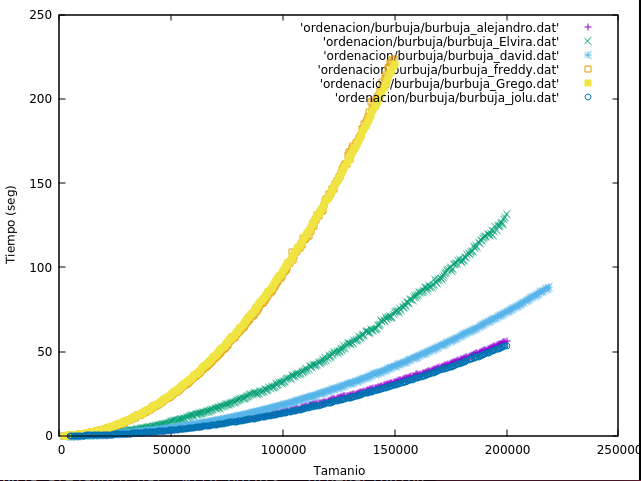
**1. Calcule la eficiencia empírica, siguiendo las indicaciones de la sección 3. Defina adecuadamente los tamaños de entrada de forma tal que se generen al menos 25 datos. Incluya en la memoria tablas diferentes para los algoritmos de distinto orden de eficiencia (una con los algoritmos de orden O(n2), otra con los O(n log n), otra con O(n3) y otra con O(2n)).**

**Burbuja:**

Hemos realizado mediciones de valores para el algoritmo de burbuja empezando en 2000 valores y terminando en ordenar vectores de 200000 valores, incrementando en cada medición el número de elementos en el vector de 500 en 500.

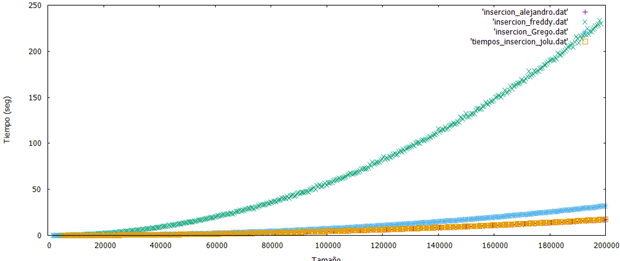
Hemos realizado estas mediciones en equipos con hardware diferente y con opciones distintas de compilación, y este ha sido el resultado:



Como podemos ver varían las gráficas dependiendo del hardware y de las opciones de compilación. Las de Grego y Fredi, están en equipos con similares características tanto hardware y con el mismo sistema operativo y sin ninguna optimización de compilación, las de Jolu, David y Alejandro están con optimización –O2 y la de Elvira está en otro hardware con otro sistema operativo diferente, esas son las diferencias que podemos ver en las ejecuciones.

Sólo añadiremos una parte de la tabla que se generó al tomar los valores para realizar la representación de la gráfica porque se tomaron muchísimas muestras y habría varias páginas con valores de tiempo y tamaño para cada uno de los algoritmos.(si se quiere ver los valores completos que se han tomado, están en la carpeta adjunta a este documento.)

|  |  |
| --- | --- |
| TAMAÑOS | TIEMPOS(Seg) |
| 2000 | 0.015625 |
| 2500 | 0.015625 |
| 3000 | 0.03125 |
| 3500 | 0.015625 |
| 4000 | 0.03125 |
| 4500 | 0.046875 |
| 5000 | 0.0625 |
| 5500 | 0.078125 |
| 6000 | 0.09375 |
| 6500 | 0.109375 |
| ………………………. | ………………………. |
| 143000 | 65.7188 |
| 143500 | 66.25 |
| 144000 | 68.6719 |
| 144500 | 68.125 |
| 145000 | 68.5 |
| ………………………. | ………………………. |
| 195500 | 122.25 |
| 196000 | 125.328 |
| 196500 | 124.375 |
| 197000 | 125.406 |
| 197500 | 126.344 |

**Inserción**

Se puede apreciar que los tiempos de ejecución son distintos según el ordenador que los ha ejecutado. Esto se debe a las diferencias de hardware de cada ordenador y también a la optimización a la hora de compilar el código. Los tiempos de ejecución de los archivos ‘insercion\_alejandro.dat’ y ‘tiempos\_insercion\_jolu.dat’ han sido compilados con la opción -O2

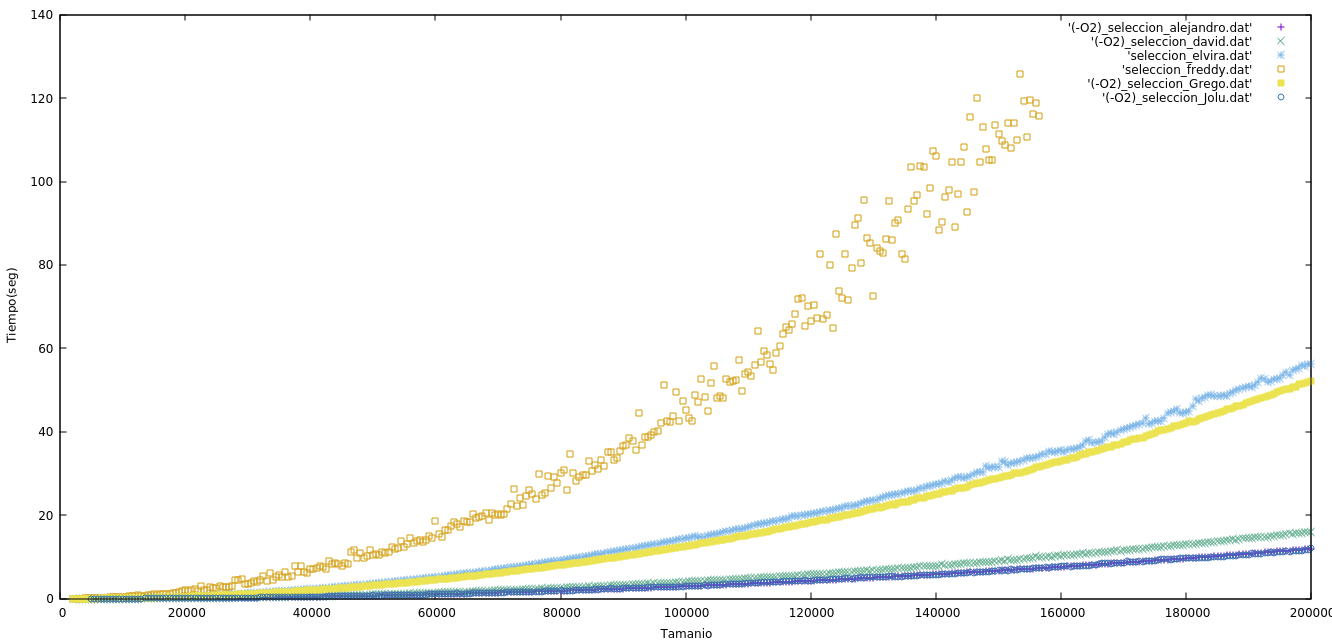
Algunos de los valores que se han obtenido han sido:

|  |  |
| --- | --- |
| Tamaño de entrada | Tiempo (Seg) |
| 6000 | 0.015625 |
| 8000 | 0.03125 |
| 9500 | 0.046875 |
| 107500 | 4.85938 |
| 108000 | 4.90625 |
| 108500 | 4.95312 |
| 199000 | 16.8906 |
| 199500 | 16.7344 |
| 200000 | 17.1719 |

**Selección:**

Hemos realizado mediciones de valores para el algoritmo de burbuja empezando en 2000 valores y terminando en ordenar vectores de 200000 valores, incrementando en cada medición el número de elementos en el vector de 500 en 500.

Hemos realizado estas mediciones en equipos con hardware diferente y con opciones distintas de compilación, y este ha sido el resultado:

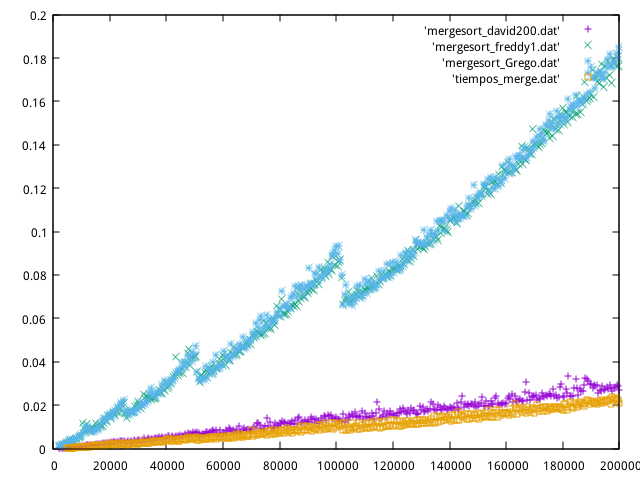


Se puede apreciar que los tiempos de ejecución son distintos según el ordenador que los ha ejecutado. Esto se debe a las diferencias de hardware de cada ordenador y también a la optimización a la hora de compilar el código. Los tiempos de ejecución de los archivos ‘seleccion\_elvira.dat’ y ‘seleccion\_freddy.dat’ han sido compilados sin optimización de compilación.

Algunos de los valores que se han obtenido han sido:

|  |  |
| --- | --- |
| TAMAÑOS | TIEMPOS(Seg) |
| 2000 | 0.00568777 |
| 2500 | 0.00804 |
| 3000 | 0.012405 |
| 3500 | 0.017437 |
| 4000 | 0.022724 |
| 4500 | 0.030412 |
| 5000 | 0.040013 |
| 5500 | 0.048152 |
| 6000 | 0.057192 |
| 6500 | 0.067044 |
| ………………………. | ………………………. |
| 143000 | 26.3186 |
| 143500 | 26.461 |
| 144000 | 26.6193 |
| 144500 | 26.6647 |
| 145000 | 26.8303 |
| ………………………. | ………………………. |
| 195500 | 50.1477 |
| 196000 | 50.2897 |
| 196500 | 50.3756 |
| 197000 | 50.7796 |
| 197500 | 50.8194 |

**Mergesort:**

****

En la gráfica anterior se muestran las ejecuciones de todos los compañeros del algoritmo *mergesort* para el mismo número de entradas en distintas máquinas.

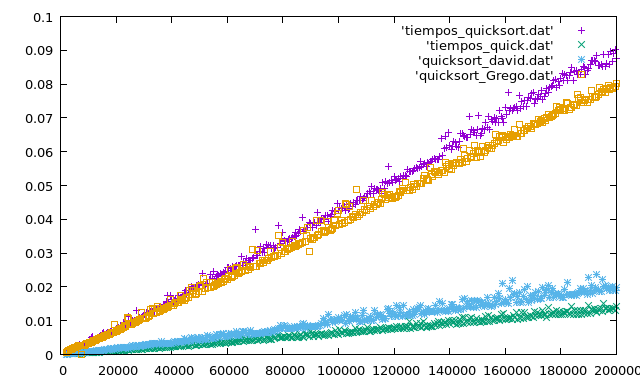
Podemos observar que hay dos nubes de puntos que se disparan y son notablemente superiores a las demás, esto se debe a que el algoritmo ha sido compilado sin usar optimización. Para las otras funciones se puede apreciar que los pequeños saltos que tiene este algoritmo son casi imperceptibles.

En la siguiente tabla podemos apreciar algunos de los valores que se han usado para representar una función de la gráfica anterior.

|  |  |
| --- | --- |
| TAMAÑOS | TIEMPOS(Seg) |
| 2000 | 0.00015162 |
| 2500 | 0.000198092 |
| 3000 | 0.000254718 |
| 3500 | 0.0002798 |
| 4000 | 0.000338138 |
| 4500 | 0.000375973 |
| 5000 | 0.00043228 |
| 5500 | 0.000499076 |
| 6000 | 0.000557404 |
| 6500 | 0.000545124 |
| ………………………. | ………………………. |
| 143000 | 0.01878 |
| 143500 | 0.018007 |
| 144000 | 0.021501 |
| 144500 | 0.024394 |
| 145000 | 0.01901 |
| ………………………. | ………………………. |
| 195500 | 0.028371 |
| 196000 | 0.02726 |
| 196500 | 0.029192 |
| 197000 | 0.028016 |
| 197500 | 0.026633 |

Como se puede observar los tiempos son mucho menores que para los algoritmos de la familia cuadrática.

**Quicksort:**

****

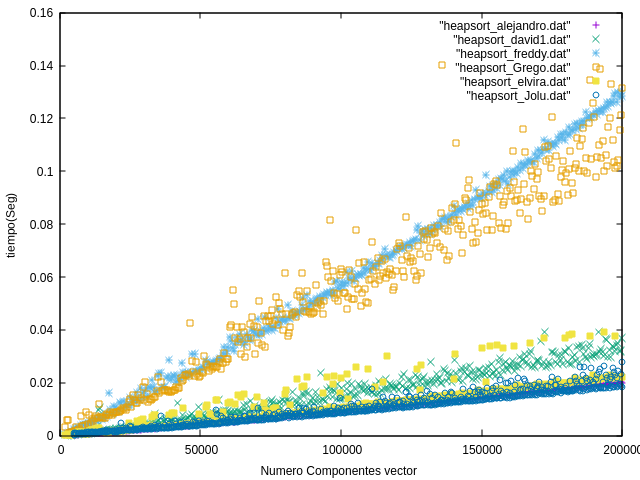
En la gráfica anterior se muestran las ejecuciones de todos los compañeros del algoritmo *quicksort* para el mismo número de entradas en distintas máquinas.

Podemos observar que hay dos nubes de puntos que se disparan y son notablemente superiores a las demás, esto se debe a que el algoritmo ha sido compilado sin usar optimización.

En la siguiente tabla podemos apreciar algunos de los valores que se han usado para representar una función de la gráfica anterior.

|  |  |
| --- | --- |
| TAMAÑOS | TIEMPOS(Seg) |
| 2000 | 0.00061 |
| 2500 | 0.000841 |
| 3000 | 0.00122 |
| 3500 | 0.001155 |
| 4000 | 0.001414 |
| 4500 | 0.001469 |
| 5000 | 0.0017 |
| 5500 | 0.002387 |
| 6000 | 0.002626 |
| 6500 | 0.002188 |
| ………………………. | ………………………. |
| 143000 | 0.061186 |
| 143500 | 0.063106 |
| 144000 | 0.065304 |
| 144500 | 0.065329 |
| 145000 | 0.064914 |
| ………………………. | ………………………. |
| 195500 | 0.088808 |
| 196000 | 0.085373 |
| 196500 | 0.086059 |
| 197000 | 0.08882 |
| 197500 | 0.085897 |

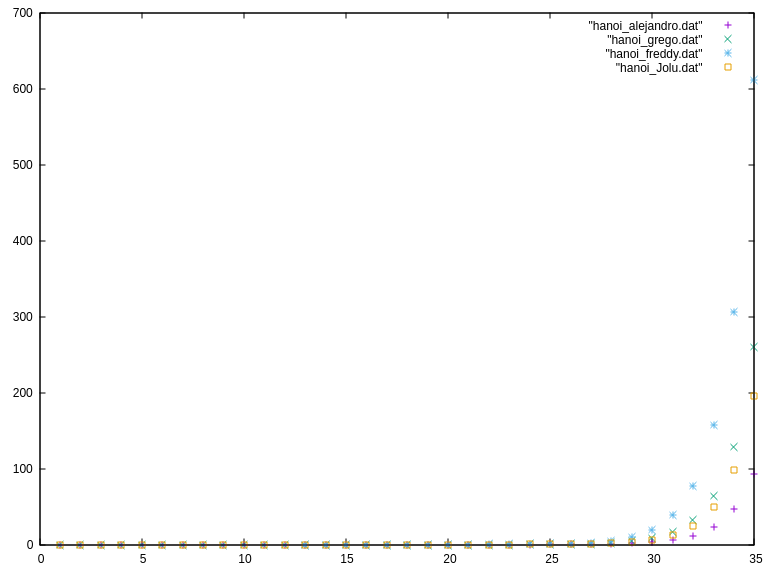
**Heapsort:**

****

En la gráfica anterior se muestran las ejecuciones de todos los compañeros del algoritmo *quicksort* para el mismo número de entradas en distintas máquinas.

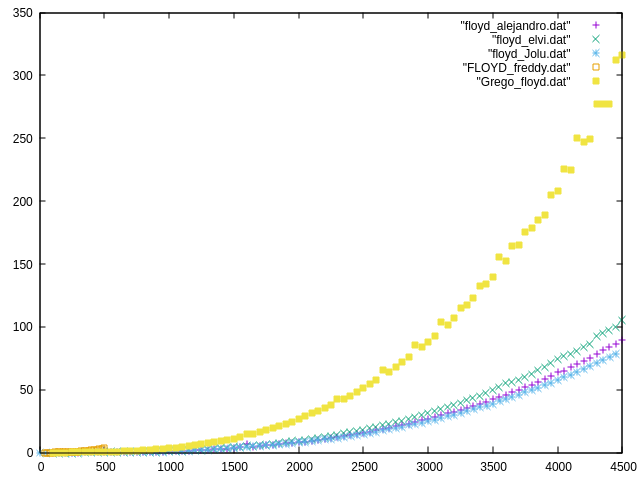
**Los tiempos de ejecución de este algoritmo se pueden consultar en los archivos adjuntos.**

**Hanoi**

En la gráfica anterior se muestran las ejecuciones de todos los compañeros del algoritmo de *Hanoi* para el mismo número de entradas en distintas máquinas.

Podemos observar el comportamiento puramente exponencial del algoritmo de las torres de Hanoi donde apreciamos que para entradas de hasta 30 discos se mantiene en tiempos estables y similares y a partir de ese umbral se dispara de una manera desmesurada.

**Los tiempos de ejecución de este algoritmo se pueden consultar en los archivos adjuntos.**

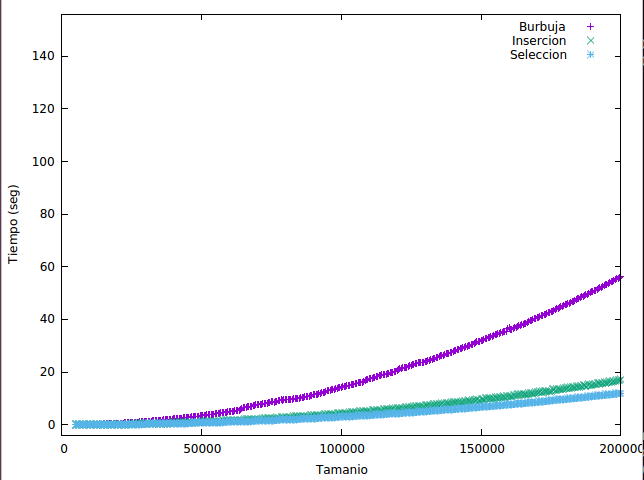
**Floyd:**

En la gráfica anterior se muestran las ejecuciones de todos los compañeros del algoritmo de *Floyd* para el mismo número de entradas en distintas computadoras.

Los tiempos de ejecución de este algoritmo se pueden consultar en los archivos adjuntos ‘floyd\_nombre.dat’

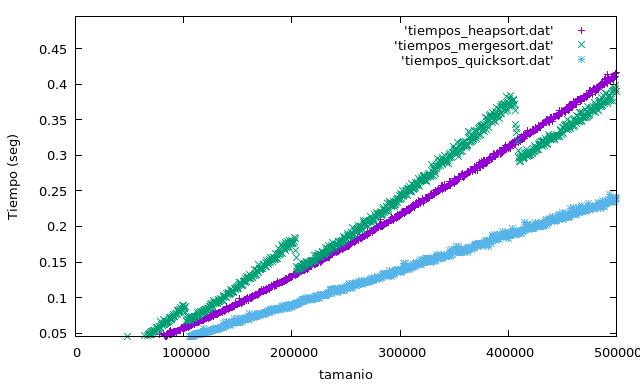
**2. Con cada una de las tablas anteriores, genere un gráfico comparando los tiempos de los algoritmos. Indique claramente el significado de cada serie. Para los algoritmos que realizan la misma tarea (los de ordenación), incluya también una gráfica con todos ellos, para poder apreciar las diferencias en rendimiento de algoritmos con diferente orden de eficiencia.**

**Familia cuadrática:**



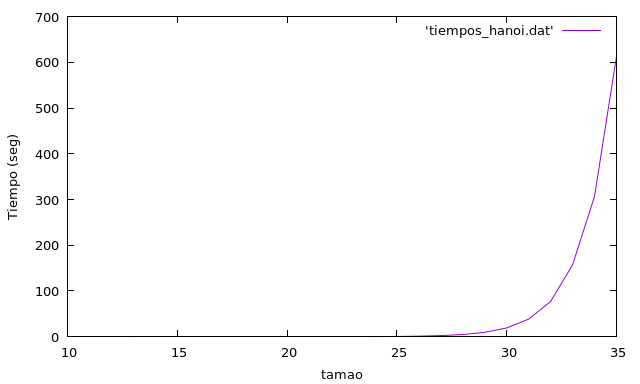
Como podemos ver en la gráfica, de la familia n^2 el que peores tiempos tiene, en el peor de los casos, para tamaños muy grandes de ‘n’ es el algoritmo burbuja, seguido del inserción.

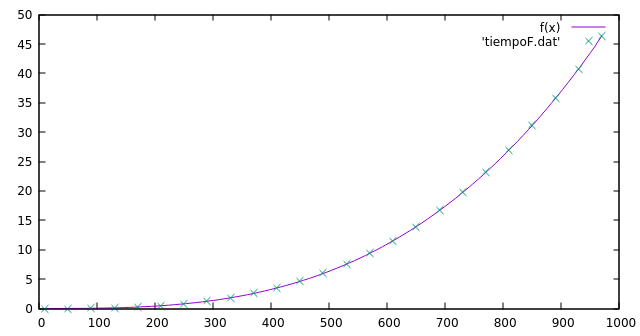
**Familia logarítmica:**



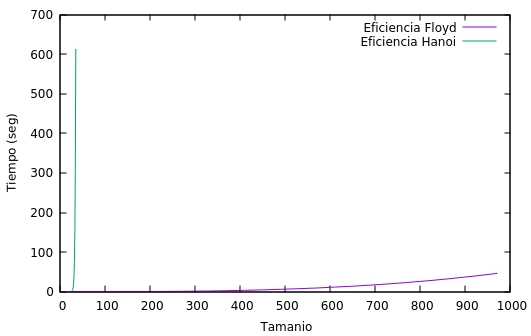
Al comparar los algoritmos de la familia logarítmica podemos observar que para valores pequeños el algoritmo mergesort es peor que los otros dos a comparar y que a partir de cierto punto para unos valores es ligeramente mejor que el heapsort. El mejor de todos es el algoritmo quicksort ya que a pesar de que se base en elegir un pivote e ir ordenando los valores mayores o menores que ese pivote al dividir el trabajo se recorren muchas menos posiciones y el ordenamiento es casi inmediato.

**Familia cúbica y exponencial:**

****



Funciones juntas:



Podemos observar que estos algoritmos poseen unos tiempos de ejecución bastante altos para la cantidad de entradas con las que estamos trabajando, es por lo que problemas como las Torres de Hanoi o encontrar el camino mínimo en un grafo son tareas con un alto tiempo computacional las cuales con las prestaciones que hoy en día existen siguen suponiendo un esfuerzo enorme.

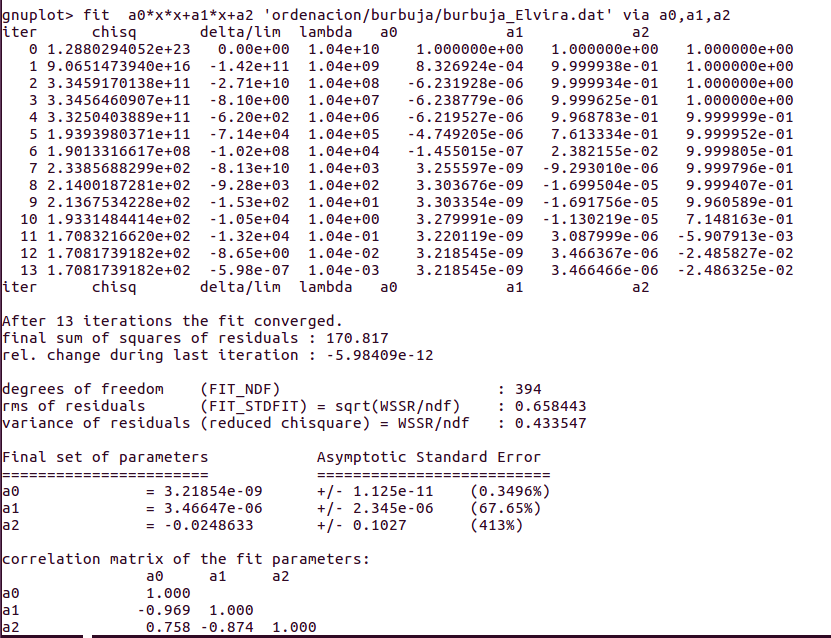
**3. Calcule también la eficiencia híbrida de todos los algoritmos, siguiendo las pautas indicadas en la sección 4. Pruebe también con otros ajustes que no se correspondan con la eficiencia teórica (ajuste lineal, cuadrático, etc) y compruebe la variación en la calidad del ajuste.**

**Burbuja:**

En este caso hemos calculado la eficiencia híbrida del algoritmo burbuja, hemos hecho un ajuste a una función cuadrática y hemos calculado los coeficientes de la recta de regresión que más se adapta entre:

*F(x) = A0\*x^2+A1\*x+A2*

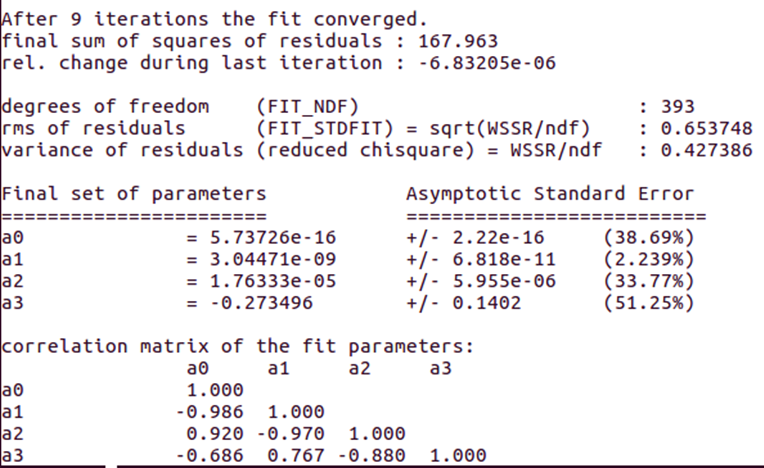
y la función que aparece al hacer la gráfica con los tiempos medidos por nosotros en uno de los ordenadores.



en esta imagen mostramos la ejecución del ajuste realizado con los algoritmos de eficiencia O(n^2) y como vemos un buen ajuste porque el error cuando nos quedamos solo con el coeficiente de x^2 es muy cercano a 0, por lo tanto se confirma que el algoritmo burbuja pertenece a la familia cuadrática.

Si realizamos el mismo ajuste pero con una función cúbica obtenemos estos resultados:

*F(x) = A0\*x^3+A1\*x^2+A2\*x+A3*

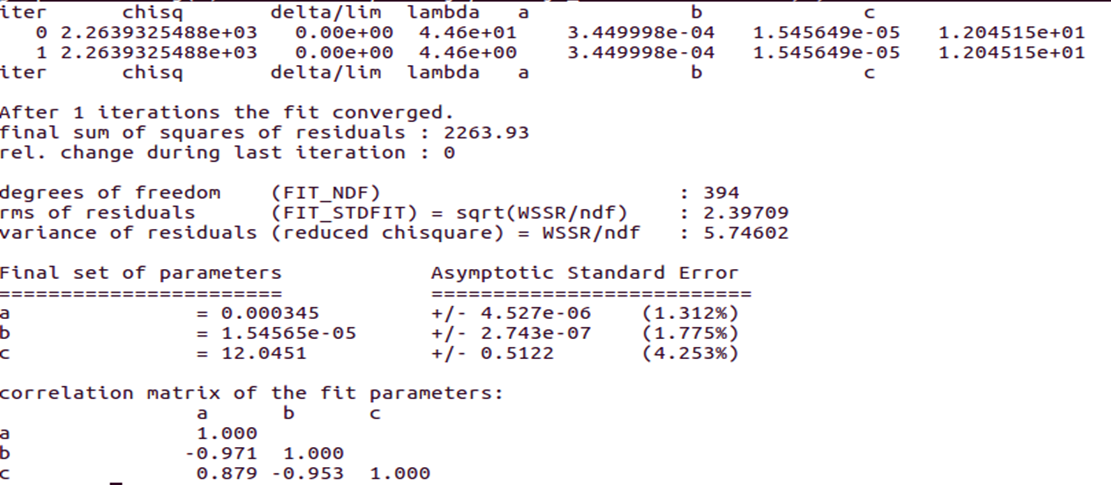


En esa imagen si nos fijamos en el coeficiente de la x^3 si ese es el único que vale 1 y los demás valen 0, tenemos un 38% de error, lo cual indica que se ajusta peor a esta familia que a la familia cuadrática

Por último hemos realizado la comprobación con los algoritmos de eficiencia *O(nlog(n))* ajustando a la función:

*F(x) = a\*x\*(log10(b\*x)/log10(2))+c*

Y los valores obtenidos son los siguientes:



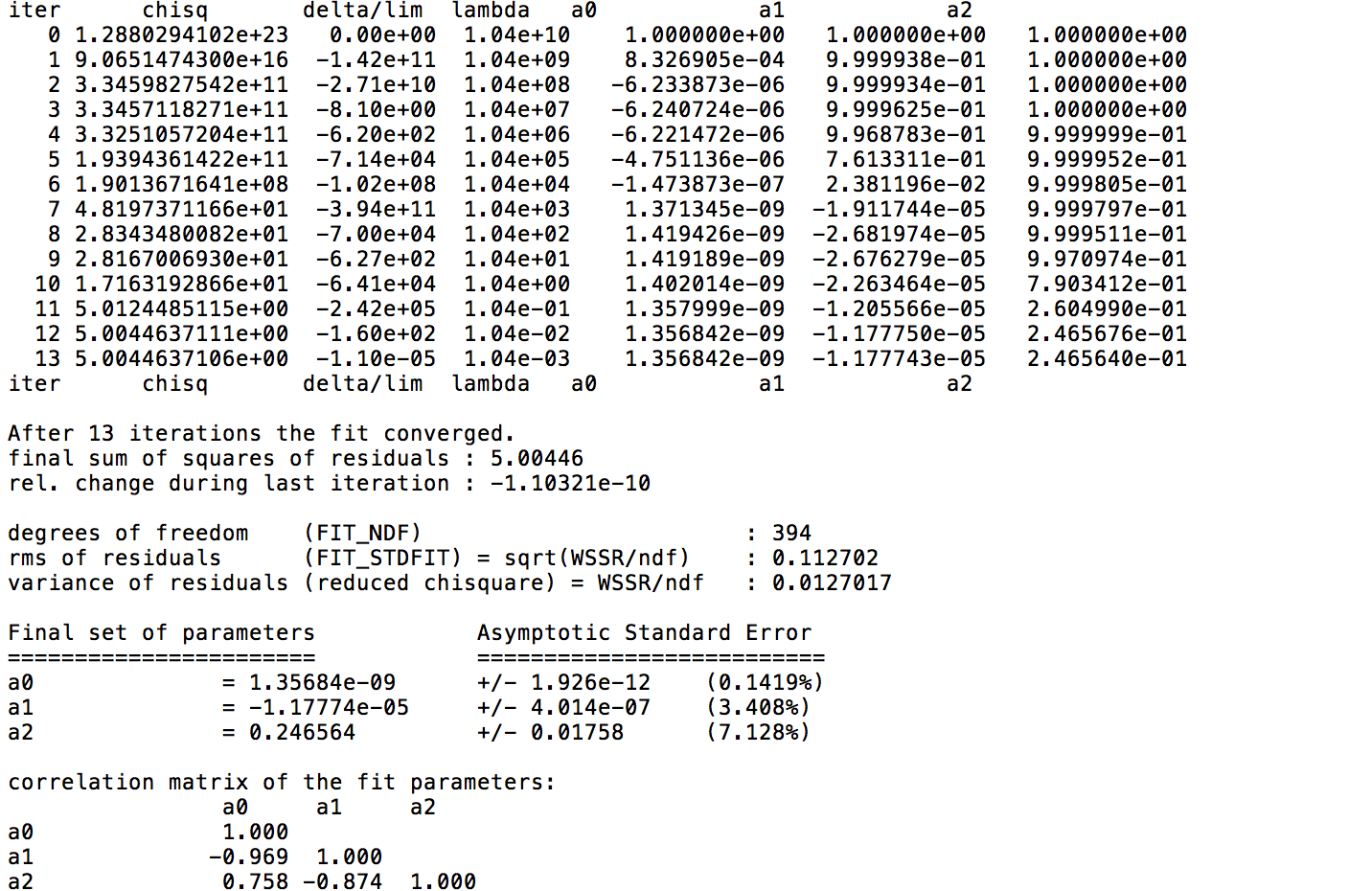
como vemos el error es inmenso en comparación con los n^2 y los n^3, lo cual demuestra que el algoritmo burbuja no es tampoco nlog(n).

**Seleccion:**

En este caso hemos calculado la eficiencia híbrida del algoritmo seleccion, hemos hecho un ajuste a una función cuadrática y hemos calculado los coeficientes de la recta de regresión que más se adapta entre:

*F(x) = A0\*x^2+A1\*x+A2*

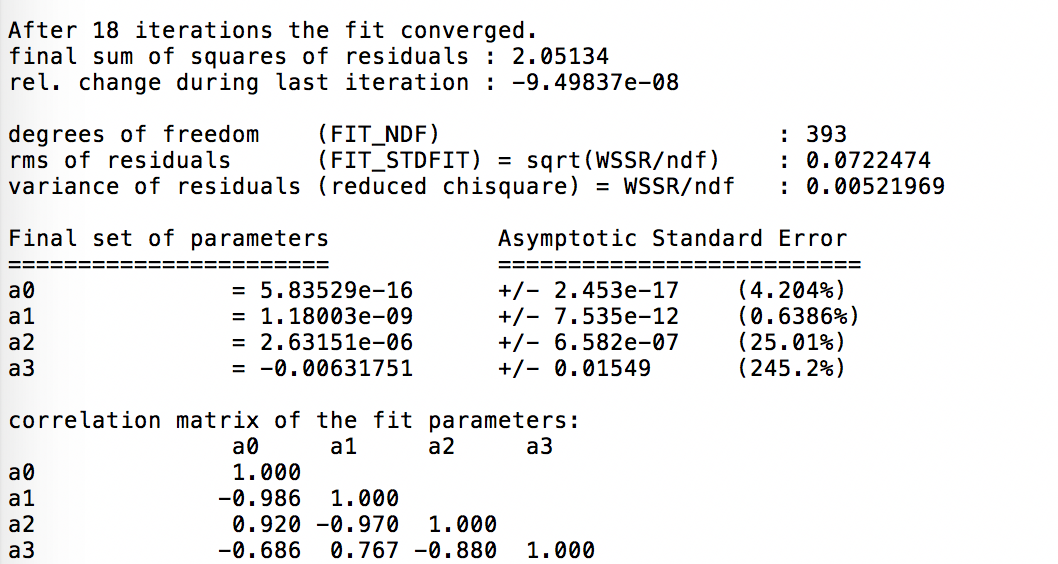
y la función que aparece al hacer la gráfica con los tiempos medidos por nosotros en uno de los ordenadores.



en esta imagen mostramos la ejecución del ajuste realizado con los algoritmos de eficiencia O(n^2) y como vemos un buen ajuste porque el error cuando nos quedamos solo con el coeficiente de x^2 es muy cercano a 0, por lo tanto se confirma que el algoritmo burbuja pertenece a la familia cuadrática.

Si realizamos el mismo ajuste pero con una función cúbica obtenemos estos resultados:

*F(x) = A0\*x^3+A1\*x^2+A2\*x+A3*



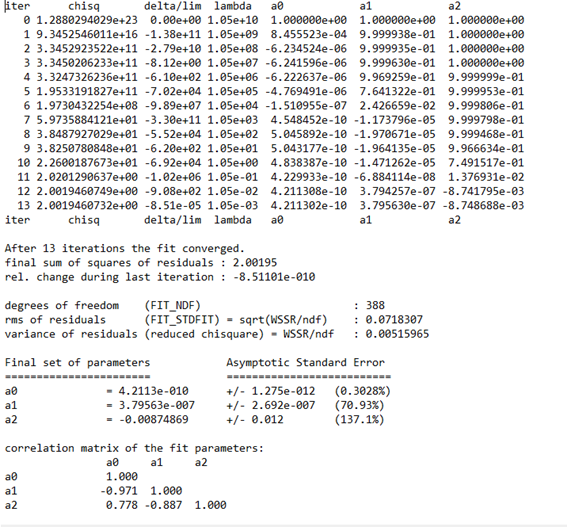
En esa imagen si nos fijamos en el coeficiente de la x^3 si ese es el único que vale 1 y los demás valen 0, tenemos un 4,2% de error y el error sigue aumentando con las de mas variables, lo cual indica que se ajusta peor a esta familia que a la familia cuadrática

**Inserción**

Para este algoritmo la función que se ha usado a ajustar es la siguiente

F(x) = A0\*x^2+A1\*x+A2

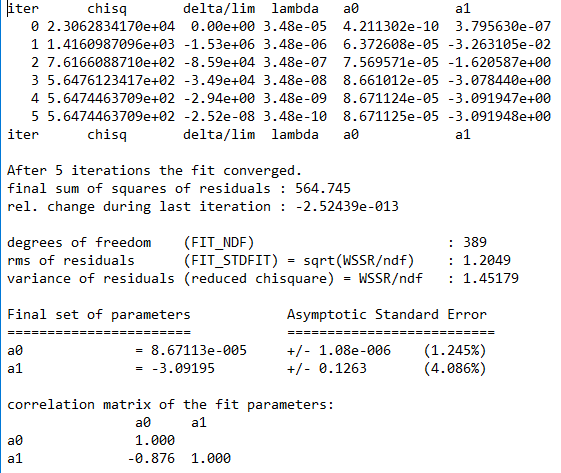
y hemos obtenido los siguientes resultados:



Como podemos observar vemos que el error de x^2 es muy cercano a 0, esto quiere decir que el ajuste realizado es bastante bueno.

Para un ajuste lineal la función a ajustar es F(x) = A0\*x+A1

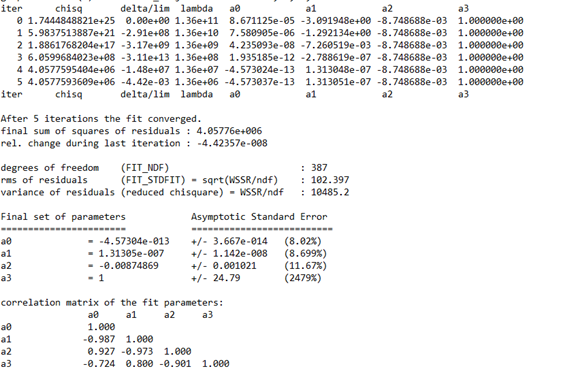
y se obtienen los siguientes resultados:



En este caso el error de x es cercano a 1, por tanto el ajuste no es muy bueno.

Para un ajuste n^3 la función a ajustar es F(x) = A0\*x^3+A1\*x^2+A2\*x+A3

y se obtienen los siguientes resultados:



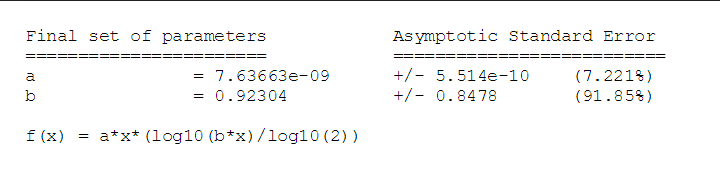
Como vemos el error x^3 es de un 8.02% y deducimos que el ajuste es muy malo.

**Mergesort:**

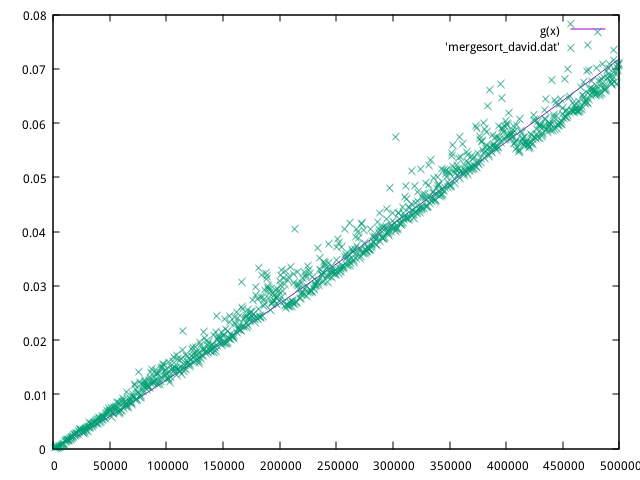
Para este algoritmo la función que se ha usado a ajustar es la siguiente

*F(x) = a\*x\*(log10(b\*x)/log10(2))*

y hemos obtenido los siguientes resultados.

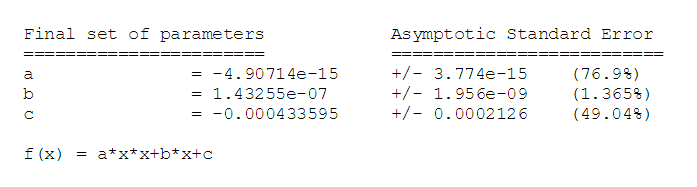


A continuación se muestra una gráfica comparando los tiempos medidos con la función ajustada.

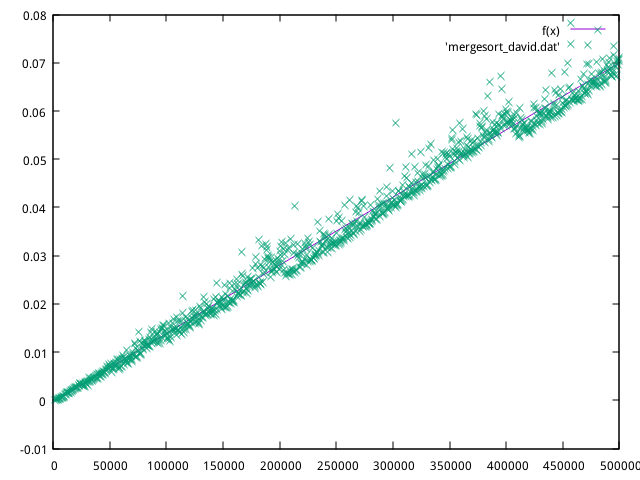


Se puede apreciar que se ajusta perfectamente ya que el algoritmo pertenece a esa familia y para otras familias se ha obtenido lo siguiente.

Para los tiempos anteriores se ha ajustado a la función cuadrática *F(x) = a\*x^2 + b\*x + c*  y obtenemos lo siguiente.



Al comparar en la gráfica esta función ajustada a los tiempos que teníamos se observa lo siguiente.



Los función se ajusta a la nube de puntos a pesar de que no sea su familia aunque podemos ver que tiene un alto índice de error por lo que se demuestra que esa función no es la óptima con la que representar los datos de la nube de puntos.

**Quicksort:**

Para este algoritmo la función que se ha usado a ajustar es la siguiente

*F(x) = f(x)=a\*x\*(log10(x)/log10(2))+b\*x*

degrees of freedom (FIT\_NDF) : 995

rms of residuals (FIT\_STDFIT) = sqrt(WSSR/ndf) : 0.00202018

variance of residuals (reduced chisquare) = WSSR/ndf : 4.08113e-06

Final set of parameters Asymptotic Standard Error

======================= ==========================

a = 2.11356e-08 +/- 4.598e-10 (2.176%)

b = 7.72479e-08 +/- 8.488e-09 (10.99%)

correlation matrix of the fit parameters:

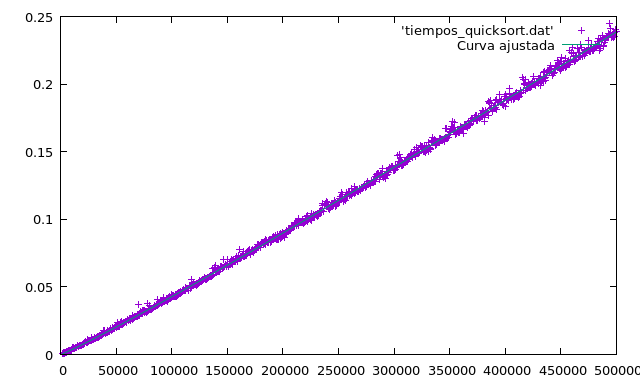
a b

a 1.000

b -1.000 1.000

Como se puede ver el error es muy bajo y muestra lo exacto que es su designación en la familia de los nlogn

A continuación presentamos la gráfica de los tiempos tomados con el ajuste



A continuación las regresiones para O(n) y O(n³)

degrees of freedom (FIT\_NDF) : 993

rms of residuals (FIT\_STDFIT) = sqrt(WSSR/ndf) : 0.00202367

variance of residuals (reduced chisquare) = WSSR/ndf : 4.09524e-06

Final set of parameters Asymptotic Standard Error

======================= ==========================

a = -1.43476e-19 +/- 2.738e-20 (19.08%)

b = 1.86762e-13 +/- 2.09e-14 (11.19%)

c = 4.21195e-07 +/- 4.524e-09 (1.074%)

d = -0.00083145 +/- 0.0002632 (31.65%)

correlation matrix of the fit parameters:

a b c d

a 1.000

b -0.986 1.000

c 0.918 -0.969 1.000

d -0.672 0.754 -0.872 1.000

Como podemos ver el porcentaje de error en la primera y ultima constante es bastante alto y eso nos muestra que no es factible como O(n³).

degrees of freedom (FIT\_NDF) : 995

rms of residuals (FIT\_STDFIT) = sqrt(WSSR/ndf) : 0.00251606

variance of residuals (reduced chisquare) = WSSR/ndf : 6.33057e-06

Final set of parameters Asymptotic Standard Error

======================= ==========================

a = 4.82484e-07 +/- 5.537e-10 (0.1148%)

b = -0.00508738 +/- 0.0001602 (3.149%)

correlation matrix of the fit parameters:

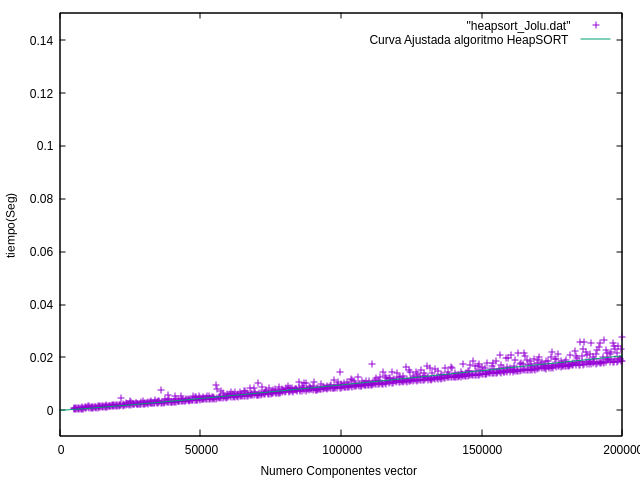
a b

a 1.000

b -0.868 1.000

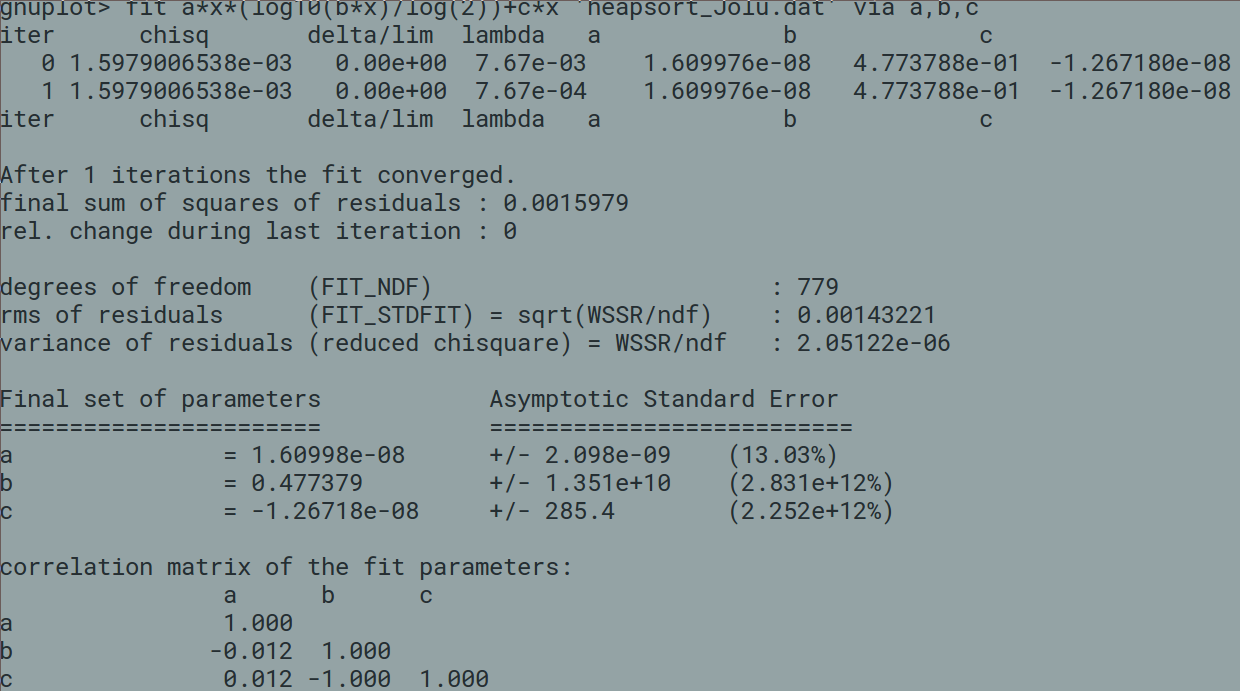
Como podemos ver en el porcentaje de error tan bajo el quicksort es un algoritmo muy cercano a O(n).

**Heapsort:**

****

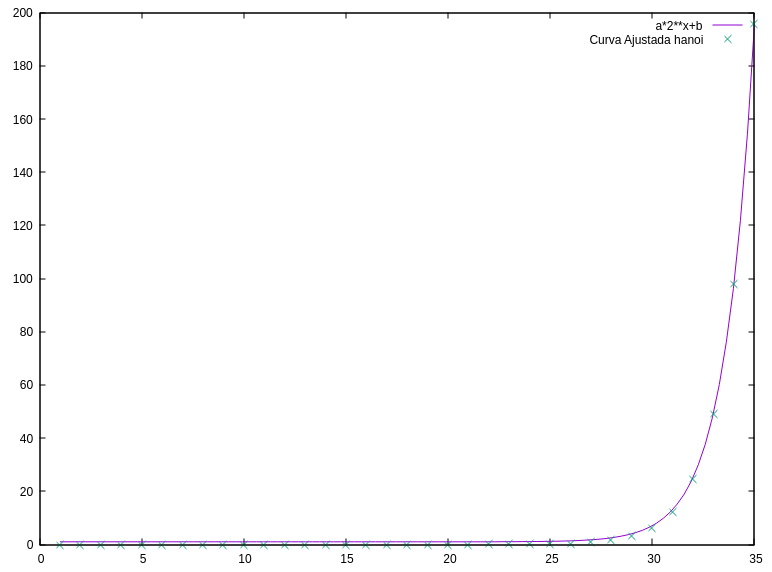
En esta gráfica se puede ver el ajuste del algoritmo de Heapsort según la función a\*x\*log(b\*x)+c\*x.

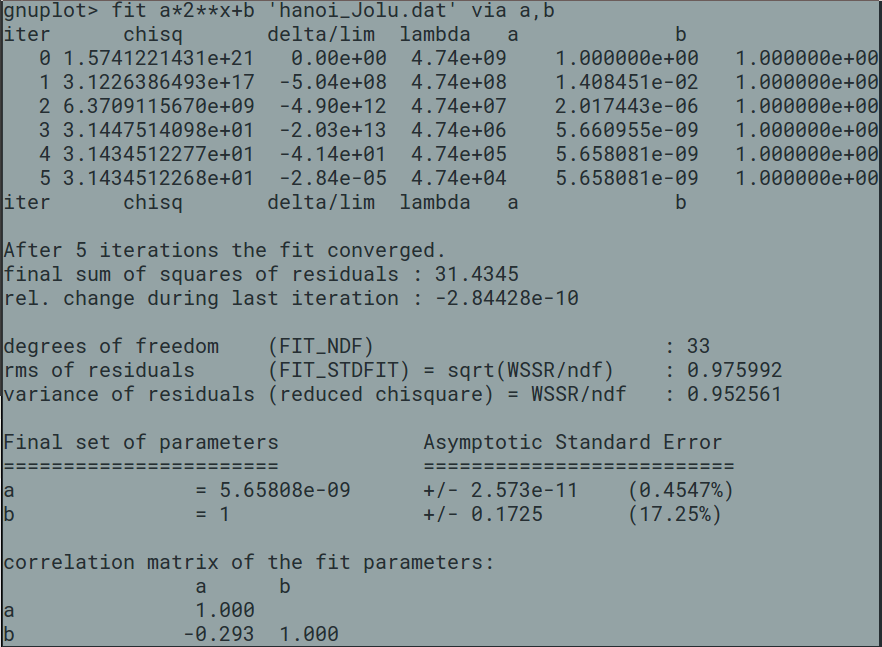
El ajuste es adecuado porque como podemos apreciar, excepto en en coeficiente a que obtenemos un porcentaje de error de 13.03%, en los demás coeficientes apenas llegamos al 0%.



HANOI

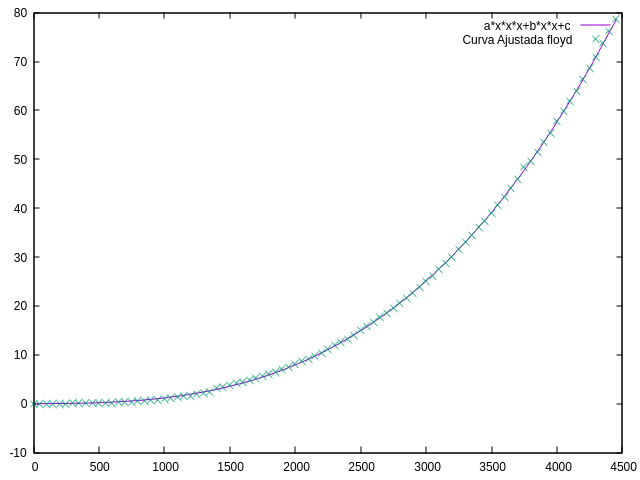
los valores que hemos tomado para representar hanoi han sido de 1 en 1 hasta 35, al ser del orden 2^n, es decir exponencial, se va muy rápido a tiempos de ejecución infinitos, el máximo de valores que le hemos dado ha sido 35.

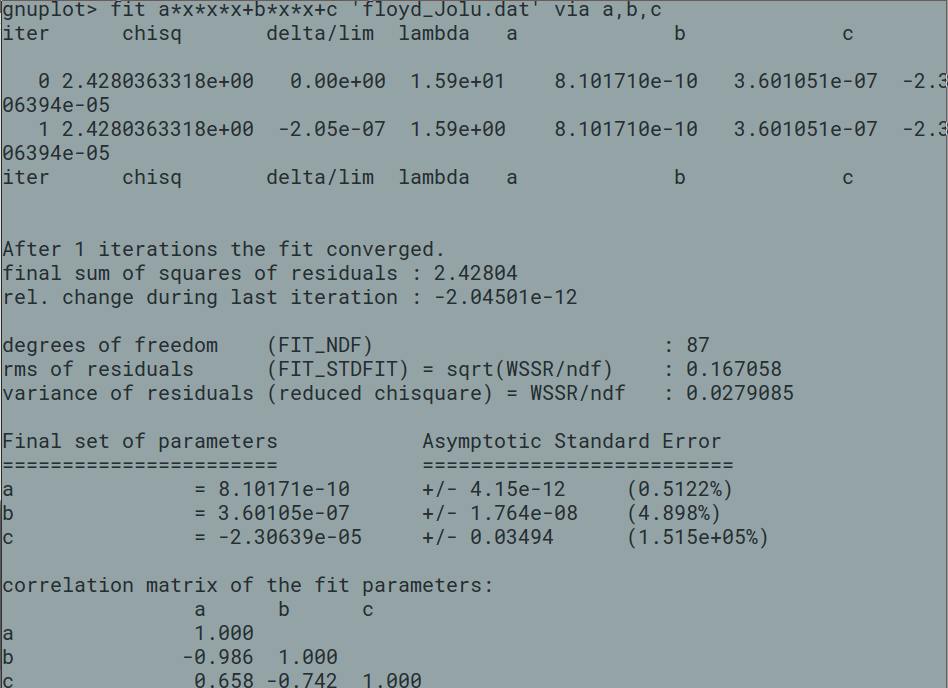
En la siguiente imagen hemos hecho el ajuste al algoritmo de hanoi con una función 2^n y como podemos ver el ajuste es bueno, dado que el error del ajuste al coeficiente a es muy bajito.



FLOYD

En esta imagen representamos los valores que hemos tomado para ejecutar el algoritmo de floyd, como es un algoritmo perteneciente a la familia de los cúbicos, no hemos podido dar una gran cantidad de valores para representarlo en la gráfica dado a que tardaba demasiado.



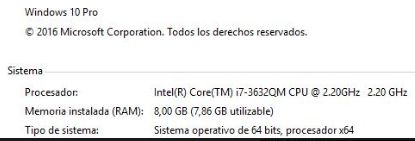


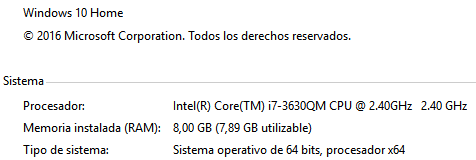
Podemos ver que el error del coeficiente a es próximo a 0 y podemos decir que el ajuste realizado es bueno.

**4. Otro aspecto interesante a analizar mediante este tipo de estudio es la variación de la eficiencia empírica en función de parámetros externos tales como: las opciones de compilación utilizada (con/sin optimización), el ordenador donde se realizan las pruebas, el sistema operativo, etc. Sugiera algún estudio de este tipo, consulte con el profesor de prácticas y llévelo a cabo.**

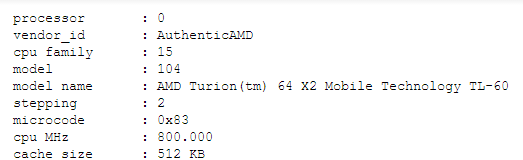
Burbuja: (explicado en el punto nº1 con gráfica y comparativas)

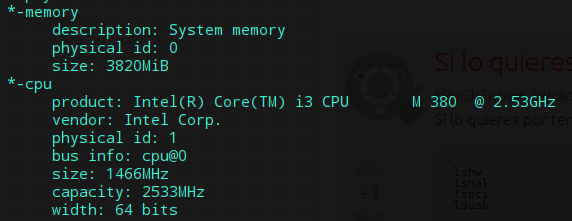
El hardware y los sistemas operativos que hemos usado para realizar esta práctica son:

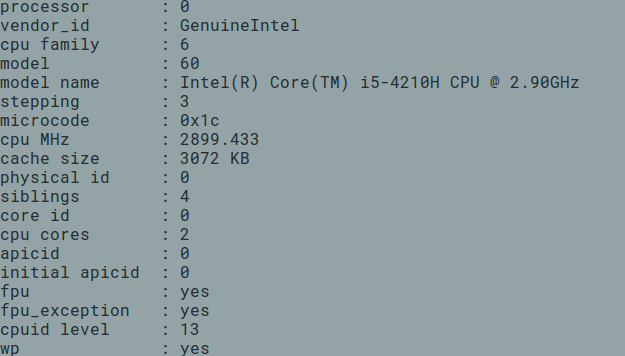




Linux:





****