022962
33	0,005214
35	0,0132942
37	0,0345578
39	0,0876682
41	0,235015
43	0,595643
45	1,55399
47	4,06551
49	10,6972
51	27,9132

1.9 Tablas orden O(n²)

Las tablas de orden n² son:

Tamaño	Burbuja	Insercion	Selecion
1000	0.0004804	0.0002746	0.0003198
2000	0.0023782	0.0009476	0.0011808
3000	0.0039438	0.0020622	0.0027534
4000	0.0081268	0.0038262	0.0048866
5000	0.0121242	0.0059172	0.0074726
6000	0.017839	0.0083288	0.0102348
7000	0.0256186	0.0112868	0.013673
8000	0.0344332	0.0145764	0.0177172
9000	0.0438832	0.0184044	0.022254
10000	0.0551738	0.0231924	0.028039
11000	0.0679514	0.0275332	0.033305
12000	0.080295	0.0323718	0.0394982
13000	0.0962724	0.0386124	0.0467548
14000	0.113597	0.0445938	0.0537928
15000	0.129094	0.0509878	0.0616594
16000	0.148562	0.0579388	0.0700812
17000	0.168871	0.0652208	0.0793692
18000	0.19002	0.0733034	0.0888308
19000	0.217003	0.0851052	0.0986498

20000	0.237514	0.0926616	0.109214
21000	0.280946	0.104502	0.120674
22000	0.296733	0.116105	0.132078
23000	0.321959	0.118703	0.144439
24000	0.353956	0.133071	0.157045
25000	0.382414	0.147696	0.17056
Tamaño	Burbuja	Insercion	Seleccion

Como se puede ver, los tiempos del algoritmo burbuja son mayores que los demás, al igual que el algoritmo seleccionado es mayor que el de inserción, siendo este último el más eficiente.

1.10 Tablas orden O(n log n)

Las tablas de orden nlog(n) son:

Tamaño	Mergesort	Heapsort	Quicksort
1000000	0.140843	0.05136	0.169305
2000000	0.298001	0.116546	0.344614
3000000	0.484057	0.188706	0.51929
4000000	0.670765	0.264342	0.707517
5000000	0.824142	0.351473	0.89087
6000000	1.0179	0.430926	1.09104
7000000	1.13646	0.518344	1.26858
8000000	1.30641	0.611572	1.45083
9000000	1.49878	0.696531	1.6571

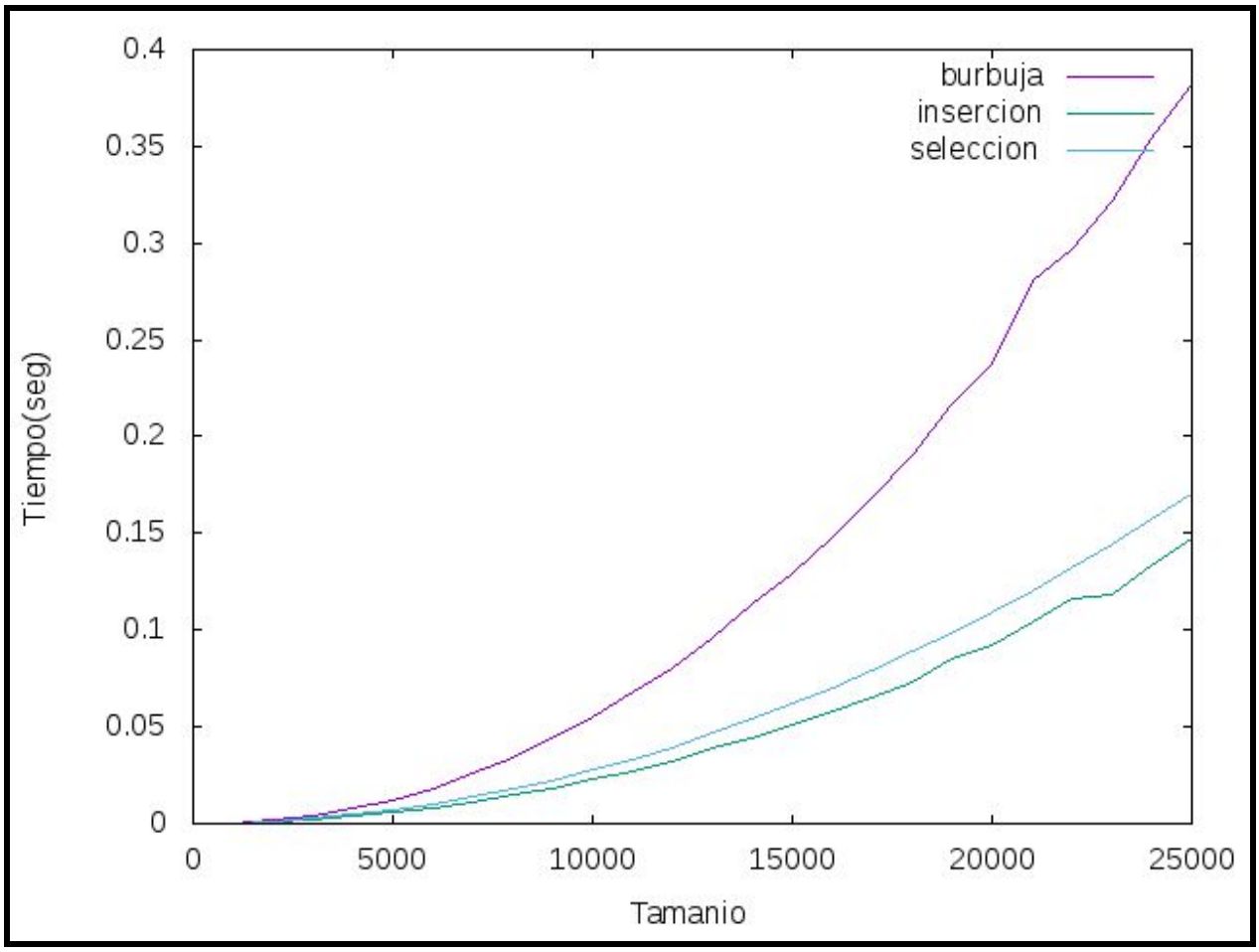
10000000	1.68701	0.788029	1.8423
11000000	1.88932	0.885974	2.05908
12000000	2.09513	0.97959	2.23227
13000000	2.27971	1.08039	2.44141
14000000	2.35446	1.23794	2.66107
15000000	2.54441	1.26974	2.86472
16000000	2.75191	1.37381	3.0586
17000000	2.95437	1.61648	3.22337
18000000	3.15141	1.56826	3.43288
19000000	3.34279	1.69776	3.64882
20000000	3.54245	1.79699	3.86758
21000000	3.76694	1.88413	4.05776
22000000	3.96556	2.0155	4.26158
23000000	4.16505	2.18586	4.45113
24000000	4.39634	2.22568	4.65702
25000000	4.60142	2.73769	4.8773
Tamaño	Mergesort	Heapsort	Quicksort

Como se puede ver, los tiempos para Quicksort son mayores, intermedios para Mergesort y menores para Heapsort.

2. Comparativa de algoritmos

2.1 Comparativa O(n²)

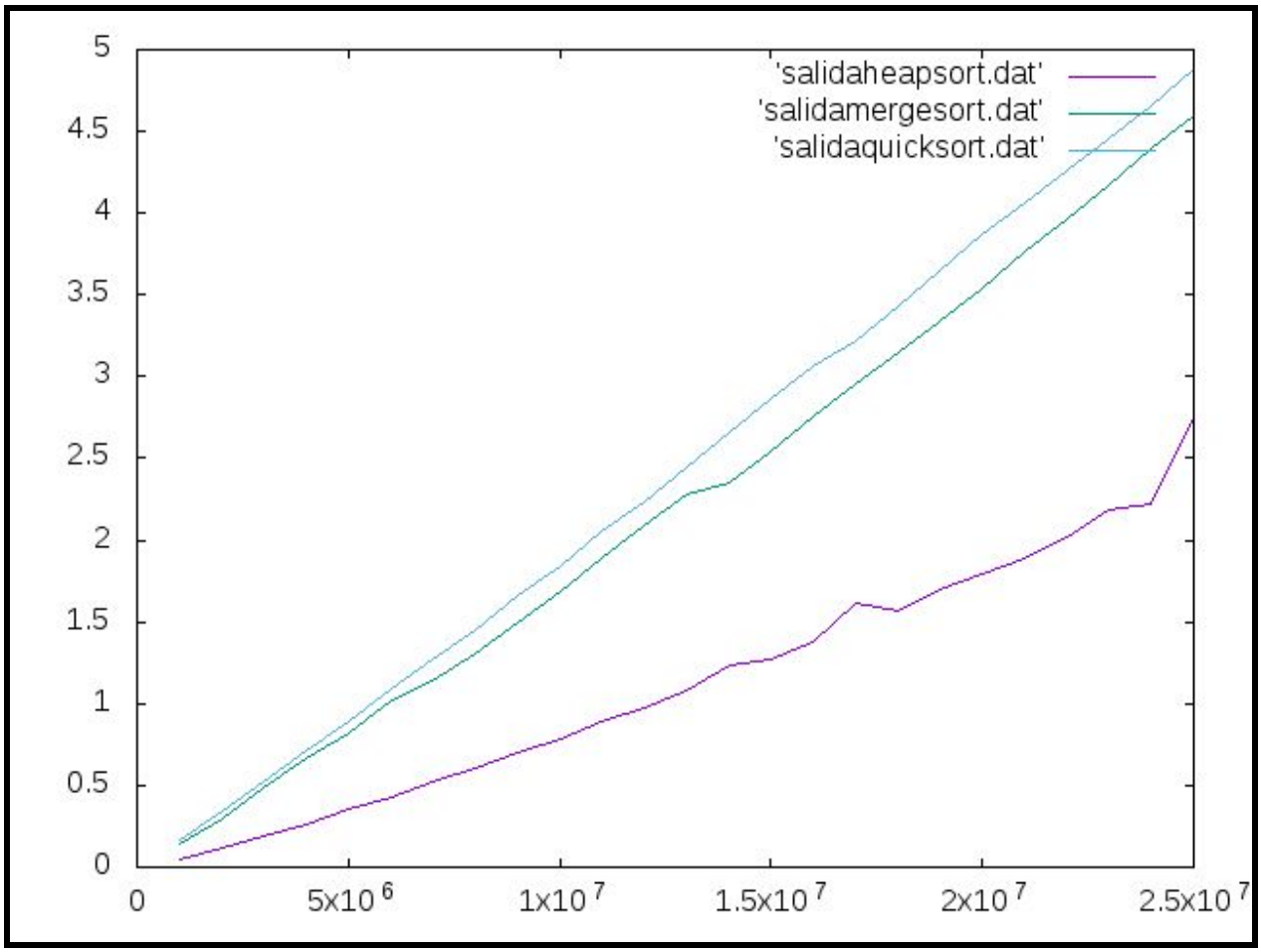
Los tres algoritmos cuadráticos en la misma gráfica.



Como se puede observar en la gráfica superior, se han representado los tres algoritmos cuadráticos juntos. De los tres algoritmos, para la misma cantidad de datos, el de Inserción es el más eficiente (tarda menos tiempo para cada cantidad de datos) mientras que el burbuja es menos eficiente (más tiempo).

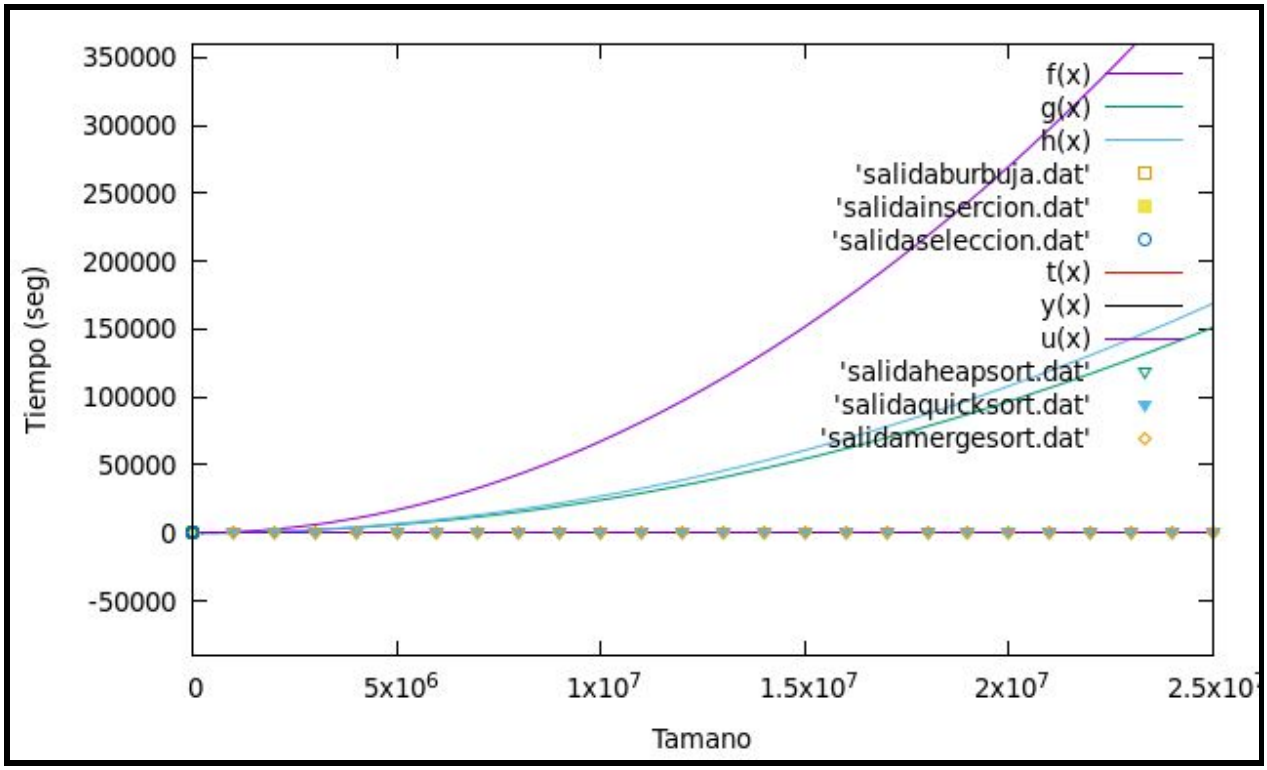
2.2 Comparativa O(n log n)

Los tres algoritmos logarítmicos en la misma gráfica:



Respecto a los algoritmos logarítmicos O(n log(n)), se puede observar que el algoritmo Heapsort es el más eficiente de los tres, siendo el Quicksort el menos eficiente.

2.3 Comparativa de algoritmos de ordenación



Como se puede ver, ajustando las funciones características a cada tabla de datos, se puede comprobar que para tiempos muy grandes, los algoritmos cuadráticos tardan mucho más tiempo que los logaritmos.

• f(x)=a0*x*x+a1*x+a2 • g(x)=b0*x*x+b1*x+b2 • h(x)=c0*x*x+c1*x+c2	• fit f(x) 'salidaburbuja.dat' via a0,a1,a2 • fit g(x) 'salidainsercion.dat' via b0,b1,b2 • fit h(x) 'salidaseleccion.dat' via c0,c1,c2
• t(x)=d0*x*log10(d1*x) • y(x)=e0*x*log10(e1*x) • u(x)=f0*x*log10(f1*x)	• fit t(x) 'salidaquicksort.dat' via d0,d1 • fit y(x) 'salidamergesort.dat' via e0,e1 • fit u(x) 'salidaheapsort.dat' via f0,f1
• plot f(x), g(x), h(x), 'salidaburbuja.dat', 'salidainat', 'salidaseleccion.dat', t(x), y(x), u(x), 'salidaheapsort.dat', 'salidaquicksort.dat', 'salidamergesort.dat'	

3. Calculo de la eficiencia hibrida
   3.1 Burbuja

Para calcular la eficiencia híbrida se va proceder a obtener las constantes ocultas, esto lo vamos a realizar con gnuplot, mediante la orden “fit”.

Primero se establece la ecuación general cuadrática y luego se ejecuta el ajuste, obteniendo el archivo siguiente:

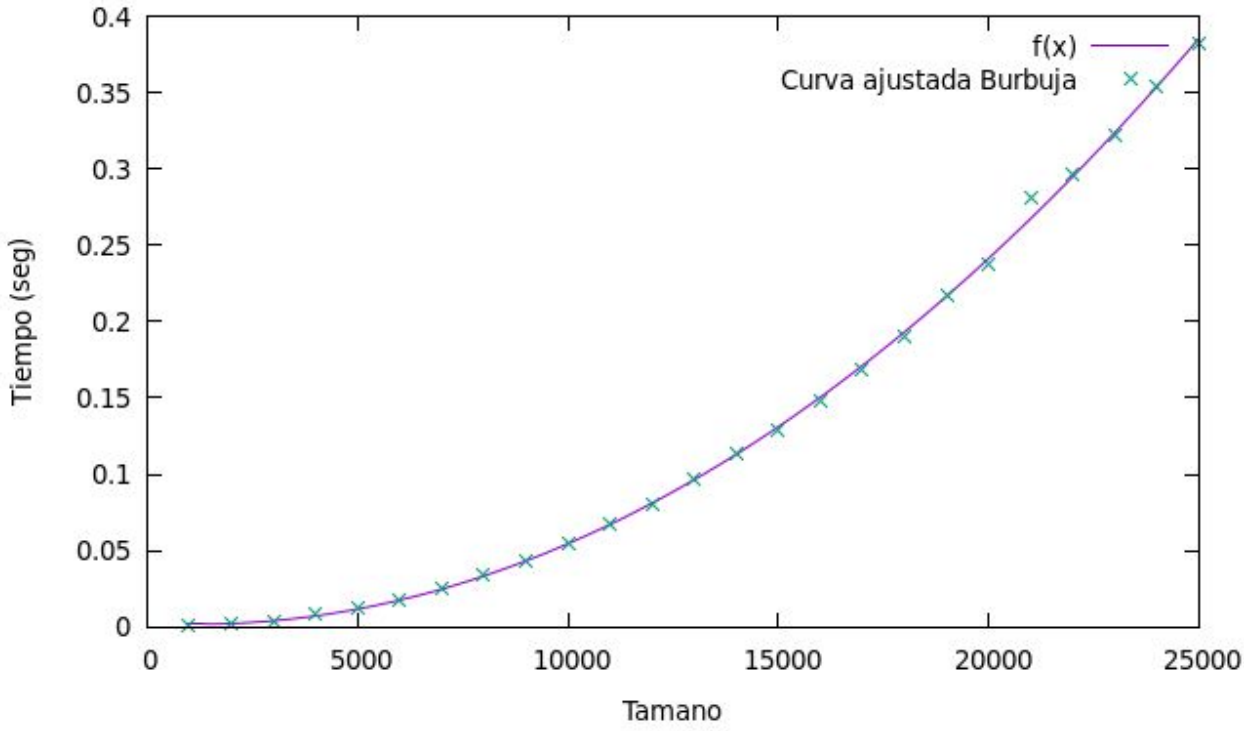
- f(x) = a0*x*x + a1*x + a2
- fit f(x) ’salida.dat’ via a0,a1,a2

```
*******************************************************************************
Thu Mar  7 17:36:57 2019


FIT:    data read from 'salidaburbuja.dat'
        format = z
        #datapoints = 25
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
  f(x)=a0*x*x+a1*x+a2
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a0             a1             a2
   0 2.1538562640e+18   0.00e+00  1.69e+08    1.000000e+00   1.000000e+00   1.000000e+00
  12 2.4143125811e-04  -6.19e-04  1.69e-04    6.750945e-10  -1.569066e-06   2.769525e-03

After 12 iterations the fit converged.
final sum of squares of residuals : 0.000241431
rel. change during last iteration : -6.19472e-09

degrees of freedom    (FIT_NDF)                      : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00331273
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.09741e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 6.75095e-10      +/- 1.428e-11    (2.115%)
a1              = -1.56907e-06     +/- 3.825e-07    (24.38%)
a2              = 0.00276952       +/- 0.002158     (77.92%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.971  1.000
a2              0.774 -0.884  1.000
```

En la parte inferior de la imagen se pueden ver los valores de las constante ocultas

- a0 = 6.75095 * 10^(-10)
- a1 = -1.56907 * 10^(-6)
- a2 = 0.00276952

Dicho esto, ahora ya se puede representar la eficiencia híbrida en una gráfica, representando la eficiencia empírica a partir de los datos obtenidos, y la curva característica correspondiente a la función cuadrática.



Como se puede ver, ambas curvas se ajustan.

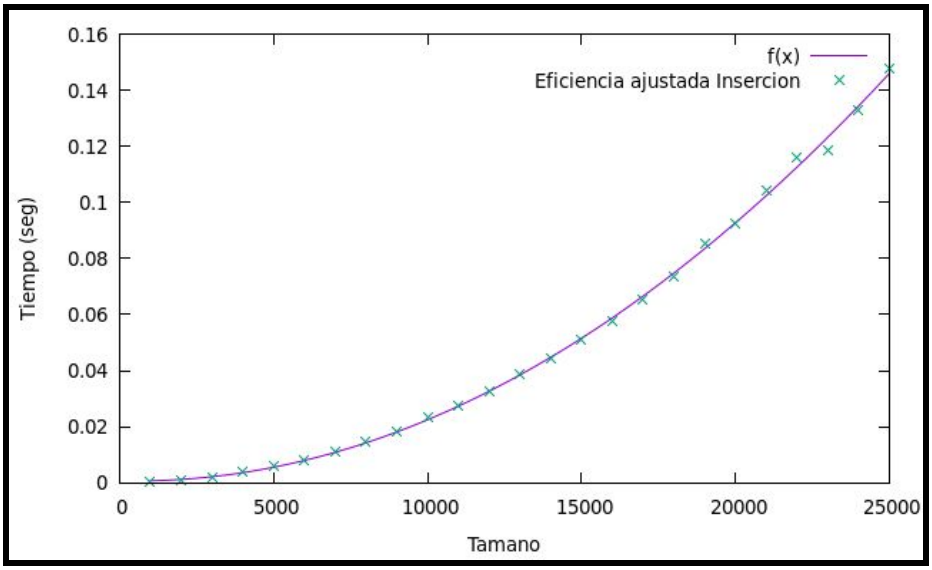
3.2 Insercion

Al igual que para el algoritmo burbuja, se han obtenido los siguientes valores:

```
*******************************************************************************
Thu Mar  7 18:07:03 2019


FIT:    data read from 'salidainsercion.dat'
        format = z
        #datapoints = 25
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
  f(x)=a0*x*x+a1*x+a2
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a0            a1            a2
   0 2.1538562656e+18   0.00e+00  1.69e+08    1.000000e+00   1.000000e+00   1.000000e+00
  12 5.1148431731e-05  -2.94e-03  1.69e-04    2.423095e-10  -2.496469e-07   7.839019e-04

After 12 iterations the fit converged.
final sum of squares of residuals : 5.11484e-05
rel. change during last iteration : -2.9357e-08

degrees of freedom    (FIT_NDF)                       : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00152477
variance of residuals (reduced chisquare) = WSSR/ndf   : 2.32493e-06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 2.4231e-10       +/- 6.573e-12    (2.712%)
a1              = -2.49647e-07     +/- 1.76e-07     (70.52%)
a2              = 0.000783902      +/- 0.0009933    (126.7%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.971  1.000
a2              0.774 -0.884  1.000
```

- a0 = 2.4231 * 10^(-10)
- a1 = -2.49647 * 10^(-7)
- a2 = 0.000783902

Con una grafica:



3.3 Seleccion

Para calcular la eficiencia híbrida se va proceder a obtener las constantes ocultas, esto lo vamos a realizar con gnuplot, mediante la orden “fit”.

Primero se establece la ecuación general cuadrática y luego se ejecuta el ajuste, obteniendo el archivo siguiente:

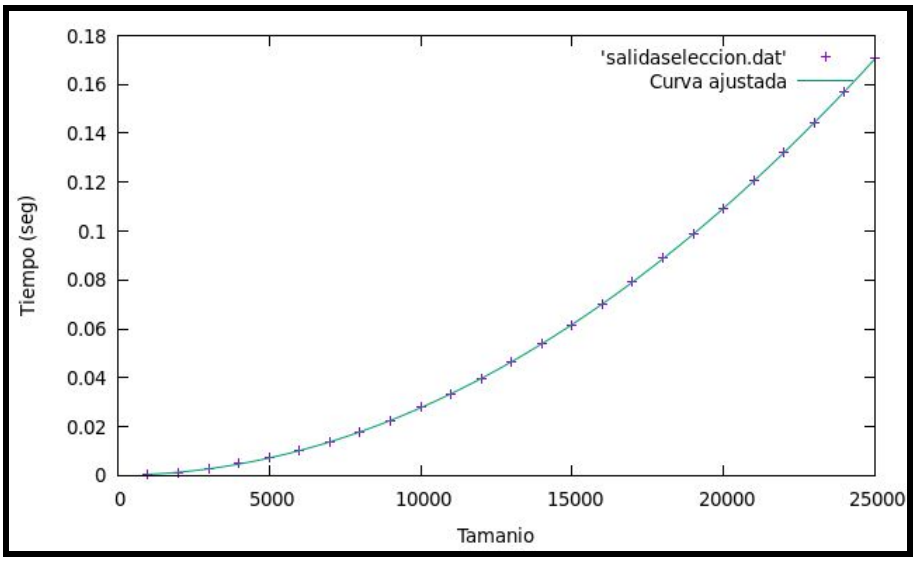
- f(x) = a0*x*x + a1*x + a2
- fit f(x) ’salida.dat’ via a0,a1,a2

```
*******************************************************************************
Thu Mar  7 19:23:13 2019


FIT:    data read from 'salidaseleccion.dat'
        format = z
        #datapoints = 25
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
  f(x) = a0*x*x+a1*x+a2
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a0            a1            a2
   0 2.1538562654e+18   0.00e+00  1.69e+08    1.000000e+00   1.000000e+00   1.000000e+00
  12 6.9329338896e-07  -2.17e-01  1.69e-04    2.704607e-10   5.144579e-08   1.386781e-04

After 12 iterations the fit converged.
final sum of squares of residuals : 6.93293e-07
rel. change during last iteration : -2.16864e-06

degrees of freedom    (FIT_NDF)                      : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00017752
variance of residuals (reduced chisquare) = WSSR/ndf   : 3.15133e-08

Final set of parameters            Asymptotic Standard Error
=======================            =========================
a0              = 2.70461e-10      +/- 7.652e-13    (0.2829%)
a1              = 5.14458e-08      +/- 2.05e-08     (39.84%)
a2              = 0.000138678      +/- 0.0001156    (83.39%)
```

En la parte inferior de la imagen se pueden ver los valores de las constante ocultas

- a0 = 2.70461 * 10^(-10)
- a1 = 5.14458 * 10^(-6)
- a2 = 0.00138678

Dicho esto, ahora ya se puede representar la eficiencia híbrida en una gráfica, representando la eficiencia empírica a partir de los datos obtenidos, y la curva característica correspondiente a la función cuadrática.



3.4 Mergesort

Para calcular la eficiencia híbrida se va proceder a obtener las constantes ocultas, esto lo vamos a realizar con gnuplot, mediante la orden “fit”.

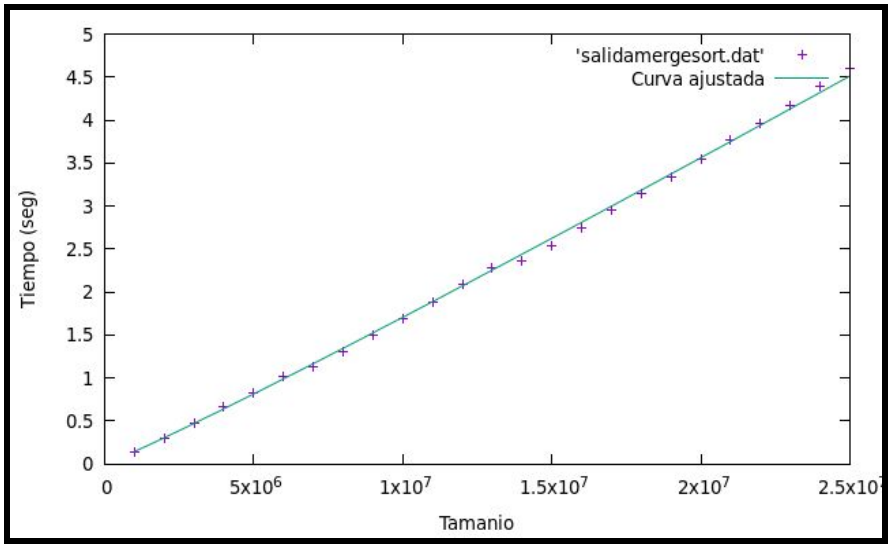
Primero se establece la ecuación general cuadrática y luego se ejecuta el ajuste, obteniendo el archivo siguiente:

```
*******************************************************************************
Mon Mar 11 20:10:56 2019


FIT:    data read from 'salidamergesort.dat'
        format = z
        #datapoints = 25
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
    f(x)=a0*x*log10(a1*x)
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a0            a1
   0 1.7350460518e+02   0.00e+00  1.30e-03    2.704607e-10   5.144579e-08
 365 3.5043365880e-02  -8.50e-01  1.30e-08    3.360828e-08   1.016659e-02

After 365 iterations the fit converged.
final sum of squares of residuals : 0.0350434
rel. change during last iteration : -8.49529e-06

degrees of freedom    (FIT_NDF)                        : 23
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0390336
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00152362

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 3.36083e-08      +/- 3.63e-09     (10.8%)
a1              = 0.0101666        +/- 0.01333      (131.2%)

correlation matrix of the fit parameters:
                a0     a1
a0              1.000
a1             -1.000  1.000
```

En la parte inferior de la imagen se pueden ver los valores de las constante ocultas

- a0 =3.36083* 10^(-8)
- a1 = 0.101666

Dicho esto, ahora ya se puede representar la eficiencia híbrida en una gráfica, representando la eficiencia empírica a partir de los datos obtenidos, y la curva característica correspondiente a la función cuadrática.


'salidamergesort.dat'
Curva ajustada
Tiempo (seg)
Tamanio
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5x10^6
1x10^7
1.5x10^7
2x10^7

3.5 Quicksort

Para calcular la eficiencia híbrida del algoritmo quicksort se va proceder a obtener las constantes ocultas, esto lo vamos a realizar con gnuplot, mediante la orden “fit”.

Primero se establece la ecuación general cuadrática y luego se ejecuta el ajuste, obteniendo el archivo siguiente:

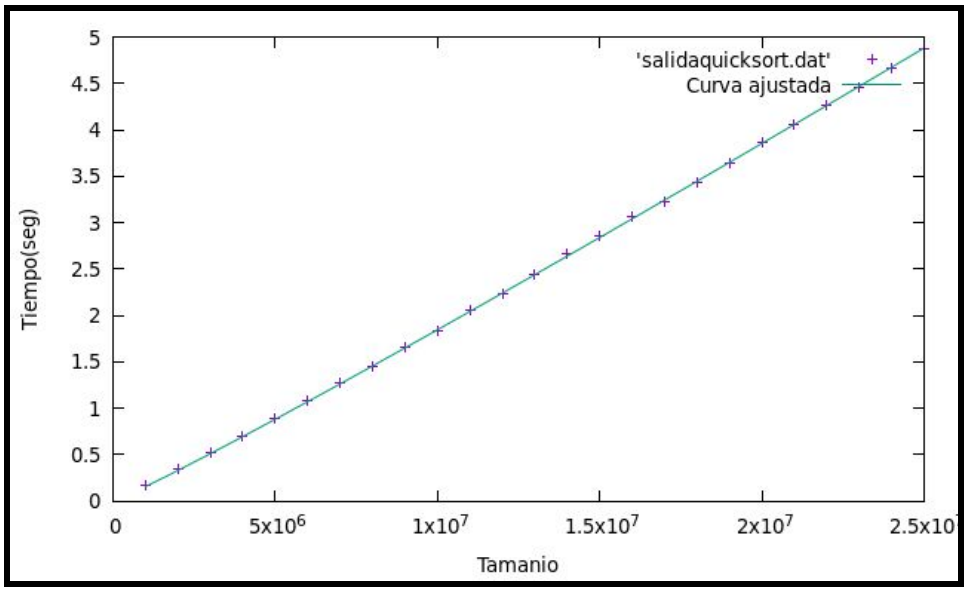
- f(x) = b0*x*log10(b1*x)
- fit f(x) ’salida.dat’ via b0,b1

```
*******************************************************************************
Thu Mar  7 17:53:41 2019


FIT:    data read from "salidaquicksort.dat"
        format = z
        #datapoints = 25
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
   f(x)=b0*x*log10(b1*x)
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   b0             b1
   0 2.9145654663e+17   0.00e+00  7.65e+07    1.000000e+00   1.000000e+00
   5 3.9808169432e-03  -2.74e-07  7.65e+02    2.648269e-08   9.415971e-01

After 5 iterations the fit converged.
final sum of squares of residuals : 0.00398082
rel. change during last iteration : -2.74427e-12

degrees of freedom    (FIT_NDF)                        : 23
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0131559
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000173079

Final set of parameters            Asymptotic Standard Error
=======================            =========================
b0              = 2.64827e-08      +/- 1.224e-09    (4.62%)
b1              = 0.941597         +/- 0.7253       (77.03%)

correlation matrix of the fit parameters:
                b0     b1
b0              1.000
b1             -1.000  1.000
```

En la parte inferior de la imagen se pueden ver los valores de las constante ocultas

- b0 = 2.64827 * 10^(-8)
- b1 = 0.941597

Dicho esto, ahora ya se puede representar la eficiencia híbrida en una gráfica, representando la eficiencia empírica a partir de los datos obtenidos y función obtenidos.

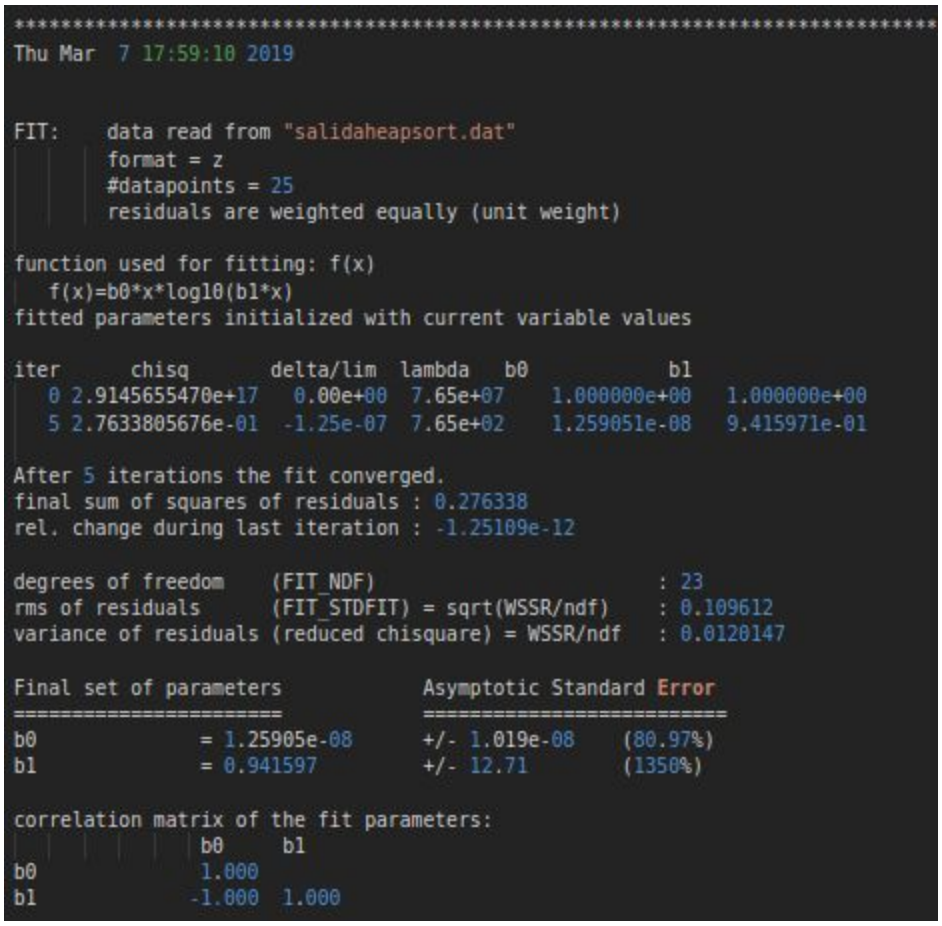


3.6 Heapsort

Para calcular la eficiencia híbrida del algoritmo heapsort se va proceder a obtener las constantes ocultas, esto lo vamos a realizar con gnuplot, mediante la orden “fit”.

Primero se establece la ecuación general cuadrática y luego se ejecuta el ajuste, obteniendo el archivo siguiente:

- f(x) = b0*x*log10(b1*x)
- fit f(x) ’salida.dat’ via b0,b1

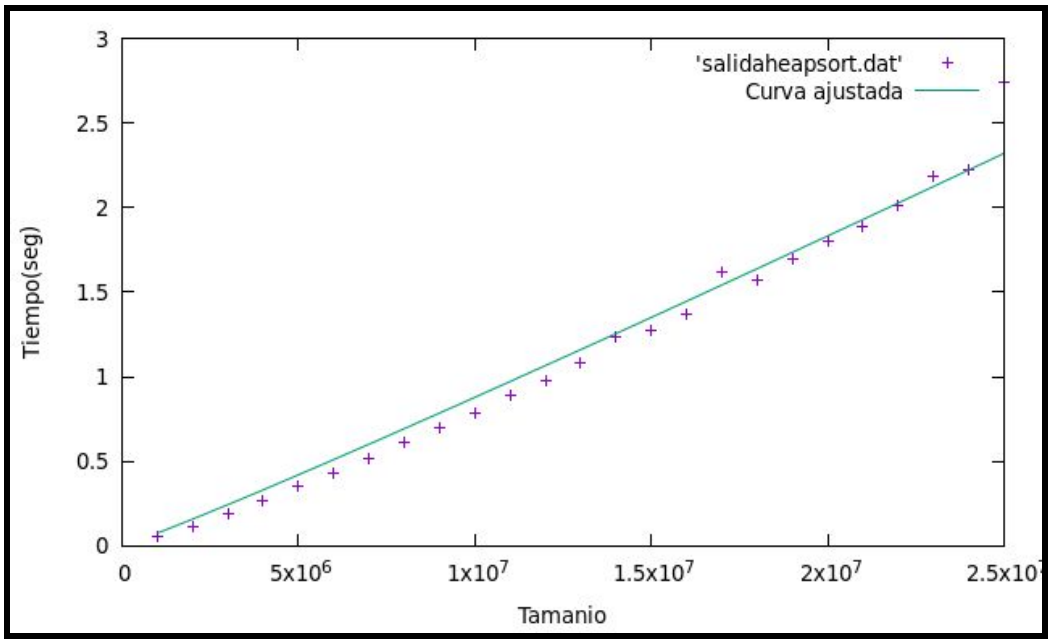


```
*******************************************************************************
Thu Mar  7 17:59:10 2019


FIT:    data read from "salidaheapsort.dat"
        format = z
        #datapoints = 25
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
   f(x)=b0*x*log10(b1*x)
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   b0             b1
   0 2.9145655470e+17   0.00e+00  7.65e+07    1.000000e+00   1.000000e+00
   5 2.7633805676e-01  -1.25e-07  7.65e+02    1.259051e-08   9.415971e-01

After 5 iterations the fit converged.
final sum of squares of residuals : 0.276338
rel. change during last iteration : -1.25109e-12

degrees of freedom    (FIT_NDF)                        : 23
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.109612
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.0120147

Final set of parameters            Asymptotic Standard Error
=======================            =========================
b0              = 1.25905e-08      +/- 1.019e-08    (80.97%)
b1              = 0.941597         +/- 12.71        (1350%)

correlation matrix of the fit parameters:
                b0     b1
b0              1.000
b1             -1.000  1.000
```

En la parte inferior de la imagen se pueden ver los valores de las constante ocultas

- b0 = 1.2595 * 10^(-8)
- b1 = 0.941597

Dicho esto, ahora ya se puede representar la eficiencia híbrida en una gráfica, representando la eficiencia empírica a partir de los datos obtenidos y función obtenidos.



3.7 Floyd

```
*******************************************************************************
Thu Mar  7 18:05:53 2019


FIT:    data read from 'salidafloyd.dat'
        format = z
        #datapoints = 25
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
  f(x)=a0*x*x*x+a1*x*x+a2*x+a3
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a0            a1            a2            a3
   0 1.5636145865e+19   0.00e+00  3.95e+08    1.000000e+00   1.000000e+00   1.000000e+00   1.000000e+00
  13 2.8179540503e-04  -1.87e-09  3.95e-05    1.283275e-09  -1.296674e-07   4.223707e-05  -2.330342e-03

After 13 iterations the fit converged.
final sum of squares of residuals : 0.000281795
rel. change during last iteration : -1.86603e-14

degrees of freedom    (FIT_NDF)                       : 21
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00366317
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.34188e-05

Final set of parameters           Asymptotic Standard Error
=======================           =========================
a0              = 1.28328e-09      +/- 2.007e-11    (1.564%)
a1              = -1.29667e-07     +/- 3.965e-08    (30.58%)
a2              = 4.22371e-05      +/- 2.241e-05    (53.07%)
a3              = -0.00233034      +/- 0.003432     (147.3%)

correlation matrix of the fit parameters:
               a0     a1     a2     a3
a0              1.000
a1             -0.987  1.000
a2              0.926 -0.973  1.000
a3             -0.719  0.795 -0.898  1.000
```

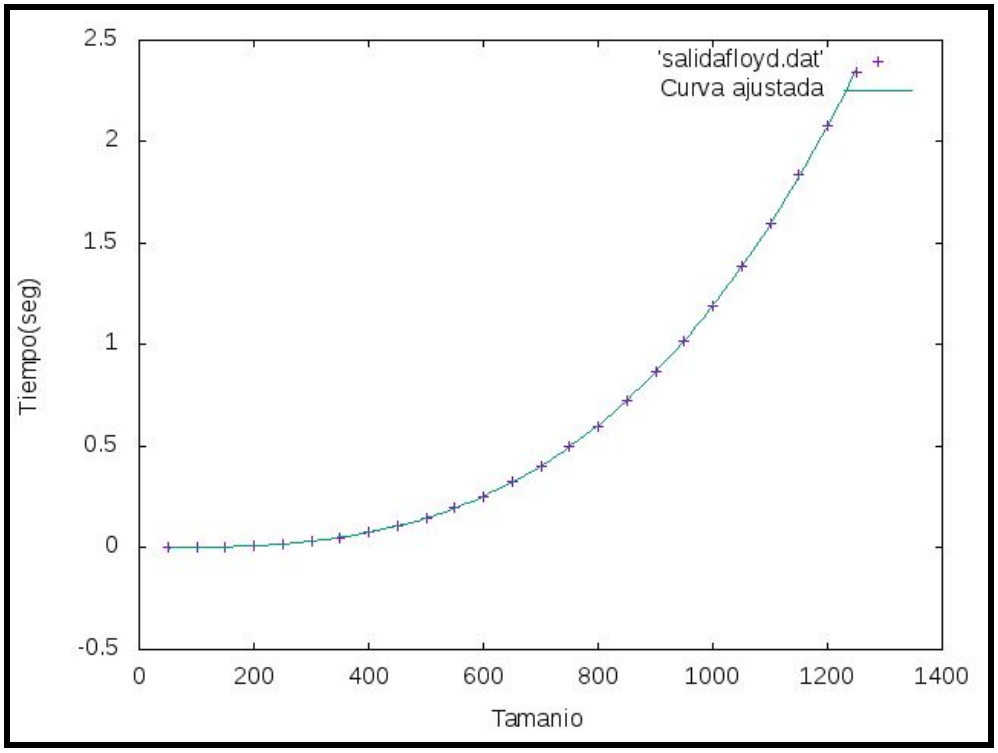
Al ser de orden cúbica la función a la que hemos ajustado el algoritmo es:

```
function used for fitting: f(x)
  f(x) = a0*x*x*x + a1*x*x + a2*x + a3
```

Con las siguientes constantes ocultas:

```
Final set of parameters           Asymptotic Standard Error
=======================           =========================
a0              = 1.28328e-09      +/- 2.007e-11    (1.564%)
a1              = -1.29667e-07     +/- 3.965e-08    (30.58%)
a2              = 4.22371e-05      +/- 2.241e-05    (53.07%)
a3              = -0.00233034      +/- 0.003432     (147.3%)
```

Ajustando a la función:



3.8 Fibonacci

```
*******************************************************************************
Thu Mar  7 18:00:26 2019


FIT:    data read from 'salidafibonacci.dat'
        format = z
        #datapoints = 26
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
   f(x) = a0*((1+sqrt(5))/2)**x
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a0
   0 1.2354941342e-03   0.00e+00  5.93e+00    6.132052e-10
   1 1.2354941342e-03   0.00e+00  5.93e+03    6.132052e-10

After 1 iterations the fit converged.
final sum of squares of residuals : 0.00123549
rel. change during last iteration : 0

degrees of freedom    (FIT_NDF)                        : 25
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00702992
variance of residuals (reduced chisquare) = WSSR/ndf   : 4.94198e-05

Final set of parameters            Asymptotic Standard Error
=======================            =========================
a0              = 6.13205e-10      +/- 1.427e-13    (0.02327%)
```

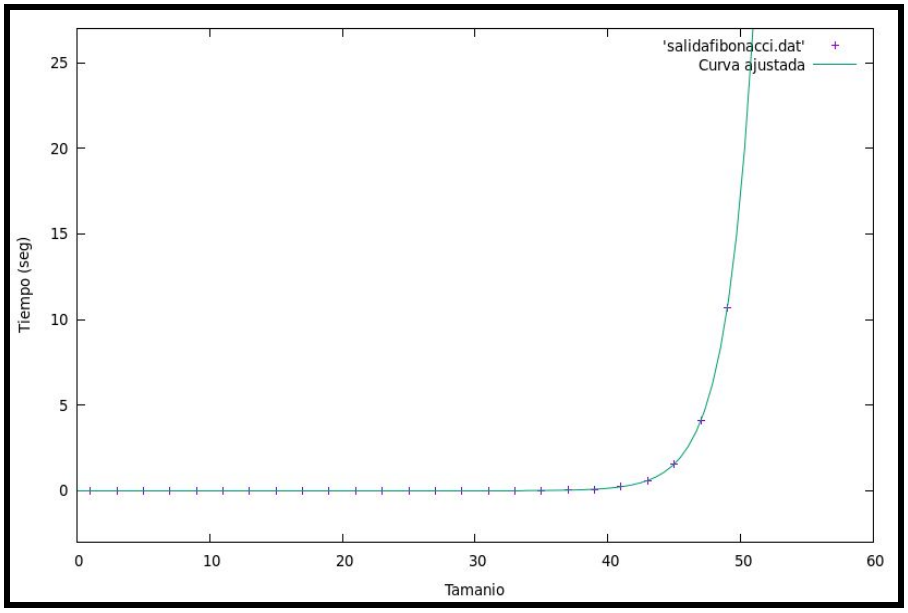
Como este logaritmo es de orden exponencial el ajuste que le hemos hecho es:

```
function used for fitting: f(x)
  f(x) = a0*((1+sqrt(5))/2)**x
```

Con un ajuste de constantes ocultas:

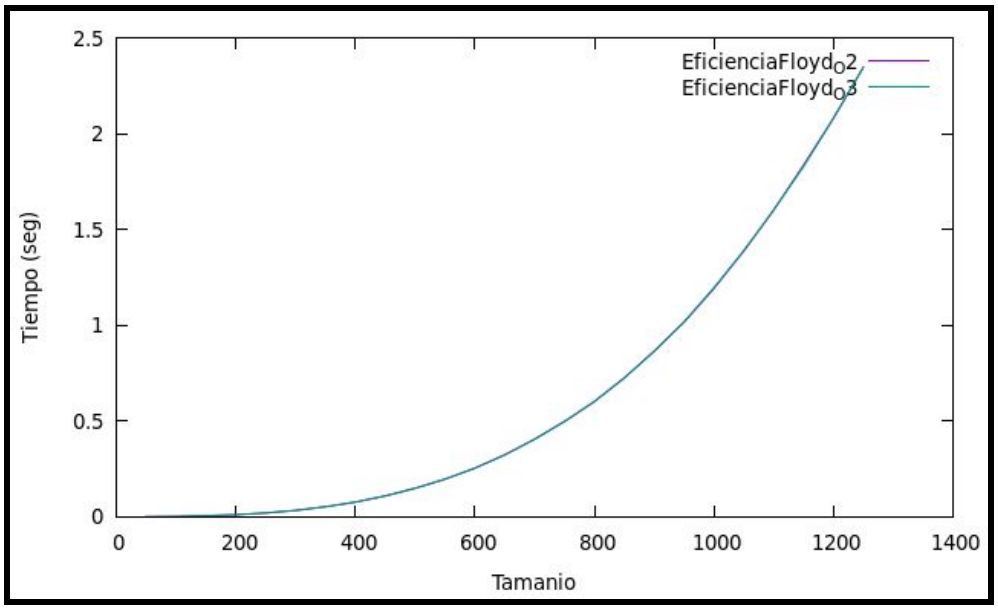
```
Final set of parameters            Asymptotic Standard Error
=======================            =========================
a0              = 6.13205e-10      +/- 1.427e-13    (0.02327%)
```

Y ajustando las funciones nos quedaría:



4. Comparativa con diferentes parámetros externos

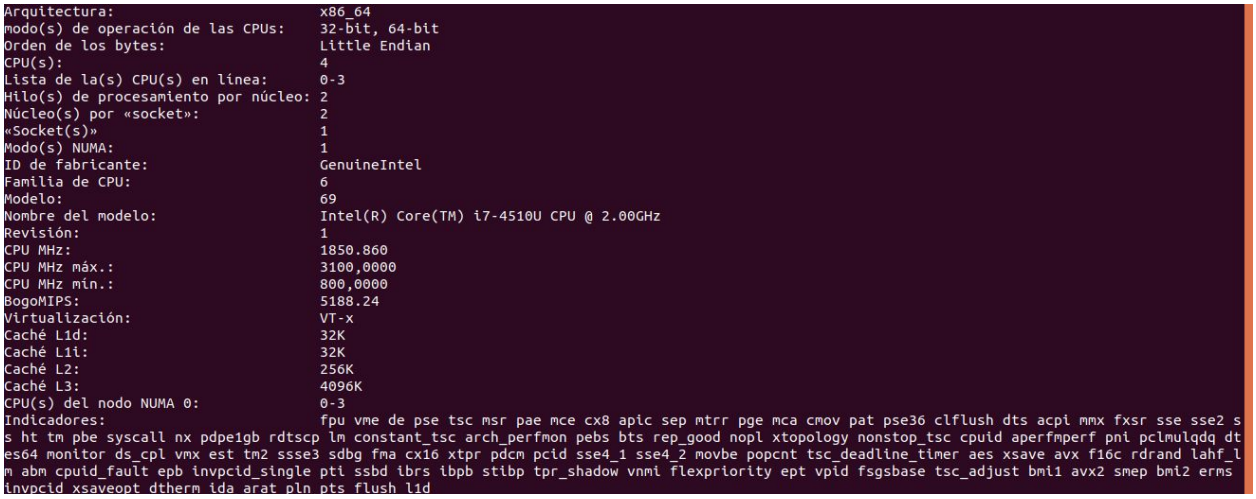
   4.1 Comparativa del algoritmo Floyd en el mismo equipo con diferente optimización de compilación



Como se puede observar en esta gráfica los tiempos de la ejecución de Floyd con optimización O3 o O2 son iguales, por lo que esto quiere decir que no se produce una mejora de tiempo al pasar de O2 a O3 como sí sucede con otros algoritmos.

4.2 Comparativa de los algoritmos con eficiencia O(n²) en diferentes equipos

Por último vamos a comparar los algoritmos con eficiencia O(n²) en dos equipos, las características del primero son se aprecian en la siguiente imagen, destacando que tiene un procesador Intel I7-4510U a 2 GHz.



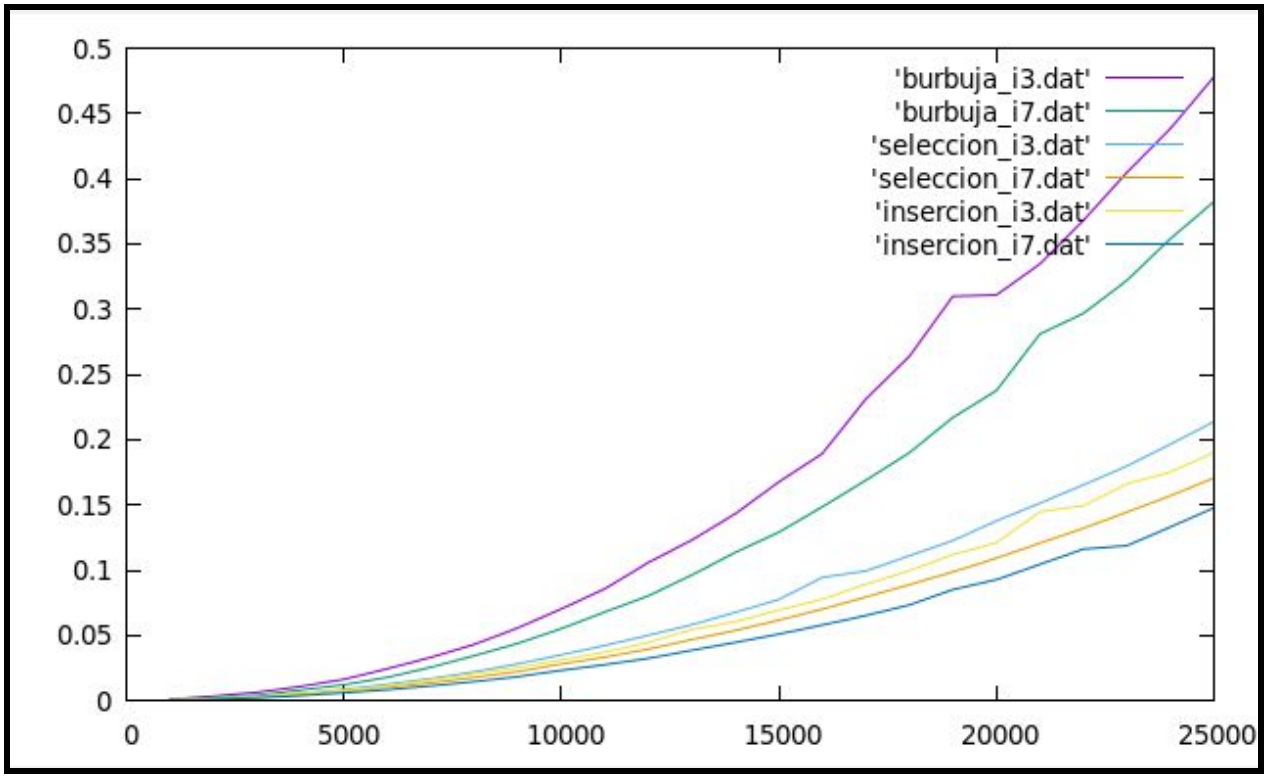
```
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
CPU(s):                              4
Lista de la(s) CPU(s) en línea:      0-3
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:              2
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              69
Nombre del modelo:                   Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
Revisión:                            1
CPU MHz:                             1850.860
CPU MHz máx.:                        3100,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            5188.24
Virtualización:                      VT-x
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            4096K
CPU(s) del nodo NUMA 0:              0-3
Indicadores:                         fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 s
s ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dt
es64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_l
m abm cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
invpcid xsaveopt dtherm ida arat pln pts flush_l1d
```

Las características del segundo equipo son las que muestra la siguiente imagen:

```
sixto@sixtoU:~$ lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPUs:32-bit, 64-bit
Orden de bytes:        Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Hilo(s) de procesamiento por núcleo:2
Núcleo(s) por «socket»:2
Socket(s):             1
Modo(s) NUMA:          1
ID de fabricante:      GenuineIntel
Familia de CPU:        6
Modelo:                58
Model name:            Intel(R) Core(TM) i3-3120M CPU @ 2.50GHz
Revisión:              9
CPU MHz:               1196.156
CPU max MHz:           2500,0000
CPU min MHz:           1200,0000
BogoMIPS:              4983.87
Virtualización:        VT-x
Caché L1d:             32K
Caché L1i:             32K
Caché L2:              256K
Caché L3:              3072K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe s
yscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl
vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsave avx f16c lahf_lm cpuid_fault epb pti ssbd ibrs ibpb s
tibp tpr_shadow vnmi flexpriority ept vpid fsgsbase smep erms xsaveopt dtherm arat pln pts flush_l1d
```

Este segundo equipo tiene un procesador Intel I3-3120M a 2.5GHz

Al ejecutar los algoritmos de burbuja, insercion y seleccion en ambos equipos hemos obtenido las siguiente gráfica:



En ella se puede apreciar como el equipo con el procesador i7 obtiene mejores resultados para todos los algoritmos que el equipo con el procesador i3.