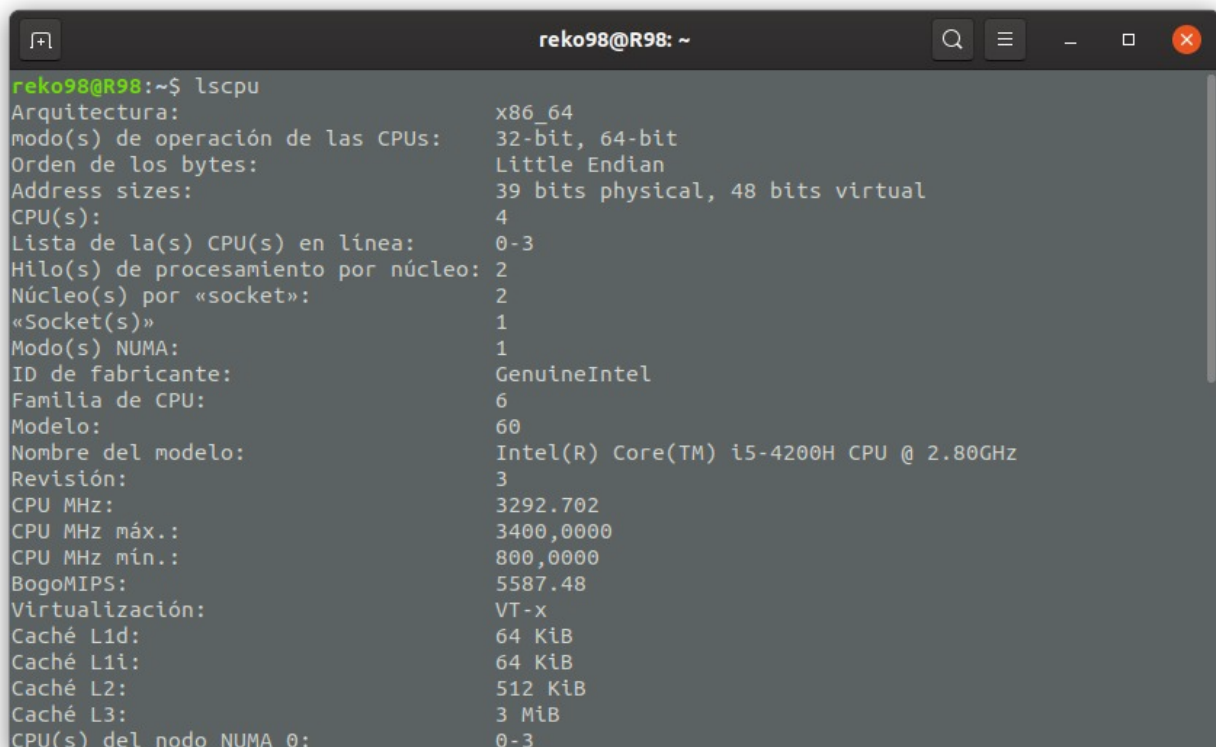


# 0.-Ordenador de pruebas

En primer lugar, las características del ordenador donde se ha realizado la práctica 1 son:

- Procesador Intel Core i5-4200H con una velocidad base de 2.8Ghz y con velocidad turbo máxima de 3.4Ghz, con 2 cores físicos y 4 lógicos.
- Memoria RAM de 8Gb
- Sistema Operativo Ubuntu 19.10 montado en una arquitectura de 64 bits.



```
reko98@R98:~$ lscpu
Arquitectura:                x86_64
modo(s) de operación de las CPUs:  32-bit, 64-bit
Orden de los bytes:          Little Endian
Address sizes:               39 bits physical, 48 bits virtual
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:                      60
Nombre del modelo:            Intel(R) Core(TM) i5-4200H CPU @ 2.80GHz
Revisión:                    3
CPU MHz:                     3292.702
CPU MHz máx.:                 3400,0000
CPU MHz mín.:                 800,0000
BogoMIPS:                    5587.48
Virtualización:               VT-x
Caché L1d:                   64 KiB
Caché L1i:                   64 KiB
Caché L2:                    512 KiB
Caché L3:                    3 MiB
CPU(s) del nodo NUMA 0:       0-3
```

*Especificaciones del sistema*

Se van a estudiar durante estas pruebas los algoritmos de ordenación *bubblesort* y *quicksort* además de el algoritmo de Floyd para calcular el costo del camino minimo entre cada par de nodos de un grafo dirigido y el algoritmo de resolución de las torres de hanoi.

Previamente, sabemos el orden de eficiencia de cada uno de ellos, siendo los siguientes:

Algoritmo	Eficiencia
Bubblesort	$O(n^2)$
Quicksort	$O(n \log n)$
Floyd	$O(n^3)$
Hanoi	$O(2^n)$

# 1.-Eficiencia empírica

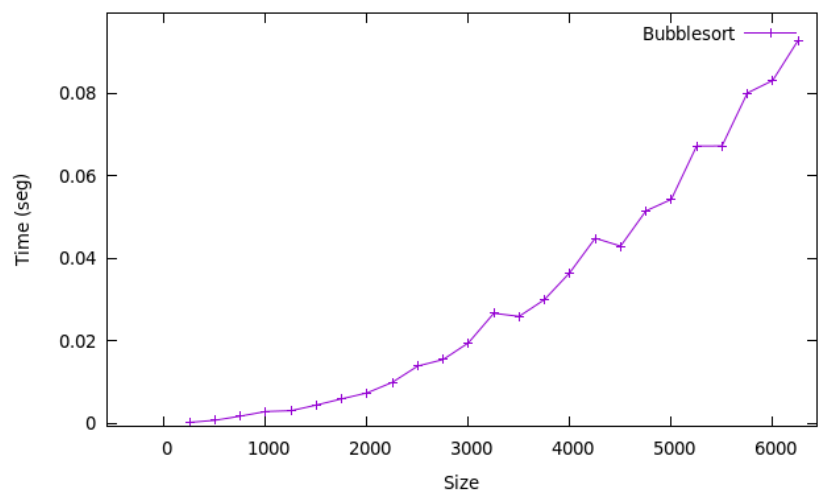
## 1.1.-Bubblesort

Para el algoritmo de ordenación de burbuja se ha implementado un script que introduce los datos en el rango 250-6250 con incrementos de 250 en cada iteración. Con estos datos se ha realizado una tabla con los resultados obtenidos:

Tamaño	Tiempo (seg)
250	0.000266
500	0.000786
750	0.001788
1000	0.002916
1250	0.003102
1500	0.004484
1750	0.005982
2000	0.007424
2250	0.009989
2500	0.013918
2750	0.013918
3000	0.019549
3000	0.019549
3500	0.019549
3750	0.030019
4000	0.036554
4250	0.044872
4500	0.042957
4750	0.051492
5000	0.054281
5250	0.054281
5500	0.067216
5500	0.067216
6000	0.067216
6250	0.092766

Se ha generado una gráfica para representar los resultados de la tabla y poder observar como se comporta el tiempo del algoritmo en función del tamaño de los datos. Dicha gráfica se ha generado mediante *gnuplot*.

En ella podemos apreciar que, aunque existen ciertas imperfecciones, la traza de los datos se asemeja bastante a los de una función cuadrática, algo bastante lógico teniendo en cuenta que el orden de eficiencia de este algoritmo es cuadrático.

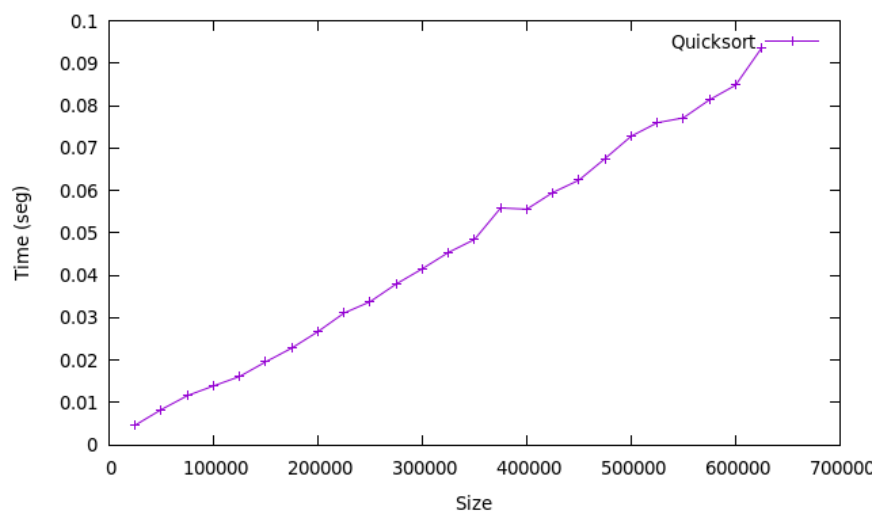


## 1.2.-Quicksort

Para el algoritmo de ordenación quicksort se ha implementado un script que introduce los datos en el rango 25000-625000 con incrementos de 25000 en cada iteración. Con estos datos se ha realizado una tabla con los resultados obtenidos:

Tamaño	Tiempo (seg)
25000	0.004692
50000	0.008373
75000	0.011666
100000	0.013956
125000	0.016169
150000	0.019633
175000	0.022871
200000	0.026747
225000	0.031115
250000	0.033801
275000	0.037986
300000	0.041555
325000	0.045409
350000	0.048488
375000	0.055878
400000	0.055589
425000	0.059543
450000	0.062453
475000	0.067535
500000	0.072798
525000	0.075969
550000	0.077118
570000	0.081381
600000	0.084833
625000	0.093619

En este caso también se ha generado una gráfica para poder observar el comportamiento del algoritmo. Apreciamos que en este caso la forma de la curva de los resultados también son parecidos a los de  $n \log n$ . Destacar también que como podemos observar en la tabla, este algoritmo tiene valores de tiempos pequeños en cuanto a los demás algoritmos aunque el tamaño de los datos de entrada sea muchísimo mayor que en las demás pruebas.



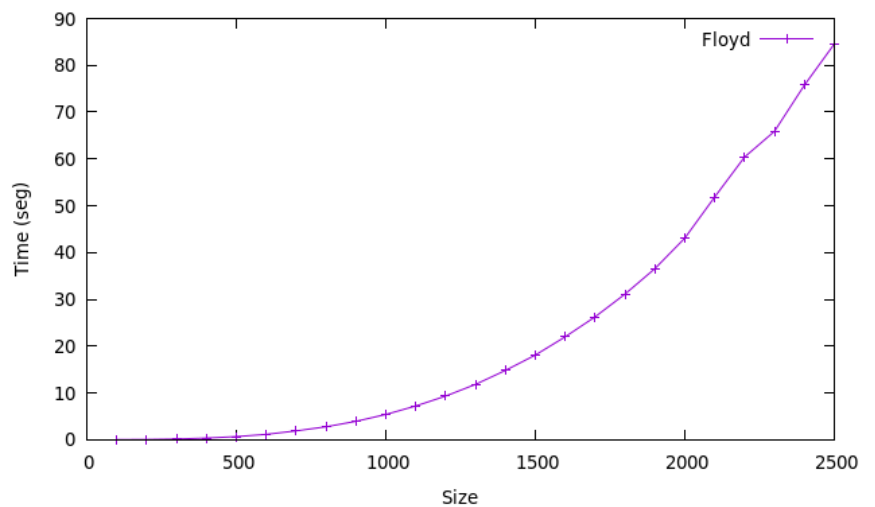
## 1.3.-Floyd

Para el algoritmo de *Floyd* se ha implementado un script que introduce los datos en el rango 100-2500 con incrementos de 100 en cada iteración. Con estos datos se ha realizado una tabla con los resultados obtenidos:

Tamaño	Tiempo (seg)
100	0.008893
200	0.046014
300	0.145983
400	0.339419
500	0.659791
600	1.13431
700	1.88486
800	2.75647
900	3.92273
1000	5.38891
1100	7.20579
1200	9.32713
1300	11.8215
1400	14.7984
1500	18.0878
1600	21.9879
1700	26.2482
1800	31.1118
1900	36.5515
2000	43.0791
2100	51.8562
2200	60.4045
2300	65.8979
2400	75.7588
2500	84.5639

En la gráfica del algoritmo de *Floyd* observamos que la curva de los resultados se parecen a los de una función  $x^3$ .

También si nos fijamos en la tabla, hemos comenzado a reducir el tamaño de los datos debido a que el tiempo de ejecución del programa comienza a ser considerablemente mayor que los algoritmos *bubblesort* y *quicksort*.

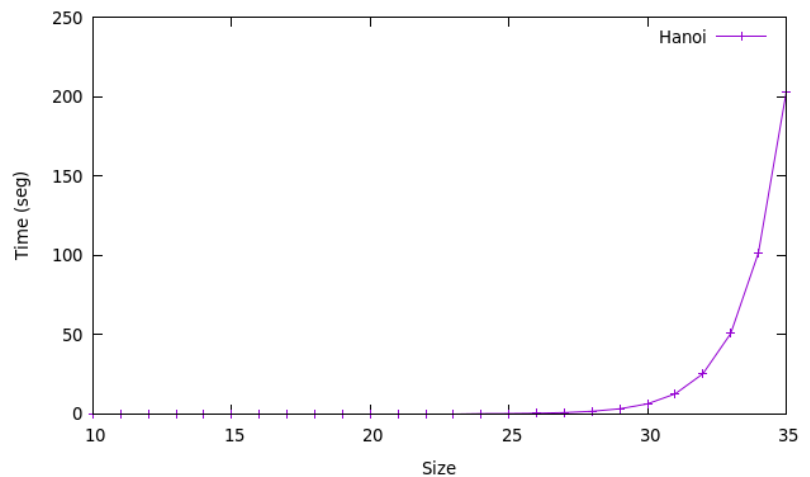


## 1.4.-Hanoi

Para el algoritmo de *Hanoi* se ha implementado un script que introduce los datos en el rango 11-35 con un solo incremento en cada iteración. Con estos datos se ha realizado una tabla con los los resultados obtenidos:

Tamaño	Tiempo (seg)
11	1.6e-05
12	3.1e-05
13	6e-05
14	0.000119
15	0.000212
16	0.000422
17	0.000854
18	0.001926
19	0.00343
20	0.006205
21	0.012729
22	0.024877
23	0.050107
24	0.09995
25	0.198019
26	0.395314
27	0.813111
28	1.62952
29	3.20465
30	6.34734
31	12.6632
32	25.3503
33	50.6367
34	101.461
35	202.457

En la gráfica del algoritmo de *Hanoi* observamos que su crecimiento es mínimo durante las 16 primeras entradas y que a partir de la entrada 17 el tiempo empieza a subir de manera considerable. Es un comportamiento común teniendo en cuenta que el orden de eficiencia de este algoritmo es  $2^n$ . En este algoritmo se ha tenido que reducir el tamaño de los datos de entrada drásticamente debido a que si usamos entradas mucho mas grandes probablemente aún estaríamos corriendo el programa.



## 2.-Eficiencia híbrida

Para el cálculo de la eficiencia híbrida se ha utilizado la función *gnuplot* disponible desde la terminal de ubuntu. Específicamente se ha usado la orden *fit* con una función predefinida anteriormente junto con los datos de entrada de datos.

En cada apartado perteneciente al algoritmo se van a incluir capturas con la información referente al resultado del ajuste realizado y una gráfica representando tanto los datos de entrada como la curva ajustada correspondiente a la eficiencia de cada algoritmo.

### 2.1.-Bubblesort

Para bubblesort se ha utilizado la función:

$$f(x) = ax^2 + bx + c$$

Los resultados del ajuste son:

```

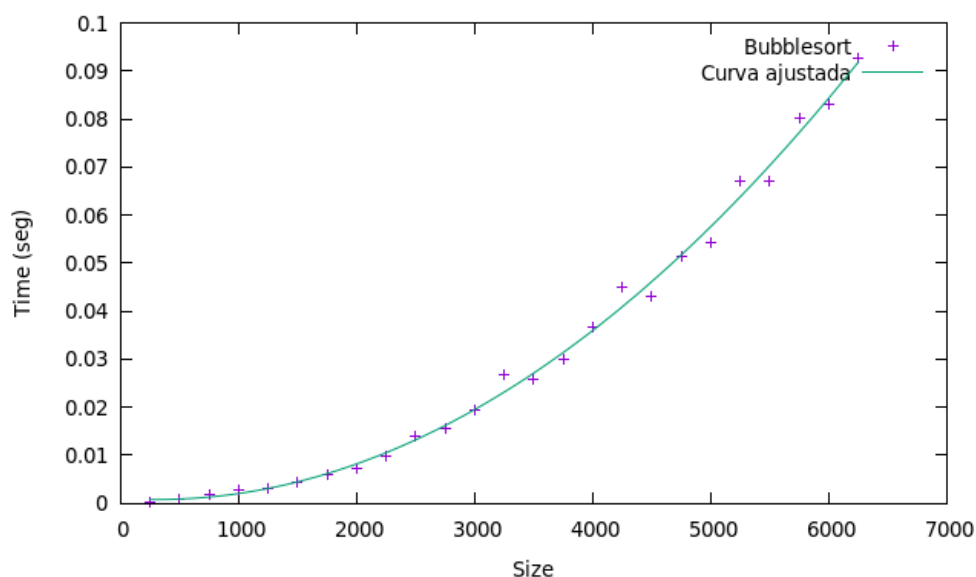
reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos
After 11 iterations the fit converged.
final sum of squares of residuals : 8.81135e-05
rel. change during last iteration : -1.35135e-10

degrees of freedom      (FIT_NDF)           : 22
rms of residuals        (FIT_STDFIT) = sqrt(WSSR/ndf) : 0.00200129
variance of residuals (reduced chisquare) = WSSR/ndf : 4.00516e-06

Final set of parameters      Asymptotic Standard Error
=====
a          = 2.57345e-09      +/- 1.38e-10      (5.363%)
b          = -1.55417e-06     +/- 9.242e-07     (59.47%)
c          = 0.001057        +/- 0.001304      (123.3%)

correlation matrix of the fit parameters:
      a      b      c
a      1.000
b     -0.971  1.000
c      0.774 -0.884  1.000
gnuplot>          input select error
  
```

Y la gráfica resultante superponiendo ambos datos (función  $f(x)$  y datos de entrada obtenidos):



## 2.2.-Quicksort

Para *quicksort* se ha utilizado la función:

$$f(x) = a * x \log(b * x) + c$$

Los resultados del ajuste son:

```

reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos
After 9 iterations the fit converged.
final sum of squares of residuals : 4.17925e-05
rel. change during last iteration : -1.69259e-12

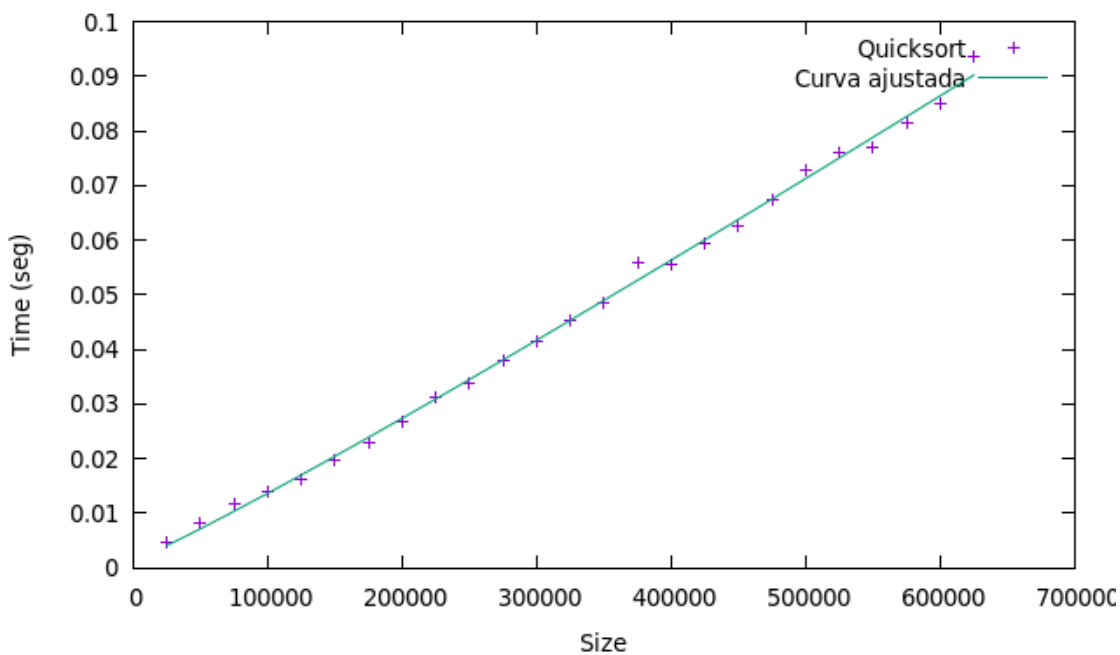
degrees of freedom (FIT_NDF) : 22
rms of residuals (FIT_STDFIT) = sqrt(WSSR/ndf) : 0.00137828
variance of residuals (reduced chisquare) = WSSR/ndf : 1.89966e-06

Final set of parameters      Asymptotic Standard Error
=====
a = 1.0644e-08              +/- 1.479e-10 (1.389%)
b = -0.731919              +/- 5.749e+09 (7.855e+11%)
c = 0.00145866             +/- 61.2 (4.196e+06%)

correlation matrix of the fit parameters:
      a      b      c
a      1.000
b      0.648  1.000
c     -0.648 -1.000  1.000
gnuplot>

```

Y la gráfica resultante superponiendo ambos datos (función  $f(x)$  y datos de entrada obtenidos):



## 2.3.-Floyd

Para *Floyd* se ha utilizado la función:

$$f(x) = ax^3 + bx^2 + cx + d$$

Los resultados del ajuste son:

```
reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos
After 13 iterations the fit converged.
final sum of squares of residuals : 9.67937
rel. change during last iteration : -3.3433e-10

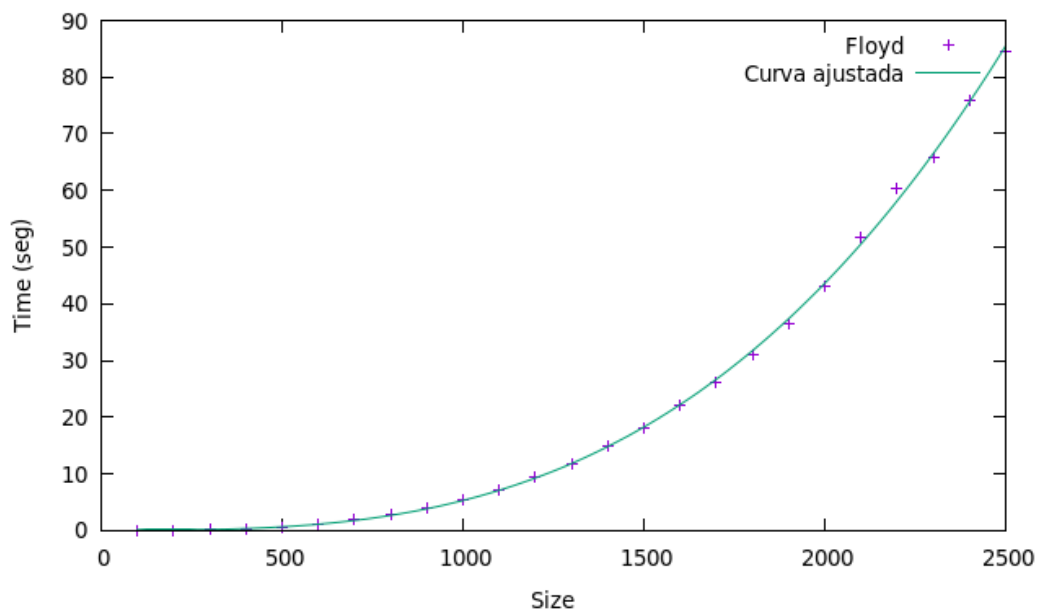
degrees of freedom (FIT_NDF) : 21
rms of residuals (FIT_STDFIT) = sqrt(WSSR/ndf) : 0.678913
variance of residuals (reduced chisquare) = WSSR/ndf : 0.460922

Final set of parameters      Asymptotic Standard Error
=====
a = 5.48214e-09              +/- 4.65e-10 (8.483%)
b = 1.51541e-07              +/- 1.837e-06 (1212%)
c = -0.000501859            +/- 0.002077 (413.9%)
d = 0.154971                 +/- 0.636 (410.4%)

correlation matrix of the fit parameters:
      a      b      c      d
a      1.000
b     -0.987  1.000
c      0.926 -0.973  1.000
d     -0.719  0.795 -0.898  1.000

gnuplot> 
```

Y la gráfica resultante superponiendo ambos datos (función  $f(x)$  y datos de entrada obtenidos):





## 2.4.-Hanoi

Para *Hanoi* se ha utilizado la función:

$$f(x) = a2^x + b$$

Los resultados del ajuste son:

```
reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos

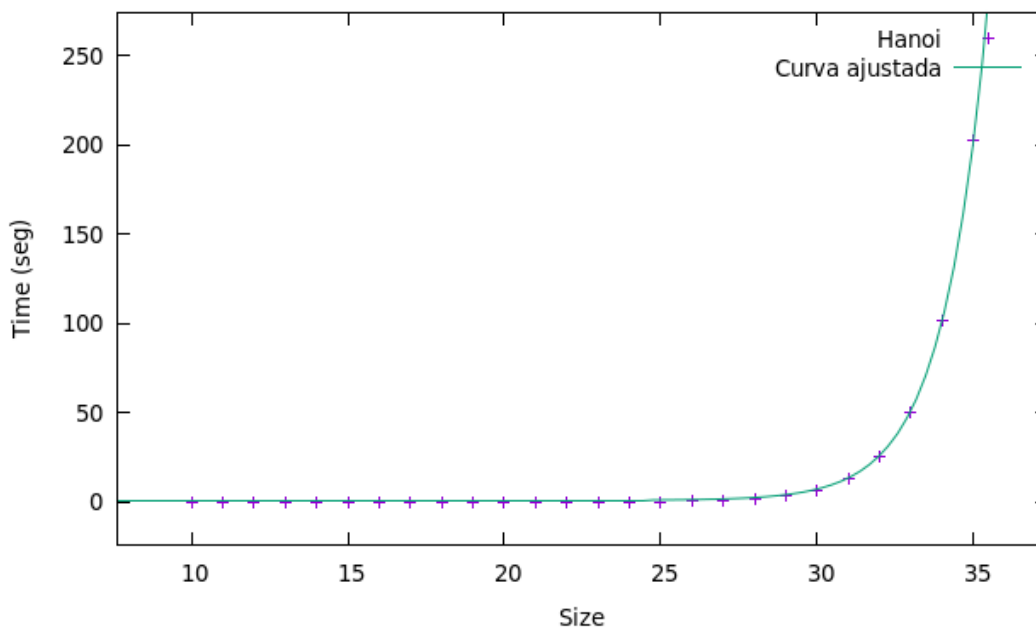
After 5 iterations the fit converged.
final sum of squares of residuals : 22.5514
rel. change during last iteration : -1.51647e-10

degrees of freedom (FIT_NDF) : 24
rms of residuals (FIT_STDFIT) = sqrt(WSSR/ndf) : 0.969352
variance of residuals (reduced chisquare) = WSSR/ndf : 0.939644

Final set of parameters      Asymptotic Standard Error
=====
a      = 5.85145e-09      +/- 2.598e-11 (0.4439%)
b      = 1                +/- 0.2021 (20.21%)

correlation matrix of the fit parameters:
      a      b
a      1.000
b     -0.340 1.000
gnuplot> 
```

Y la gráfica resultante superponiendo ambos datos (función  $f(x)$  y datos de entrada obtenidos):



## 2.5.-Conclusiones

En todos los casos expuestos anteriormente podemos ver que cada una de las funciones propuestas se ajustan muy bien para cada situación.

Este resultado obtenido es bastante lógico puesto que para llevar a cabo el ajuste se han empleado funciones que corresponden al orden de eficiencia de cada uno de los algoritmos.

Como conclusión podemos ver en cada gráfica que los datos de cada algoritmo siguen las trayectorias(curva) adecuadas y que se corresponden bien al orden de eficiencia de los mismos.

## 3.-Curvas mal ajustadas

Podemos apreciar sin embargo, que los datos no se ajustan adecuadamente cuando utilizamos funciones con un orden de eficiencia diferente al del algoritmo analizado. Para comprobar y mostrar esta situación, se realizarán las mismas órdenes usadas en el apartado anterior pero cambiando las funciones declaradas para cada ajuste. Se sustituirán las funciones a analizar de forma que:

Algoritmo	$f(x)$ [Original]	$f(x)$ [Cambiado]
Bubblesort	$f(x) = ax^2 + bx + c$	$f(x) = a * \log(x + b)$
Quicksort	$f(x) = a * x \log(b * x) + c$	$f(x) = ax^3 + bx^2 + cx + d$
Floyd	$f(x) = ax^3 + bx^2 + cx + d$	$f(x) = ax^2 + bx + c$
Hanoi	$f(x) = a2^x + b$	$f(x) = a * x \log(b * x) + c$

En la mayoría de los casos podremos ver como, claramente en la representación gráfica al superponer los datos y las funciones, la trayectoria de estos no se asemejarán demasiado.

En otros casos sin embargo, aunque el orden del algoritmo sea diferente, las trayectorias tendrán cierta similitud, aunque se presentarán irregularidades mucho mas notables que en los ajustes con el orden correcto.

Como datos informativos se incluirán las imágenes de los ajustes diferentes realizados para este apartado.

### 3.1.-Bubblesort

```

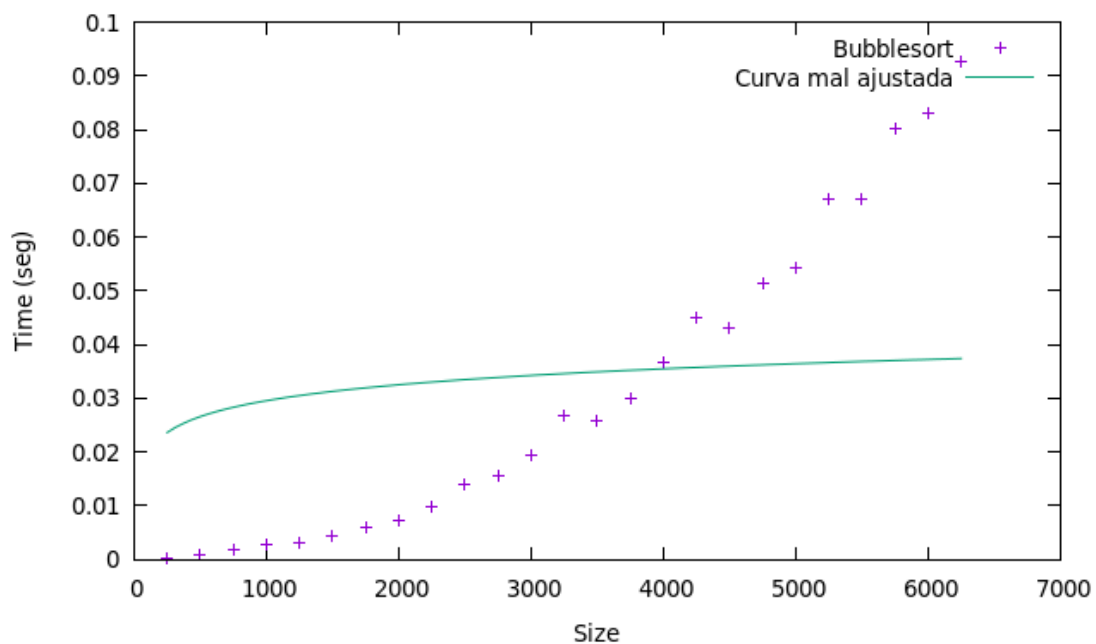
reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos
  2 1.6595954866e-02 -3.57e+06 5.58e-02 4.284522e-03 9.999054e-01
  3 1.6595930356e-02 -1.48e-01 5.58e-03 4.280618e-03 9.995416e-01
iter   chisq      delta/lim  lambda   a           b
After 3 iterations the fit converged.
final sum of squares of residuals : 0.0165959
rel. change during last iteration : -1.47687e-06

degrees of freedom    (FIT_NDF)                : 23
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf) : 0.0268619
variance of residuals (reduced chisquare) = WSSR/ndf : 0.000721562

Final set of parameters      Asymptotic Standard Error
=====
a          = 0.00428062      +/- 0.0008013    (18.72%)
b          = 0.999542        +/- 1460          (1.461e+05%)

correlation matrix of the fit parameters:
      a      b
a      1.000
b     -0.526 1.000
gnuplot>

```



## 3.2.-Quicksort

```

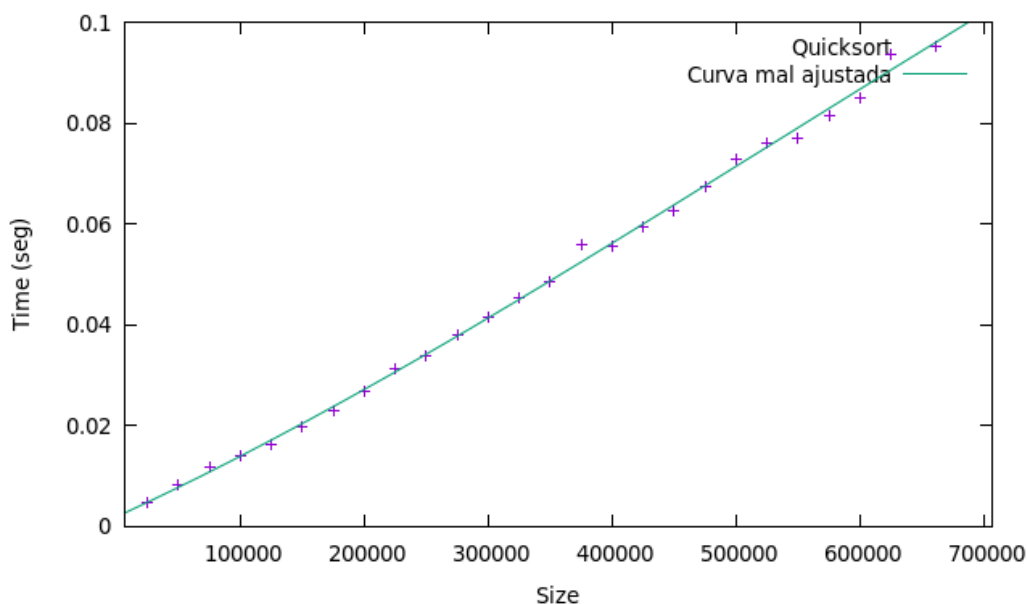
reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos
After 21 iterations the fit converged.
final sum of squares of residuals : 4.01544e-05
rel. change during last iteration : -1.67118e-12

degrees of freedom (FIT_NDF) : 21
rms of residuals (FIT_STDFIT) = sqrt(WSSR/ndf) : 0.00138279
variance of residuals (reduced chisquare) = WSSR/ndf : 1.91212e-06

Final set of parameters      Asymptotic Standard Error
=====
a = -4.73877e-20             +/- 6.062e-20 (127.9%)
b = 7.56435e-14              +/- 5.987e-14 (79.15%)
c = 1.13003e-07              +/- 1.692e-08 (14.97%)
d = 0.00192107               +/- 0.001295 (67.43%)

correlation matrix of the fit parameters:
      a      b      c      d
a      1.000
b     -0.987  1.000
c      0.926 -0.973  1.000
d     -0.719  0.795 -0.898  1.000
gnuplot>

```



Este es un caso en el cual podemos ver como la trayectoria es similar en ambos ajustes, no obstante, al aumentar la zona de visión del gráfico podemos ver como la orientación se desvía considerablemente y cada uno de ellos sigue su propia trayectoria.

### 3.3.-Floyd

```

reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos

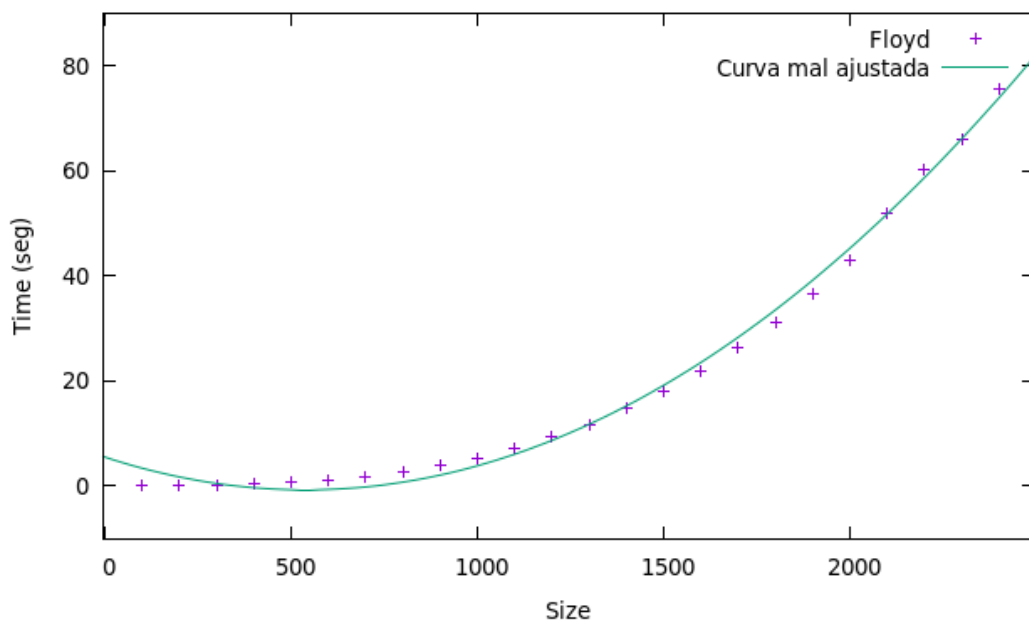
After 10 iterations the fit converged.
final sum of squares of residuals : 73.7324
rel. change during last iteration : -4.19778e-13

degrees of freedom (FIT_NDF) : 22
rms of residuals (FIT_STDFIT) = sqrt(WSSR/ndf) : 1.8307
variance of residuals (reduced chisquare) = WSSR/ndf : 3.35147

Final set of parameters      Asymptotic Standard Error
=====
a = 2.15319e-05             +/- 7.891e-07 (3.665%)
b = -0.023176              +/- 0.002114 (9.12%)
c = 5.54282                +/- 1.193 (21.52%)

correlation matrix of the fit parameters:
      a      b      c
a      1.000
b     -0.971  1.000
c      0.774 -0.884  1.000
gnuplot>

```



Este es un caso mas claro en el cual podemos observar que aunque ambas curvas tienen una trayectoria similar, en realidad, si observamos bien, apreciamos que al principio ambas están desfasadas y en el momento en el que parecen unirse, se presentan ciertas imperfecciones.

### 3.4.-Hanoi

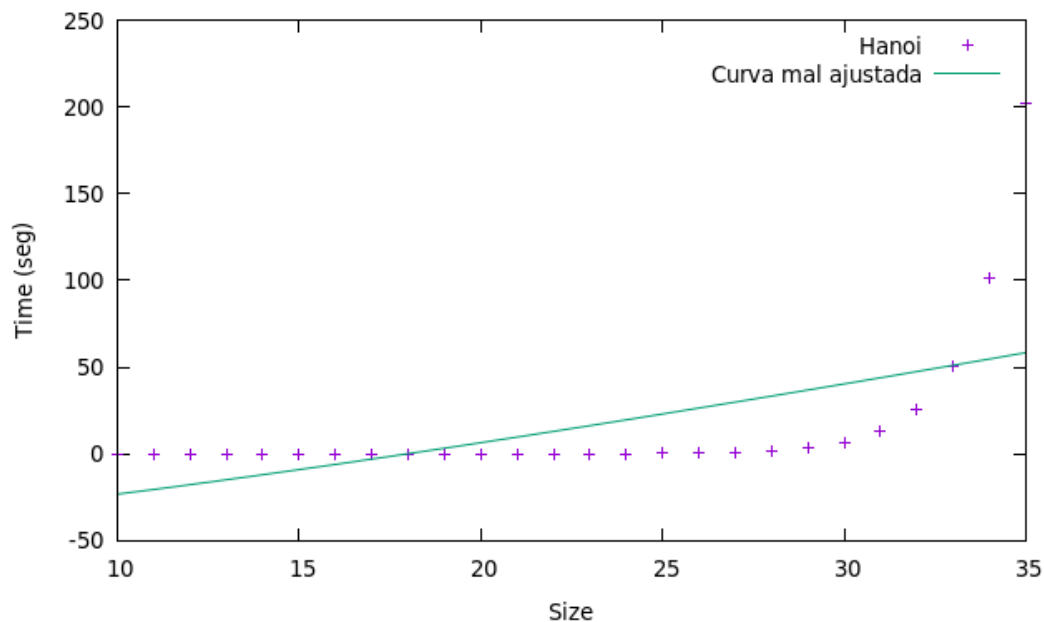
```
reko98@R98: ~/repos/Algoritmica/P1-Eficiencia/Codigos

After 25 iterations the fit converged.
final sum of squares of residuals : 32471.6
rel. change during last iteration : -9.70546e-06

degrees of freedom (FIT_NDF) : 23
rms of residuals (FIT_STDFIT) = sqrt(WSSR/ndf) : 37.574
variance of residuals (reduced chisquare) = WSSR/ndf : 1411.81

Final set of parameters      Asymptotic Standard Error
=====
a = 0.803922                +/- 0.6406 (79.68%)
b = -0.731919              +/- 1536 (2.098e+05%)
c = -41.1125               +/- 1324 (3221%)

correlation matrix of the fit parameters:
      a      b      c
a      1.000
b      0.927  1.000
c     -0.931 -1.000  1.000
gnuplot> 
```



En este caso, podemos apreciar que claramente las trayectorias no coinciden en ningún momento.