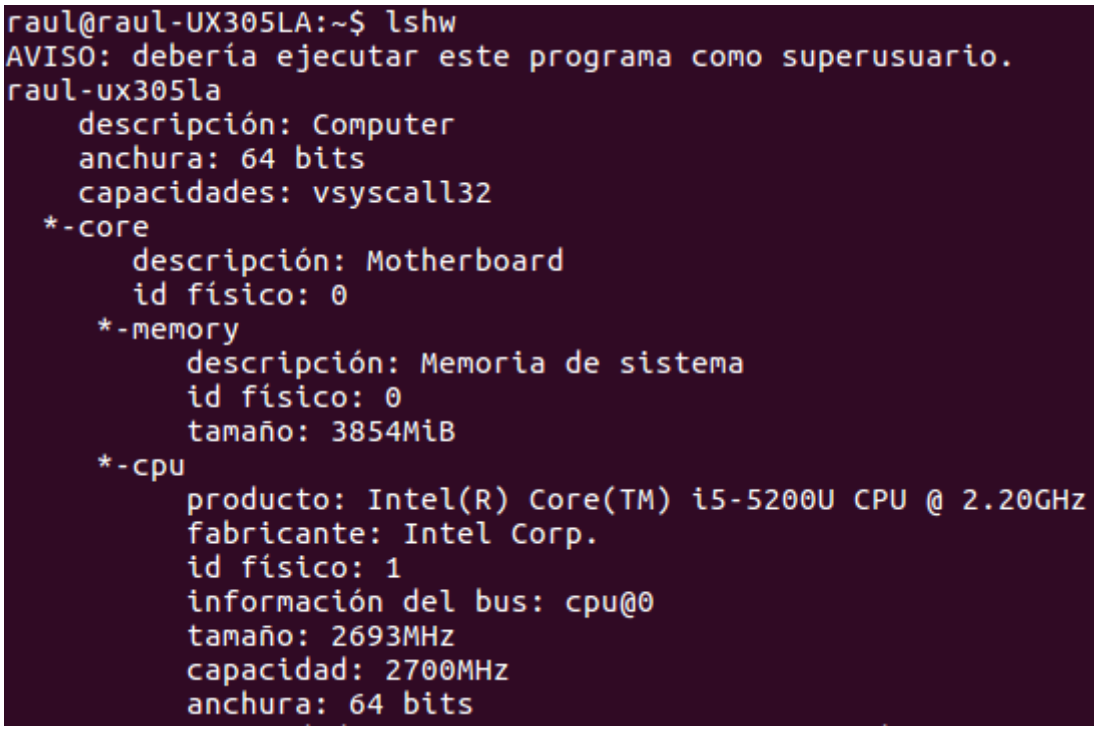
Practica 1 : Eficiencia

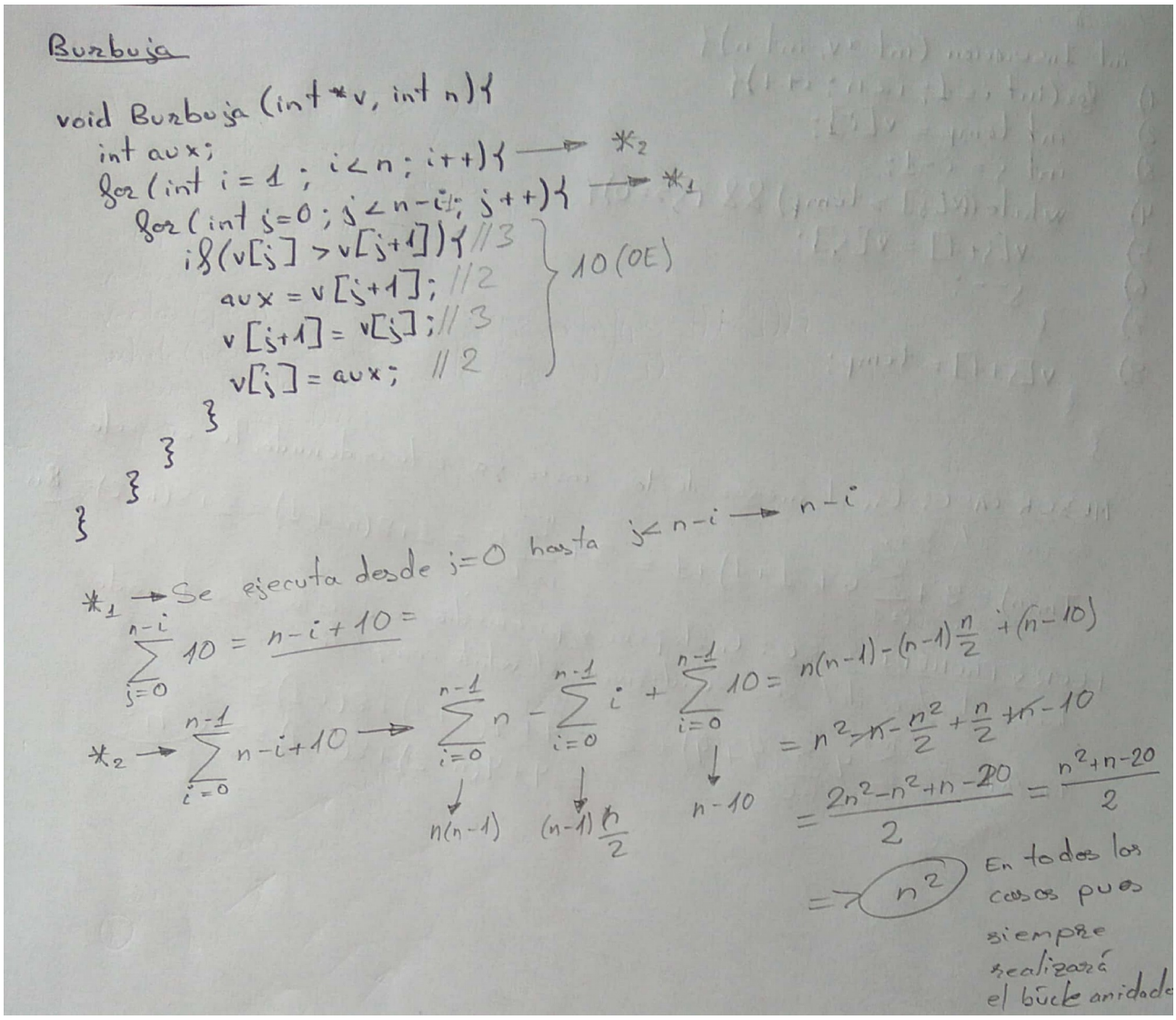
Lo primero que voy a mostrar es el hardware que he usado para realizar la practica en mi caso dejo una captura con el procesador y la memoria RAM. Además, destacar que uso Ubuntu 16.04 y que en ningún momento en toda la practica he compilado mis programas con ninguna optimización ya sea O1 o por ejemplo O2.



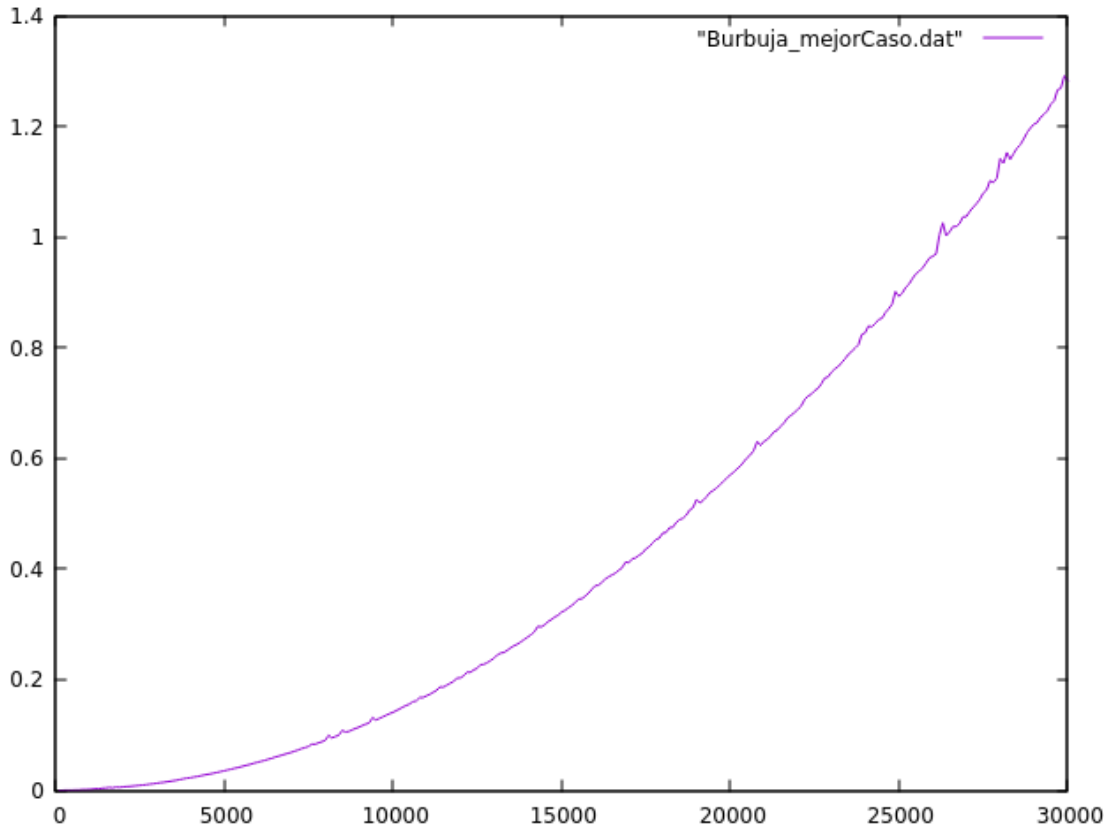
Adjunto todos los códigos usado en mi practica cada uno en sus respectivas carpetas, así como todas las capturas realizadas durante todo el proceso. He usado como compilador G++ desde el terminal de Ubuntu 16.04.

Algoritmos de Ordenación Básicos

El primer algoritmo que he estudiado es el algoritmo de ordenación conocido como ***Burbuja*** este algoritmo funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Dejo el proceso que he realizado para calcular la eficiencia teórica de este algoritmo en una captura, al final he deducido que este algoritmo tiene una eficiencia de N^2 en toda sus situaciones, ya que siempre a menos tiene que realizar el bucle anidado for, luego como siempre va a tener que realizarlo la eficiencia siempre será la misma.



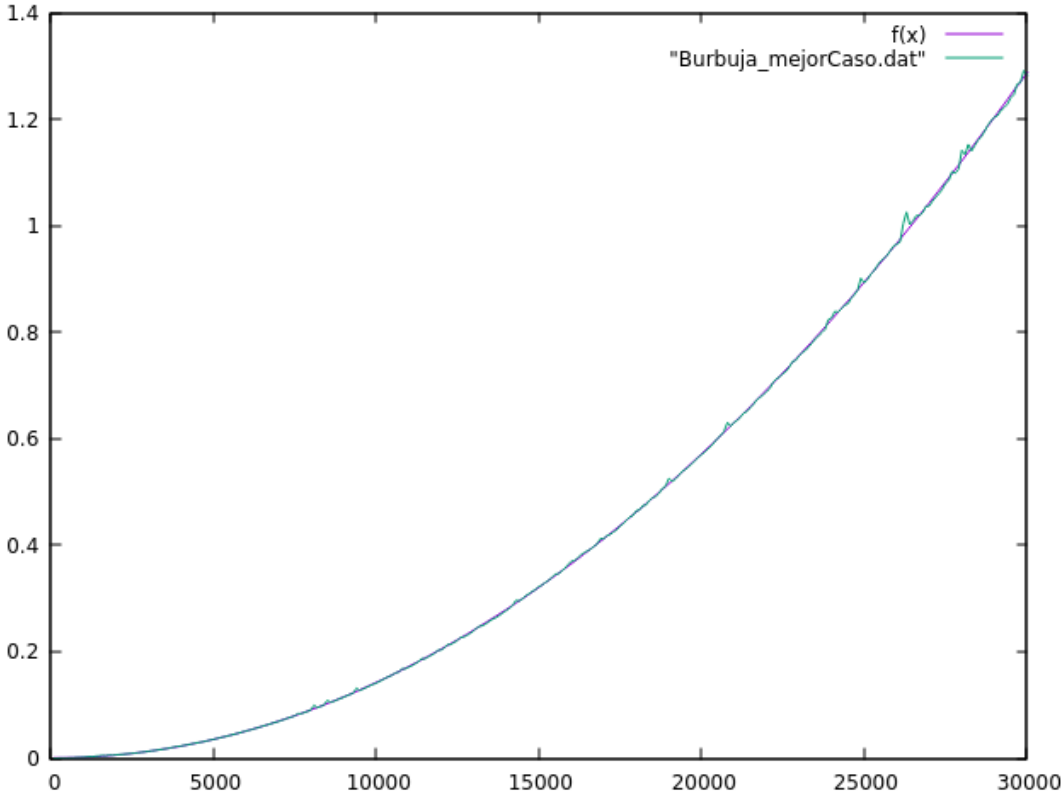
En cuanto a la eficiencia empírica, he ejecutado este algoritmo con un conjunto de datos y he sacado las gráficas correspondientes, el código .cpp que he utilizado lo adjunto en su carpeta correspondiente.



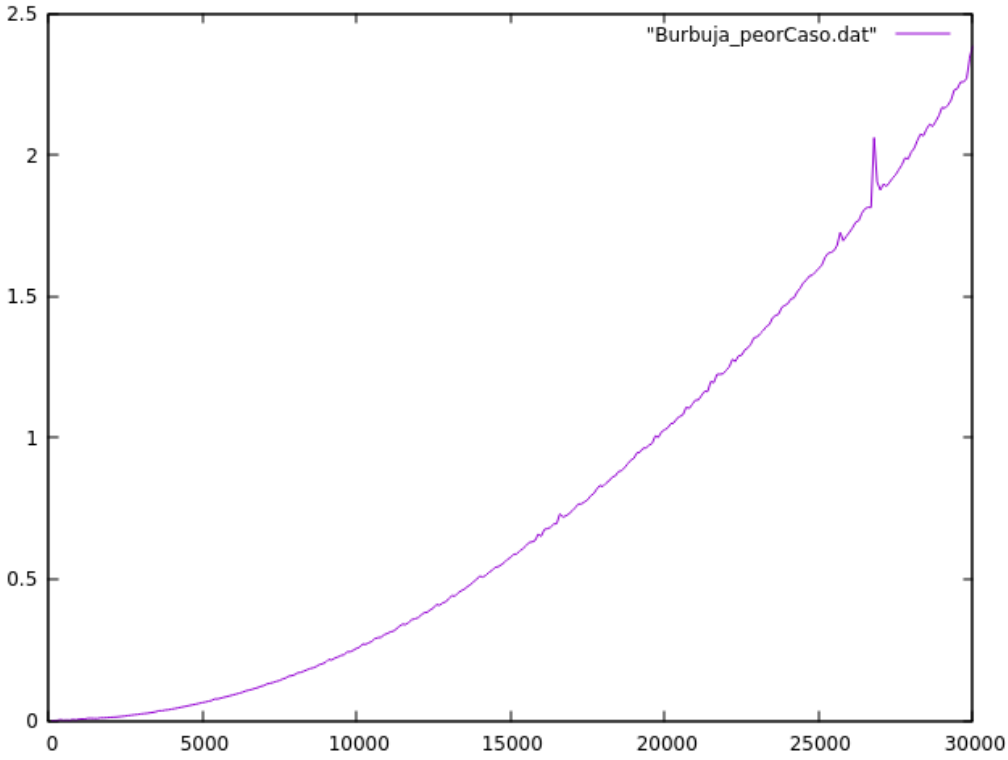
Como observamos tiene una tendencia clara a ser la eficiencia que hemos calculado teóricamente con anterioridad, voy a dejar el procedimiento de ajuste a la curva x^2.

```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "Burbuja_mejorCaso.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 1.0705606544e+03   0.00e+00  1.08e+00    3.915747e-09   9.238081e-07   1.000000e+00
   1 1.2339915678e-02  -8.68e+09  1.08e-01    1.395754e-09   9.134146e-07   2.967878e-05
   2 6.2697183941e-03  -9.68e+04  1.08e-02    1.406508e-09   8.048106e-07  -5.472427e-03
   3 4.7007156052e-03  -3.34e+04  1.08e-03    1.436701e-09  -1.641176e-07   6.703755e-06
   4 4.6881228464e-03  -2.69e+02  1.08e-04    1.439656e-09  -2.589531e-07   5.430430e-04
   5 4.6881228343e-03  -2.57e-04  1.08e-05    1.439659e-09  -2.590460e-07   5.435684e-04
iter      chisq       delta/lim  lambda   a             b             c

After 5 iterations the fit converged.
final sum of squares of residuals : 0.00468812
rel. change during last iteration : -2.57308e-09

degrees of freedom    (FIT_NDF)                        : 297
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00397302
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.57849e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 1.43966e-09      +/- 3.42e-12     (0.2375%)
b               = -2.59046e-07     +/- 1.063e-07    (41.03%)
c               = 0.000543568      +/- 0.0006928    (127.4%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.968  1.000
c               0.748 -0.868  1.000
gnuplot> plot f(x), "Burbuja_mejorCaso.dat"
gnuplot> plot f(x), "Burbuja_mejorCaso.dat" w l
```

Finalmente, los datos recopilados y la curva calculada y ajustada a los datos sería la siguiente:



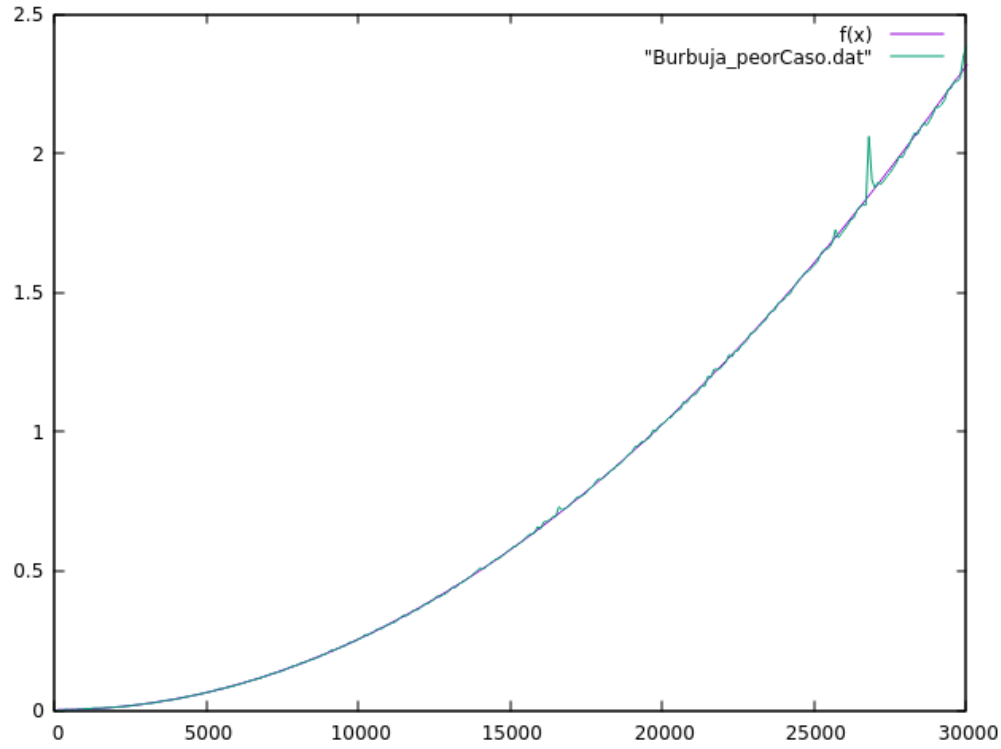
Luego deducimos finalmente que concuerda con el calculo teórico que calculamos antes. Voy a poner todos los cálculos realizados de el peor y caso promedio. En el peor caso lo que he realizado es insertar un vector justo al revés, en el mejor caso es el vector ordenado y en el caso promedio lo que he realizado ha sido ejecutar el algoritmo una series de veces, sumar y dividir entre el numero de veces ejecutado. Adjunto capturas de peor y mejor caso en burbuja, todos los cálculos realizados.



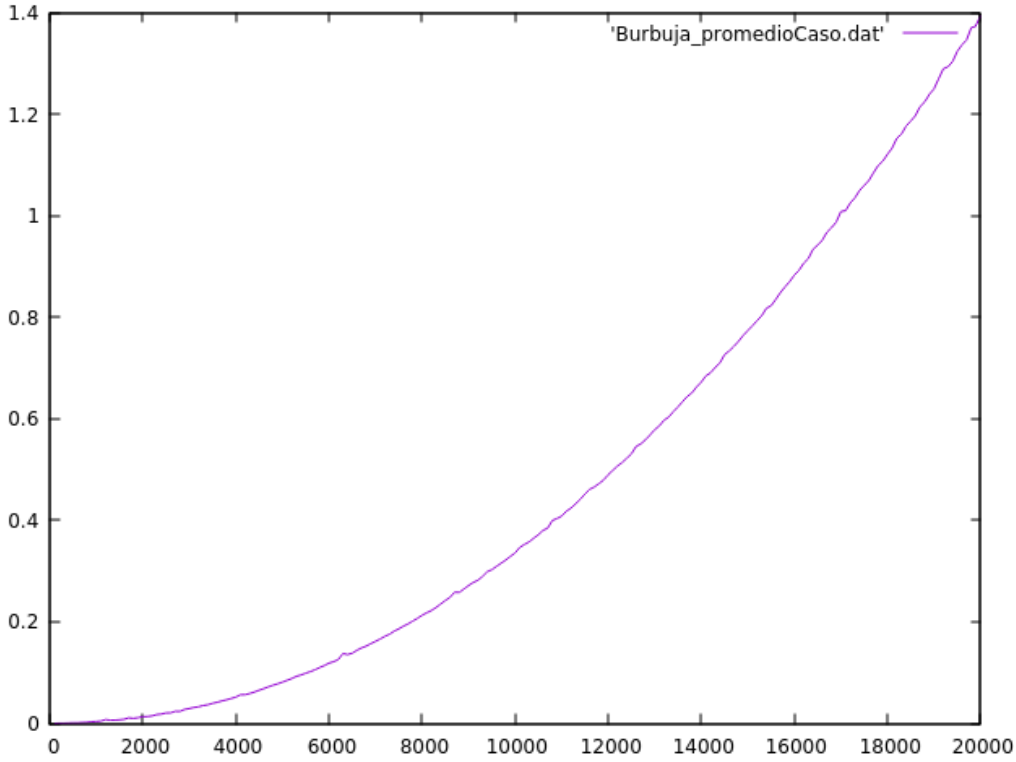
Aquí dejo el calculo de las constantes de la función que voy ajustar a los datos recopilados en el peor caso.

```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x), "Burbuja_peorCaso.dat" via a,b,c
                ^
         invalid expression

gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "Burbuja_peorCaso.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 4.9009977063e+19   0.00e+00  2.33e+08    1.000000e+00   1.000000e+00   1.000000e+00
   1 6.0377564295e+13  -8.12e+10  2.33e+07    1.068329e-03   9.999583e-01   1.000000e+00
   2 5.6540058727e+09  -1.07e+09  2.33e+06   -4.158227e-05   9.999479e-01   1.000000e+00
   3 5.6422767659e+09  -2.08e+02  2.33e+05   -4.155145e-05   9.989107e-01   9.999999e-01
   4 4.6308234742e+09  -2.18e+04  2.33e+04   -3.764275e-05   9.049458e-01   9.999874e-01
   5 3.5745792297e+07  -1.29e+07  2.33e+03   -3.301504e-06   7.938532e-02   9.998774e-01
   6 6.5932768285e+01  -5.42e+10  2.33e+02    3.095094e-09  -5.695078e-05   9.998608e-01
   7 3.2793059388e+01  -1.01e+05  2.33e+01    6.275861e-09  -1.333894e-04   9.992582e-01
   8 2.9170479908e+01  -1.24e+04  2.33e+00    6.066134e-09  -1.258263e-04   9.424345e-01
   9 6.5078516612e-01  -4.38e+06  2.33e-01    3.086815e-09  -1.839203e-05   1.353309e-01
  10 6.3574549200e-02  -9.24e+05  2.33e-02    2.594402e-09  -6.356202e-07   1.935258e-03
  11 6.3572945144e-02  -2.52e+00  2.33e-03    2.593587e-09  -6.062249e-07   1.714424e-03
  12 6.3572945144e-02  -5.89e-10  2.33e-04    2.593587e-09  -6.062244e-07   1.714421e-03
iter      chisq       delta/lim  lambda   a             b             c

After 12 iterations the fit converged.
final sum of squares of residuals : 0.0635729
rel. change during last iteration : -5.89402e-15

degrees of freedom    (FIT_NDF)                        : 297
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0146305
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00021405

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 2.59359e-09      +/- 1.259e-11    (0.4855%)
b               = -6.06224e-07     +/- 3.914e-07    (64.56%)
c               = 0.00171442       +/- 0.002551     (148.8%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.968  1.000
c               0.748 -0.868  1.000
gnuplot> plot f(x), "Burbuja_peorCaso.dat" w l
```

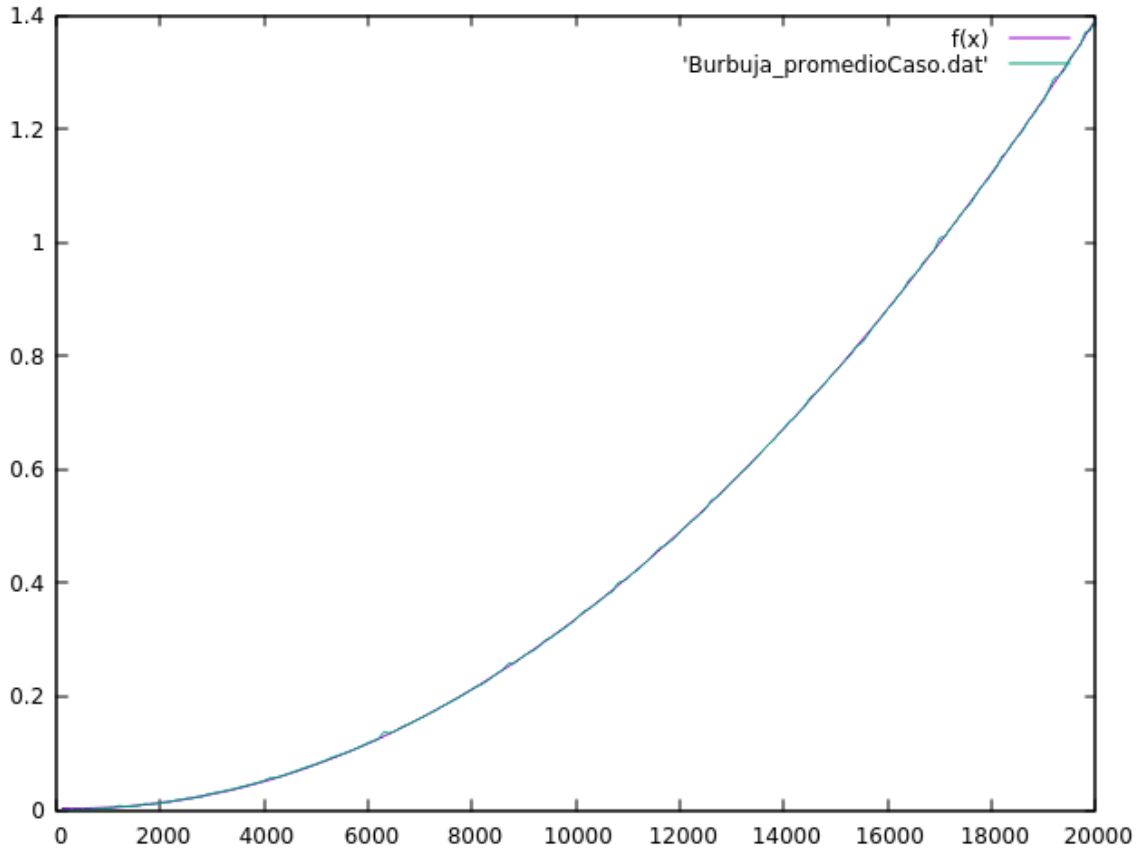
Finalmente ajustamos ambas gráficas:



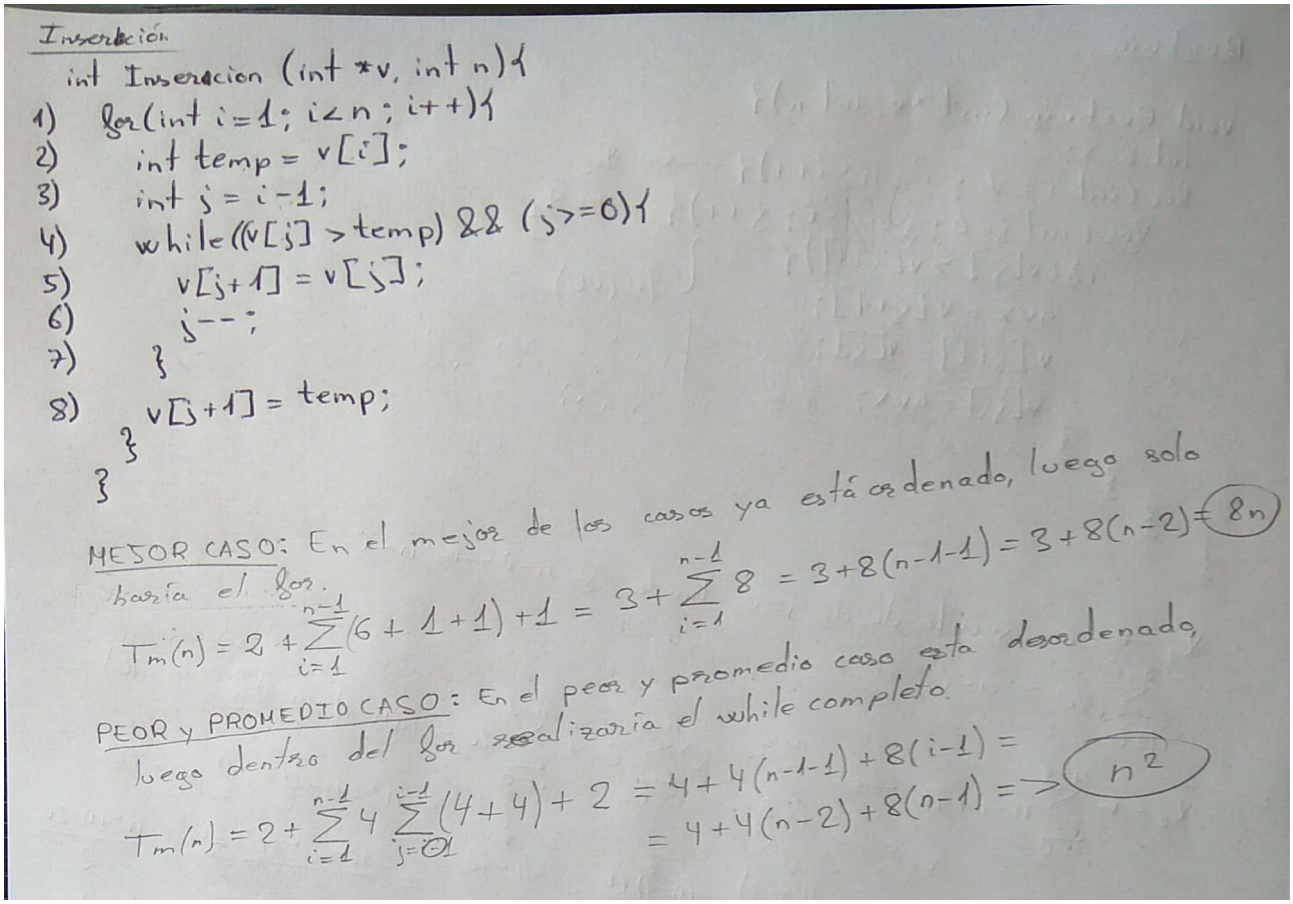
Lo mismo voy a realizar con el caso promedio, en este caso no explicare todo de nuevo por que ya lo he hecho anteriormente, es el mismo procedimiento teniendo en cuenta lo dicho anteriormente del caso promedio.



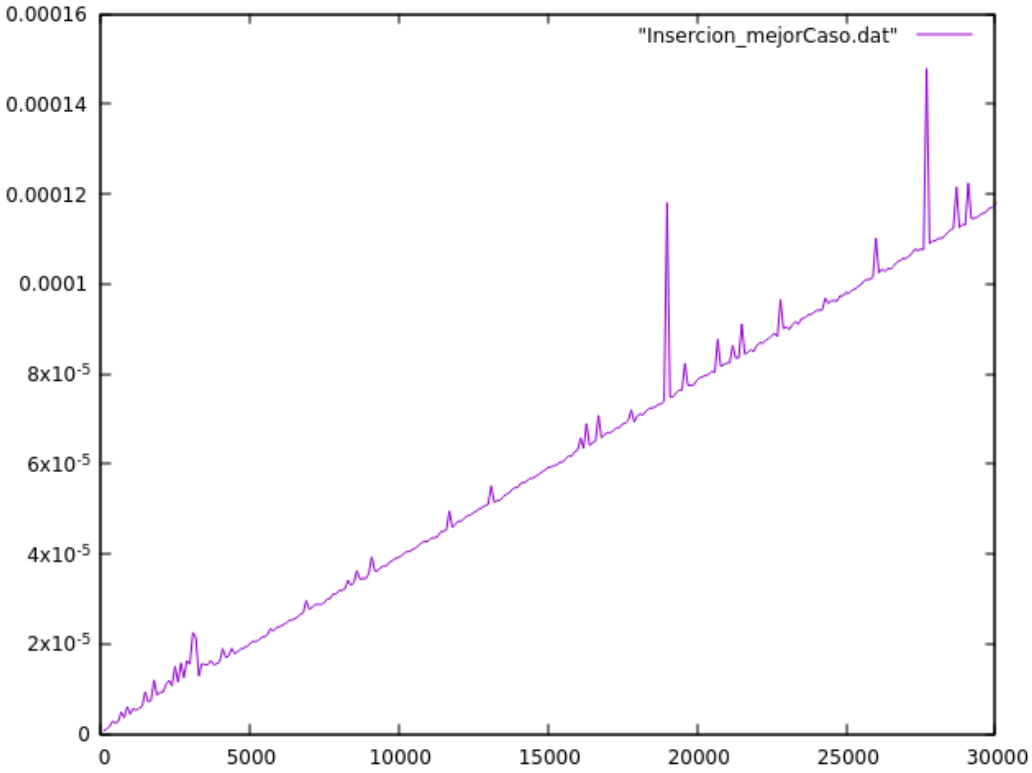
```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "Burbuja_promedioCaso.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 6.4810747219e+18   0.00e+00  1.04e+08    1.000000e+00   1.000000e+00   1.000000e+00
   1 1.7944804683e+13  -3.61e+10  1.04e+07    1.601656e-03   9.999376e-01   1.000000e+00
   2 1.6795546249e+09  -1.07e+09  1.04e+06   -6.231275e-05   9.999219e-01   1.000000e+00
   3 1.6743394332e+09  -3.11e+02  1.04e+05   -6.224370e-05   9.983695e-01   9.999997e-01
   4 1.2540846105e+09  -3.35e+04  1.04e+04   -5.386717e-05   8.640111e-01   9.999729e-01
   5 4.5803123014e+06  -2.73e+07  1.04e+03   -3.244277e-06   5.202665e-02   9.998107e-01
   6 2.3516159599e+01  -1.95e+10  1.04e+02    9.768180e-09  -1.678064e-04   9.997802e-01
   7 2.1536665810e+01  -9.19e+03  1.04e+01    1.184459e-08  -2.009784e-04   9.977749e-01
   8 1.4914858280e+01  -4.44e+04  1.04e+00    1.046092e-08  -1.676871e-04   8.308997e-01
   9 3.4136745406e-02  -4.36e+07  1.04e-01    3.923774e-09  -1.040348e-05   4.250722e-02
  10 8.4813894754e-04  -3.92e+06  1.04e-02    3.599777e-09  -2.608102e-06   3.432462e-03
  11 8.4813077036e-04  -9.64e-01  1.04e-03    3.599616e-09  -2.604237e-06   3.413086e-03
iter      chisq       delta/lim  lambda   a             b             c

After 11 iterations the fit converged.
final sum of squares of residuals : 0.000848131
rel. change during last iteration : -9.64141e-06

degrees of freedom    (FIT_NDF)                        : 197
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00207491
variance of residuals (reduced chisquare) = WSSR/ndf   : 4.30523e-06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 3.59962e-09      +/- 4.921e-12    (0.1367%)
b               = -2.60424e-06     +/- 1.021e-07    (3.922%)
c               = 0.00341309       +/- 0.0004446    (13.03%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.969  1.000
c               0.749 -0.868  1.000
gnuplot> plot f(x), 'Burbuja_promedioCaso.dat'
```



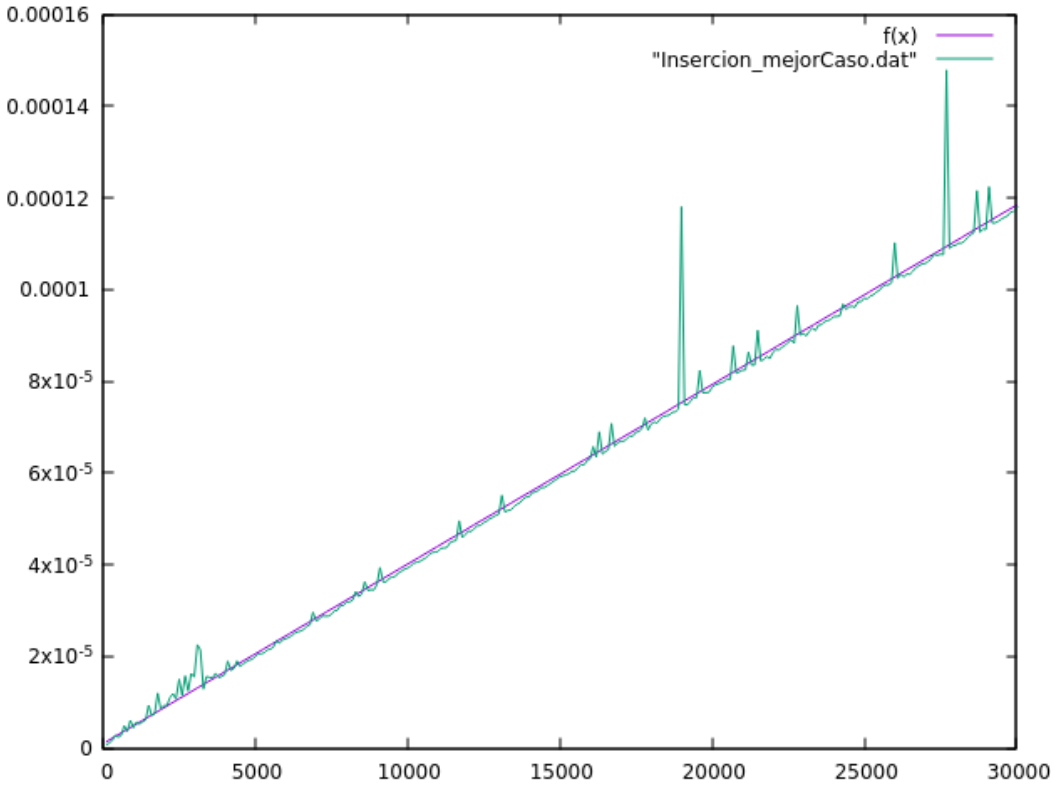
El segundo algoritmo que he estudiado es el algoritmo de ordenación por ***Inserción*** que funciona tal que inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento k+1 y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se inserta el elemento k+1 debiendo desplazarse los demás elementos. Lo primero que he realizado es calcular la eficiencia teórica de mi algoritmo, que finalmente he llegado a la conclusión de que es de eficiencia N^2 en el peor y caso promedio debido a que en esto casos va a tener que realizar todo el procedimiento del bucle for, así como el bucle while que esta dentro del bucle for, dando lugar a una eficiencia de N^2 (explicada en el procedimiento). Luego en el mejor de los casos, como solo va a tener que realizar el bucle for sin tener que realizar el bucle while, luego la eficiencia sería de orden N (constante). Dejo todo el procedimiento con el que he calculado todo con una captura:



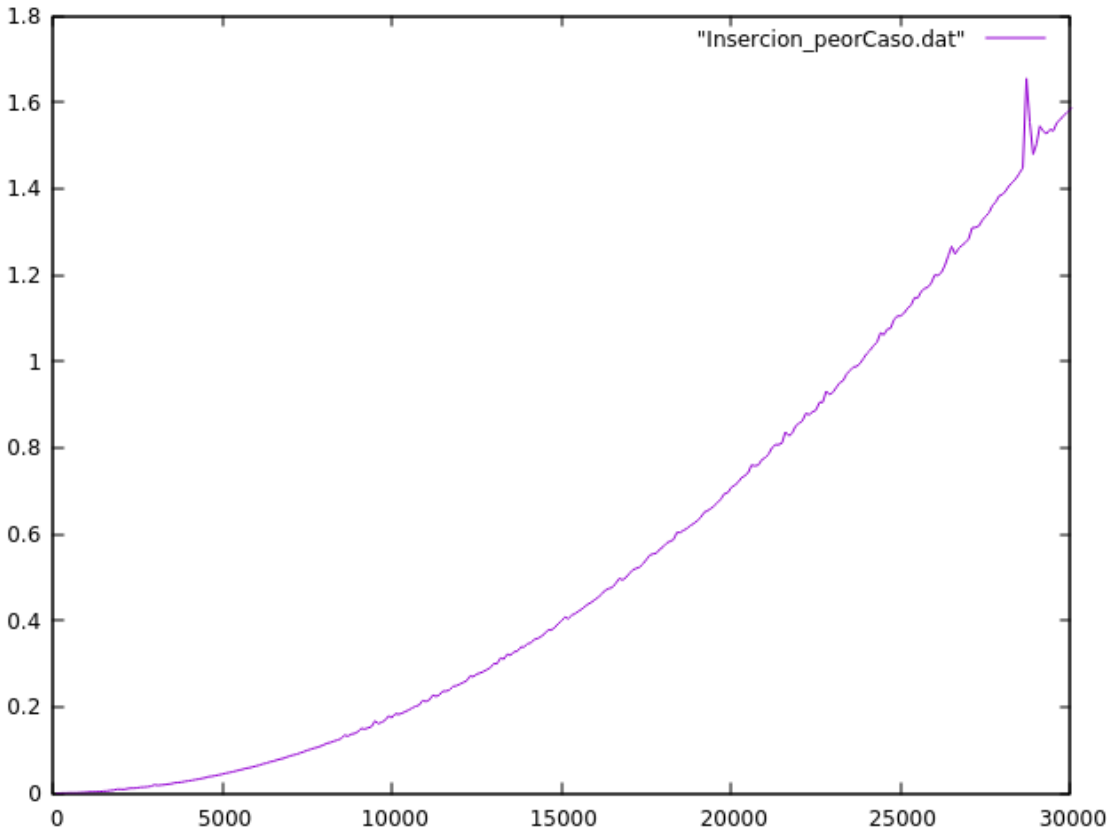
En cuanto a la eficiencia empírica de mi algoritmo lo he calculado igual que para el burbuja realizando muchas ejecuciones con diferentes tamaños de vector, en el mejor caso con el vector ordenado de primeras, en el peor caso el con el vector desordenado y en el caso promedio realizando una serie de ejecuciones, acumulando tiempos y dividiendo entre el numero de ejecuciones realizadas. Adjunto procedimiento realizados para el mejor, peor y caso promedio.



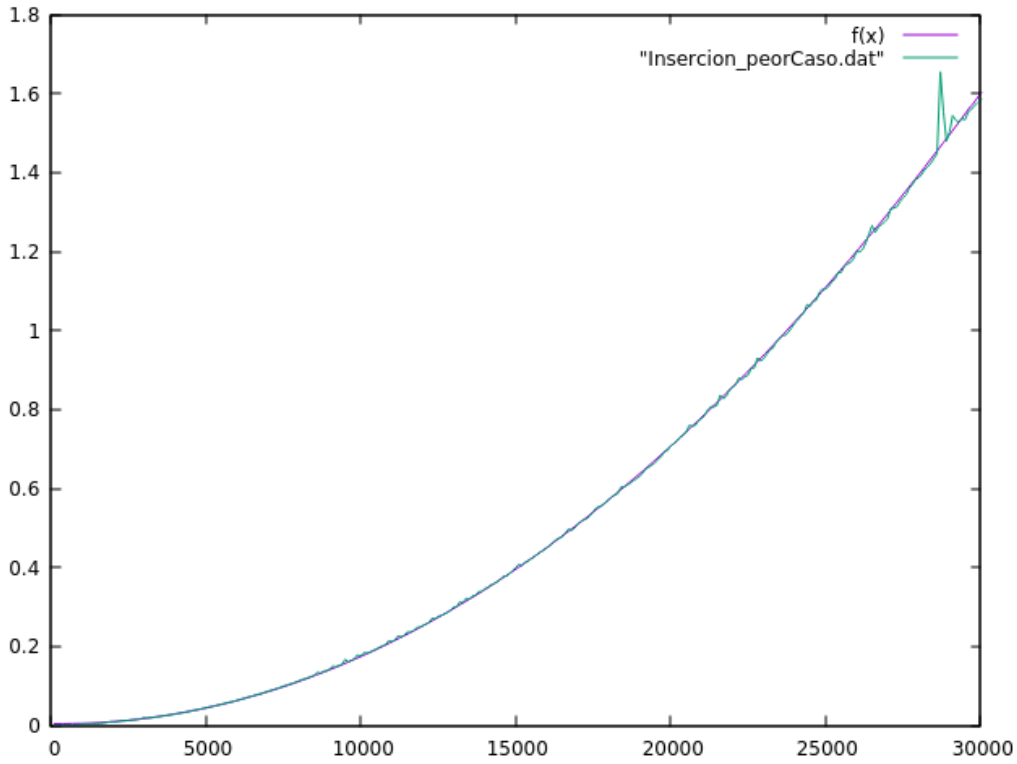

```
gnuplot> f(x)=a*x+b
gnuplot> fit f(x) "Insercion_mejorCaso.dat" via a,b
iter      chisq       delta/lim  lambda   a             b
   0 1.8122732471e-07   0.00e+00  3.18e-05    2.593587e-09  -6.062244e-07
   1 4.3060183898e-09  -4.11e+06  3.18e-06    3.987601e-09  -5.622392e-07
   2 4.1527453125e-09  -3.69e+03  3.18e-07    3.935776e-09   5.225747e-07
   3 4.1407316279e-09  -2.90e+02  3.18e-08    3.915820e-09   9.223296e-07
   4 4.1407314648e-09  -3.94e-03  3.18e-09    3.915747e-09   9.238081e-07
iter      chisq       delta/lim  lambda   a             b

After 4 iterations the fit converged.
final sum of squares of residuals : 4.14073e-09
rel. change during last iteration : -3.93992e-08

degrees of freedom    (FIT_NDF)                        : 298
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 3.72761e-06
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.38951e-11

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 3.91575e-09      +/- 2.485e-11    (0.6346%)
b               = 9.23808e-07      +/- 4.315e-07    (46.71%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -0.867  1.000
gnuplot> plot f(x), "Insercion_mejorCaso.dat" w l
```



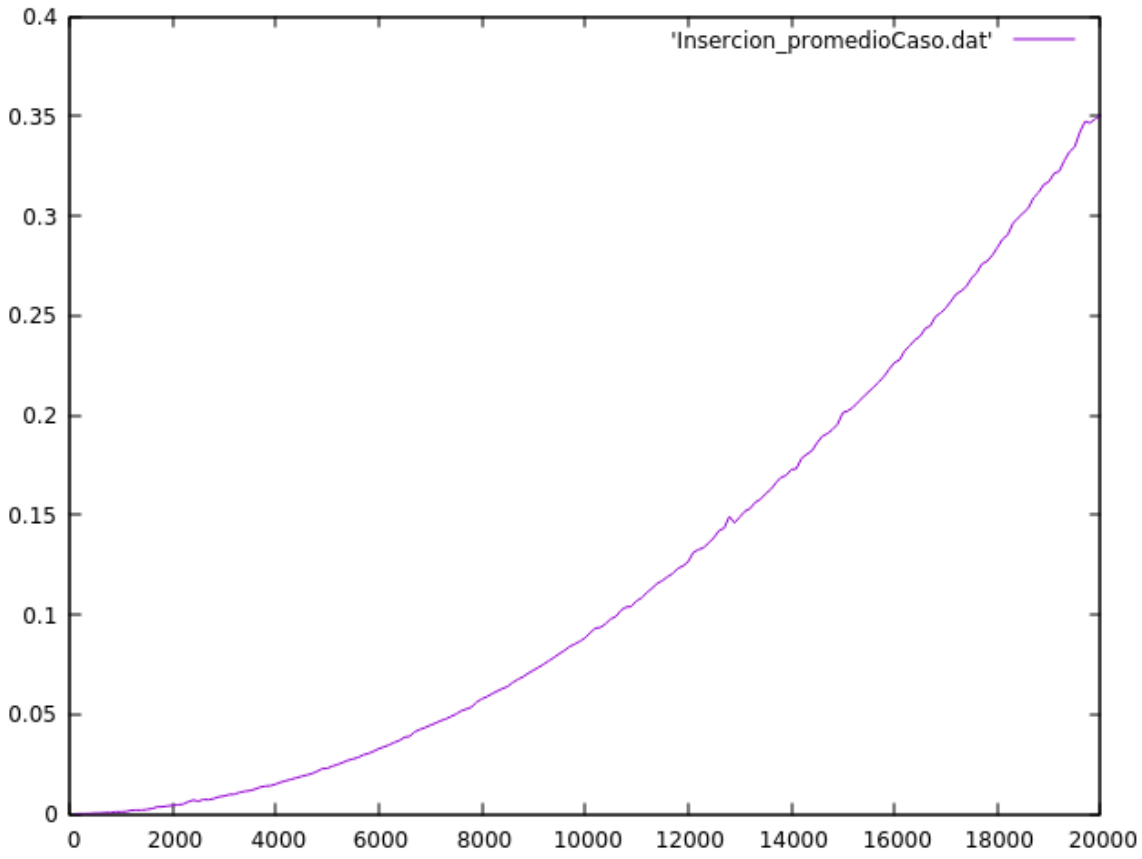
Como podemos observar en el mejor de los casos coincide con la eficiencia que hemos calculado teóricamente. Ahora voy a dejar el procedimiento para el peor caso:



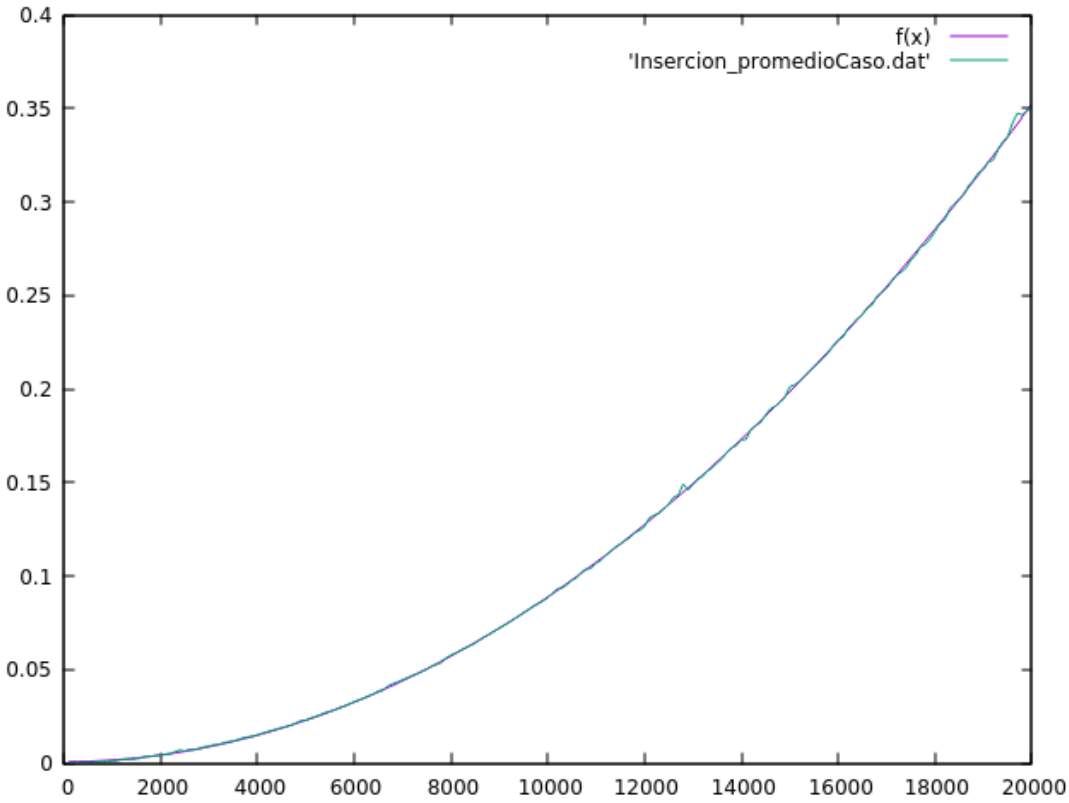
```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "Insercion_peorCaso.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 2.3368257340e+02   0.00e+00  9.14e-01    3.915747e-09   9.238081e-07   1.714421e-03
   1 6.7694422720e-02  -3.45e+08  9.14e-02    1.735439e-09   9.088197e-07   1.709391e-03
   2 5.7183066087e-02  -1.84e+04  9.14e-03    1.759176e-09   2.910462e-07   1.458954e-03
   3 5.0539142982e-02  -1.31e+04  9.14e-04    1.807437e-09  -9.251029e-07   2.720302e-03
   4 5.0455466259e-02  -1.66e+02  9.14e-05    1.814057e-09  -1.153324e-06   4.276978e-03
   5 5.0455459478e-02  -1.34e-02  9.14e-06    1.814111e-09  -1.155273e-06   4.291326e-03
iter      chisq       delta/lim  lambda   a             b             c

After 5 iterations the fit converged.
final sum of squares of residuals : 0.0504555
rel. change during last iteration : -1.34392e-07

degrees of freedom    (FIT_NDF)                        : 297
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0130339
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000169884

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 1.81411e-09      +/- 1.122e-11    (0.6184%)
b               = -1.15527e-06     +/- 3.487e-07    (30.18%)
c               = 0.00429133       +/- 0.002273     (52.96%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.968  1.000
c               0.748 -0.868  1.000
gnuplot> plot f(x), "Insercion_peorCaso.dat" w l
```



Una vez más la eficiencia empírica hace reflejo de la eficiencia teórica calculada anteriormente. Por ultimo dejo el caso promedio, para finalmente concluir que este algoritmo a excepción del mejor de los casos siempre será de eficiencia N^2.




```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "Insercion_promedioCaso.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 1.3697816092e+00   0.00e+00  1.39e-01    1.336320e-09   2.433282e-08   9.366535e-04
   1 1.7393933897e-04  -7.87e+08  1.39e-02    8.773626e-10   2.432208e-08   9.350020e-04
   2 1.6876721124e-04  -3.06e+03  1.39e-03    8.768137e-10   2.424882e-08   8.839978e-04
   3 1.6718342463e-04  -9.47e+02  1.39e-04    8.771031e-10   2.897831e-08   7.430758e-04
   4 1.6652014034e-04  -3.98e+02  1.39e-05    8.753813e-10   6.617801e-08   5.989837e-04
   5 1.6651651182e-04  -2.18e+00  1.39e-06    8.752432e-10   6.914062e-08   5.877480e-04
   6 1.6651651182e-04  -1.38e-06  1.39e-07    8.752431e-10   6.914298e-08   5.877391e-04
iter      chisq       delta/lim  lambda   a             b             c

After 6 iterations the fit converged.
final sum of squares of residuals : 0.000166517
rel. change during last iteration : -1.38162e-11

degrees of freedom    (FIT_NDF)                        : 197
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.000919381
variance of residuals (reduced chisquare) = WSSR/ndf   : 8.45261e-07

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 8.75243e-10      +/- 2.181e-12    (0.2491%)
b               = 6.9143e-08       +/- 4.525e-08    (65.45%)
c               = 0.000587739      +/- 0.000197     (33.52%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.969  1.000
c               0.749 -0.868  1.000
gnuplot> plot f(x), 'Insercion_promedioCaso.dat'
```



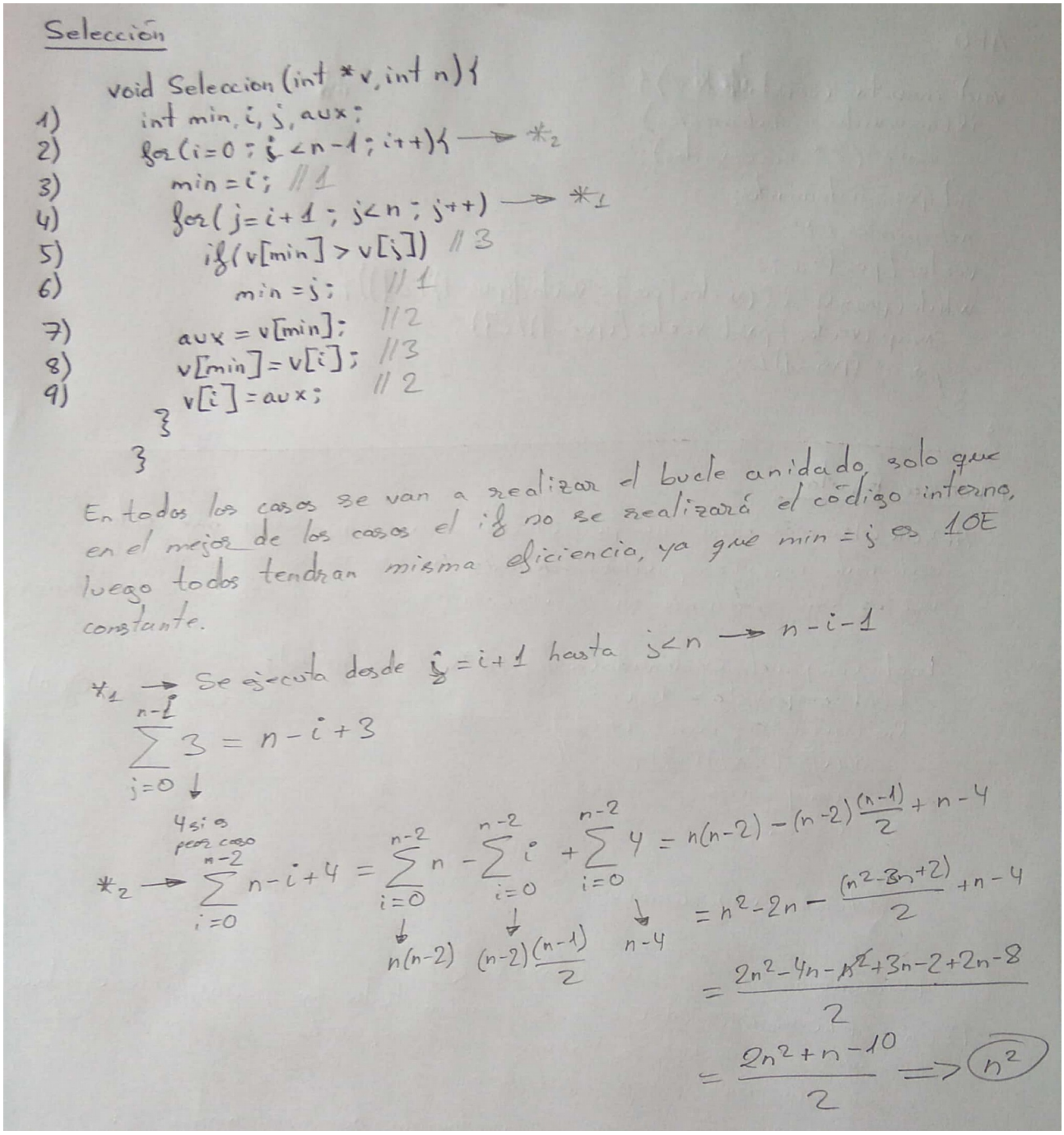
Con esto dejamos claro que hemos realizado correctamente el calculo de la eficiencia teórica, pues se ve reflejado en el calculo de la eficiencia empírica. Por ultimo voy a calcular eficiencia del ultimo algoritmo de ordenación básico conocido como algoritmo de ***Selección*** el cual funciona de la siguiente manera:

1)Buscar el mínimo elemento de la lista

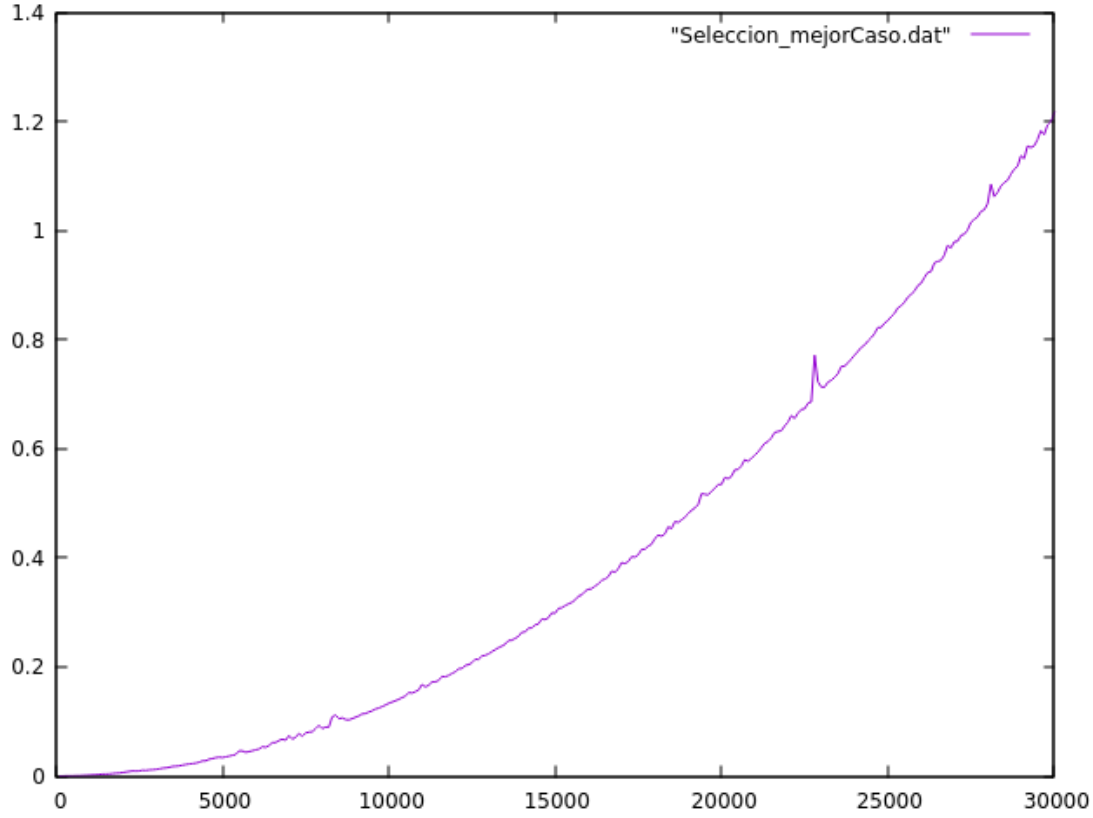
2)Intercambiarlo con el primero

3)Buscar el siguiente mínimo en el resto de la lista
4)Intercambiarlo con el segundo

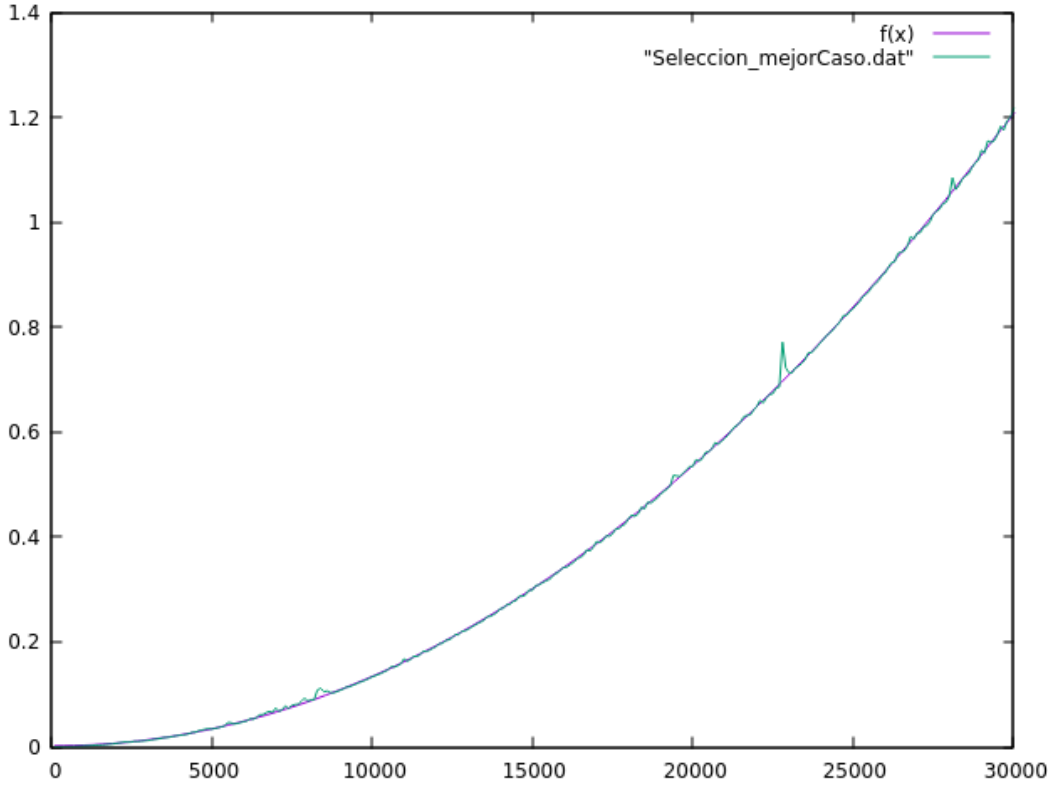
Lo primero que he realizado nuevamente ha sido calcular su eficiencia teórica. Destacar que su eficiencia va a ser de siempre N^2 debido a que siempre ya sea que el vector este o no ordenado va a realizar los dos bucles for, lo que hace que este algoritmo se nos vaya a N^2 en todos los casos, dejo claro todo esto en la captura que dejo a continuación.



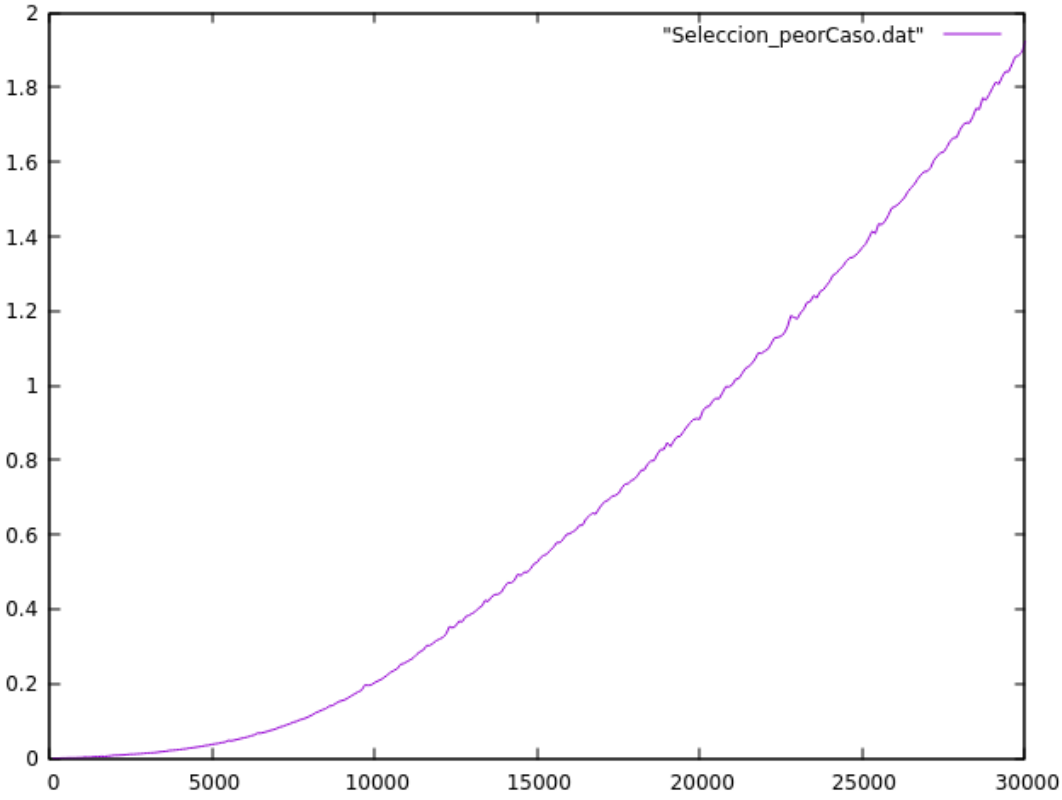
Para comprobar que lo que hemos calculado teóricamente se corresponde con la realidad, he lanzado este algoritmo igual que con los anteriores con diferentes tamaños de vectores y en las situaciones de mejor, peor y caso promedio, siendo el mejor caso que el vector este ordenado, que el peor caso este desordenado y que el caso promedio sea el calculo de los tiempos de varias ejecuciones y su media. Dejo todo el procedimiento aquí:



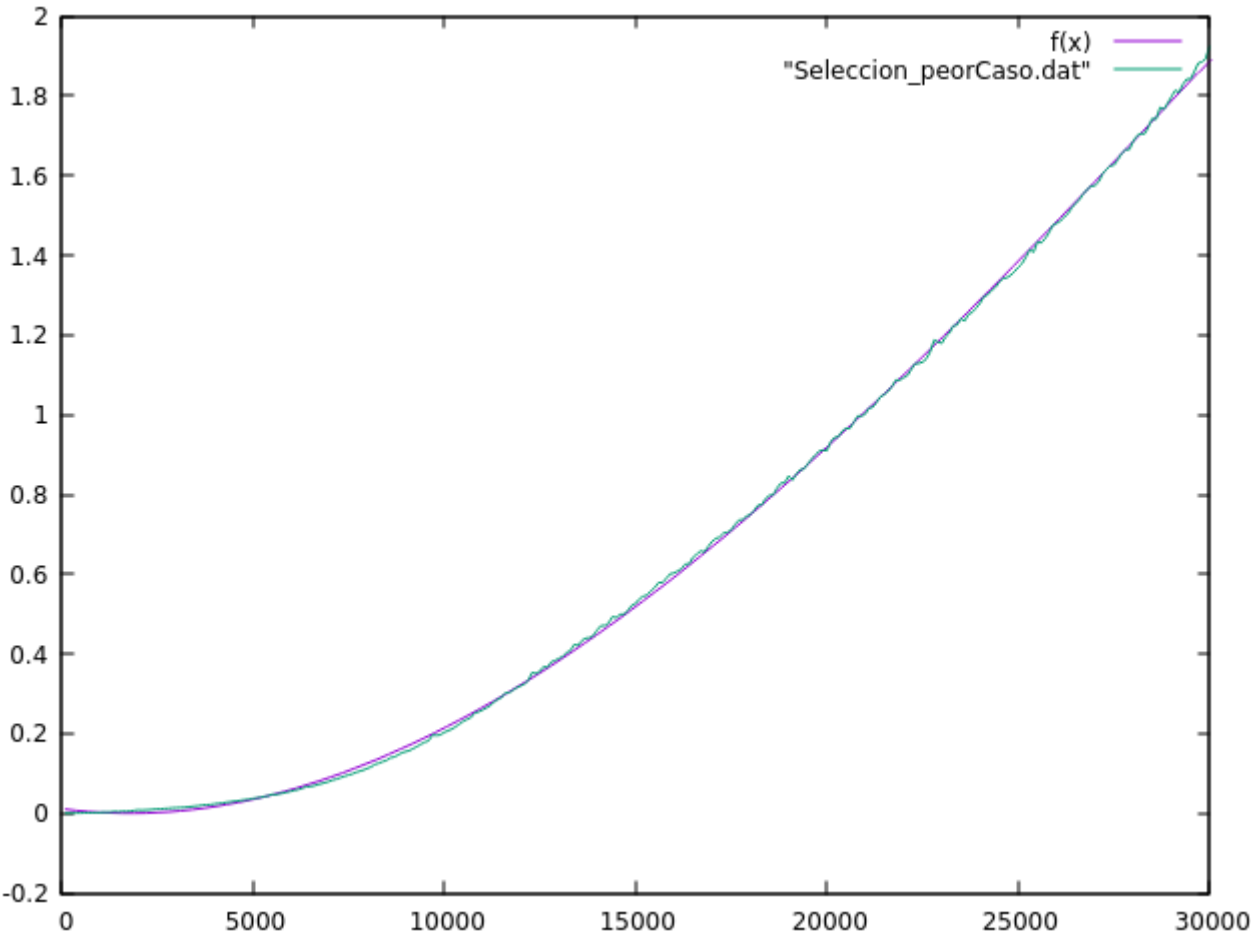

```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "Seleccion_mejorCaso.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 9.1895080440e+00   0.00e+00  4.23e-01    1.814111e-09  -1.155273e-06   4.291326e-03
   1 1.1709971072e-02  -7.84e+07  4.23e-02    1.381134e-09  -1.139680e-06   4.320999e-03
   2 1.0087164225e-02  -1.61e+04  4.23e-03    1.362392e-09  -7.011695e-07   4.331959e-03
   3 9.8437707526e-03  -2.47e+03  4.23e-04    1.349734e-09  -2.964265e-07   2.067688e-03
   4 9.8435046391e-03  -2.70e+00  4.23e-05    1.349366e-09  -2.836139e-07   1.978662e-03
   5 9.8435046390e-03  -3.77e-07  4.23e-06    1.349365e-09  -2.836092e-07   1.978629e-03
iter      chisq       delta/lim  lambda   a             b             c

After 5 iterations the fit converged.
final sum of squares of residuals : 0.0098435
rel. change during last iteration : -3.76622e-12

degrees of freedom    (FIT_NDF)                        : 297
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00575701
variance of residuals (reduced chisquare) = WSSR/ndf   : 3.31431e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 1.34937e-09      +/- 4.955e-12    (0.3672%)
b               = -2.83609e-07     +/- 1.54e-07     (54.3%)
c               = 0.00197863       +/- 0.001004     (50.73%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.968  1.000
c               0.748 -0.868  1.000
gnuplot> plot f(x), "Seleccion_mejorCaso.dat" w l
```



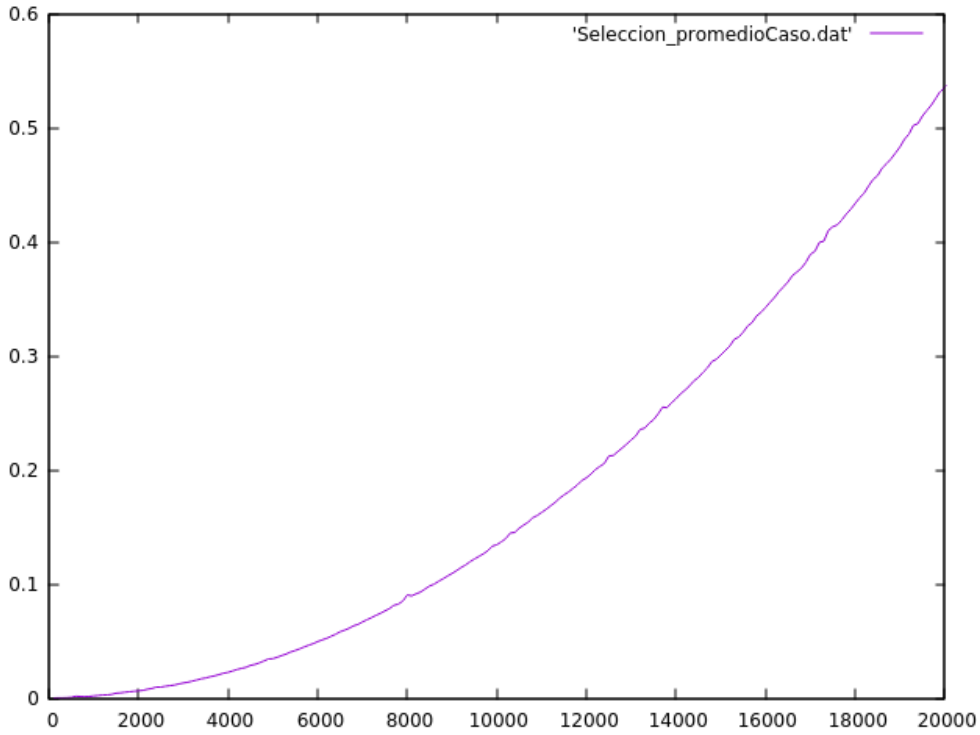
Luego el del peor caso es el siguiente:



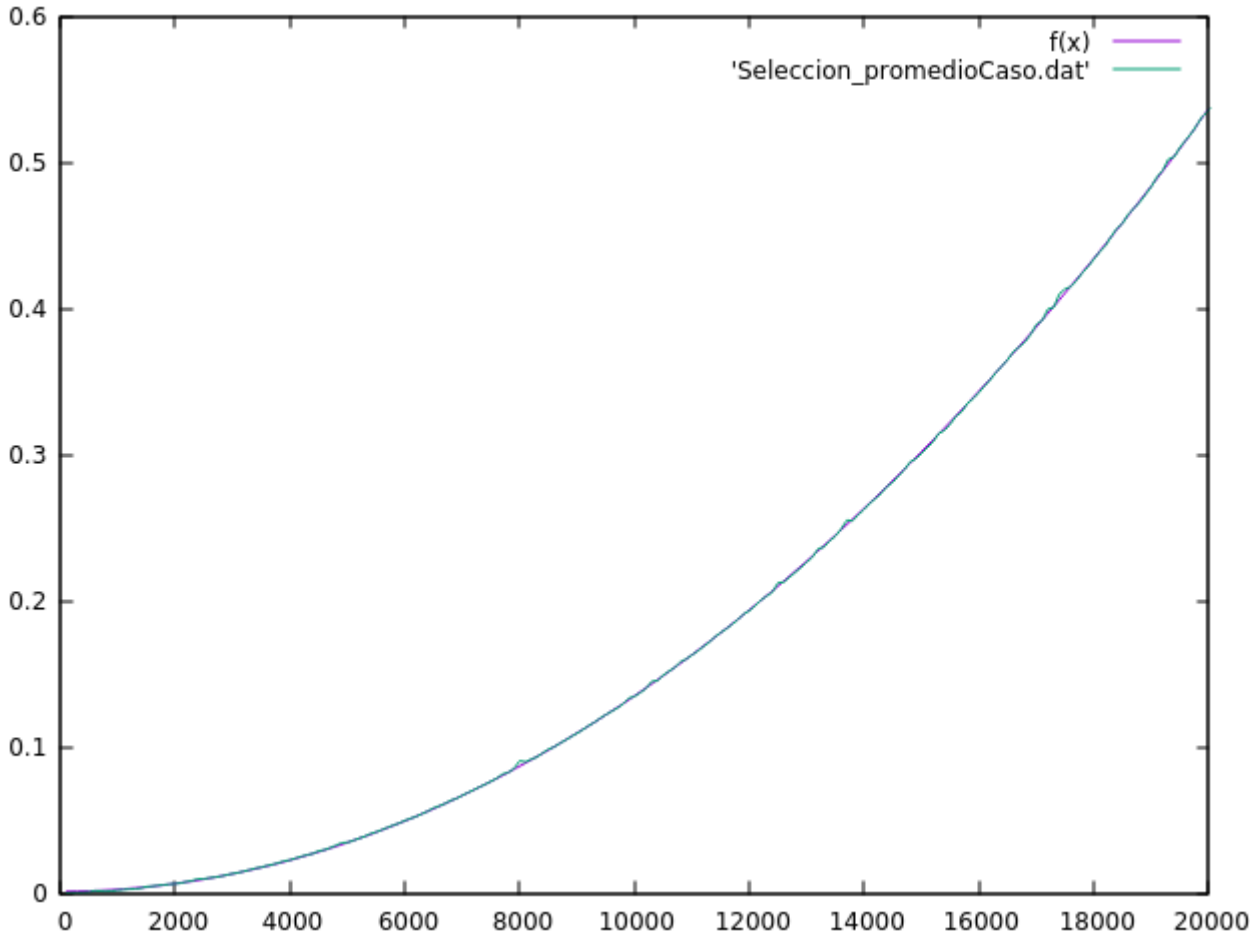
```
gnuplot> f(x)=a*x**3+b*x**2+c*x+d
gnuplot> fit f(x) "Seleccion_peorCaso.dat" via a,b,c,d
iter      chisq       delta/lim  lambda   a             b             c             d
   0 2.2113374182e-02   0.00e+00  7.83e-01   -3.952879e-14   3.700968e-09  -1.283871e-05   1.127334e-
02
   * 2.2113374182e-02   4.71e-10  7.83e+00   -3.952879e-14   3.700968e-09  -1.283871e-05   1.127334e-
02
   * 2.2113374182e-02   2.51e-10  7.83e+01   -3.952879e-14   3.700968e-09  -1.283871e-05   1.127334e-
02
   * 2.2113374182e-02   3.92e-10  7.83e+02   -3.952879e-14   3.700968e-09  -1.283871e-05   1.127334e-
02
   1 2.2113374182e-02   0.00e+00  7.83e+01   -3.952879e-14   3.700968e-09  -1.283871e-05   1.127334e-
02
iter      chisq       delta/lim  lambda   a             b             c             d

After 1 iterations the fit converged.
final sum of squares of residuals : 0.0221134
rel. change during last iteration : 0

degrees of freedom    (FIT_NDF)                        : 296
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00864334
variance of residuals (reduced chisquare) = WSSR/ndf   : 7.47073e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = -3.95288e-14     +/- 9.781e-16    (2.474%)
b               = 3.70097e-09      +/- 4.478e-11    (1.21%)
c               = -1.28387e-05     +/- 5.806e-07    (4.522%)
d               = 0.0112733        +/- 0.002021     (17.93%)

correlation matrix of the fit parameters:
                a      b      c      d
a               1.000
b              -0.986  1.000
c               0.917 -0.969  1.000
d              -0.666  0.750 -0.869  1.000
gnuplot> plot f(x), "Seleccion_peorCaso.dat" w l
```



Y por ultimo dejo el procedimiento del caso promedio:



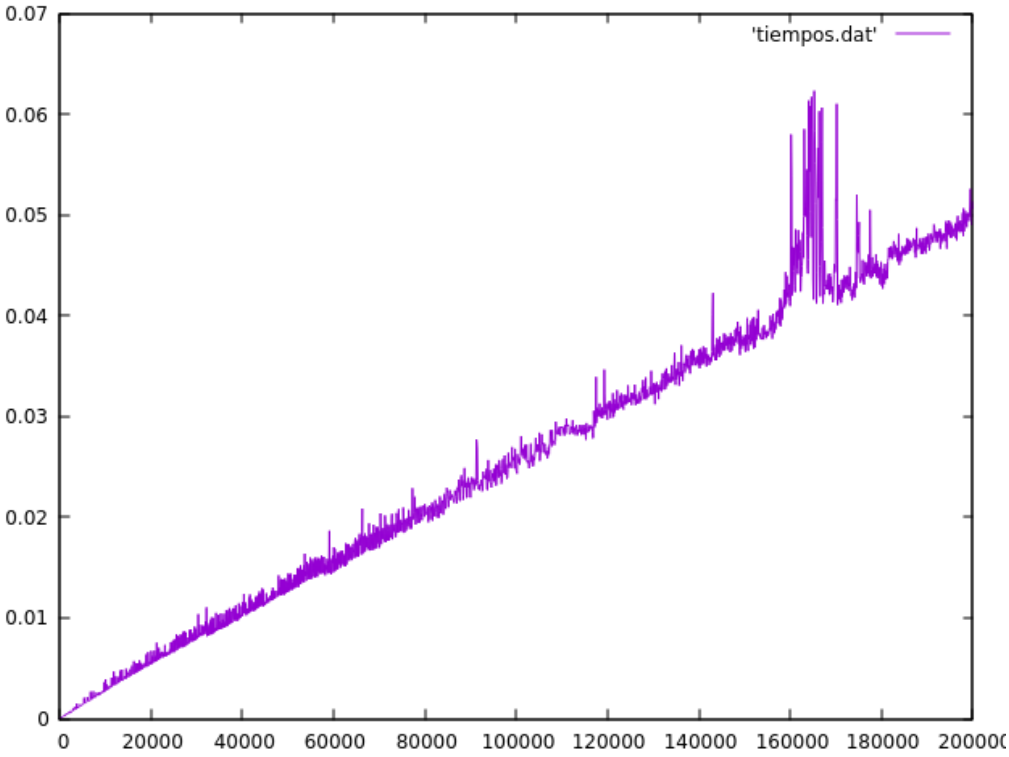

```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "Seleccion_promedioCaso.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 2.8850149001e+01   0.00e+00  3.75e-01    3.599616e-09  -2.604237e-06    3.413086e-03
   1 7.2808426522e-03  -3.96e+08  3.75e-02    1.487297e-09  -2.508349e-06    3.457131e-03
   2 3.7788367901e-04  -1.83e+06  3.75e-03    1.372038e-09  -7.249554e-07    3.589230e-03
   3 1.4378929278e-04  -1.63e+05  3.75e-04    1.337634e-09  -6.846460e-09    1.088501e-03
   4 1.4328576120e-04  -3.51e+02  3.75e-05    1.336320e-09   2.431453e-08    9.367433e-04
   5 1.4328576102e-04  -1.23e-04  3.75e-06    1.336320e-09   2.433282e-08    9.366535e-04
iter      chisq       delta/lim  lambda   a             b             c

After 5 iterations the fit converged.
final sum of squares of residuals : 0.000143286
rel. change during last iteration : -1.22737e-09

degrees of freedom    (FIT_NDF)                        : 197
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.000852842
variance of residuals (reduced chisquare) = WSSR/ndf   : 7.27339e-07

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 1.33632e-09      +/- 2.023e-12    (0.1514%)
b               = 2.43328e-08      +/- 4.198e-08    (172.5%)
c               = 0.000936654      +/- 0.0001827    (19.51%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.969  1.000
c               0.749 -0.868  1.000
gnuplot> plot f(x), 'Seleccion_promedioCaso.dat'
```



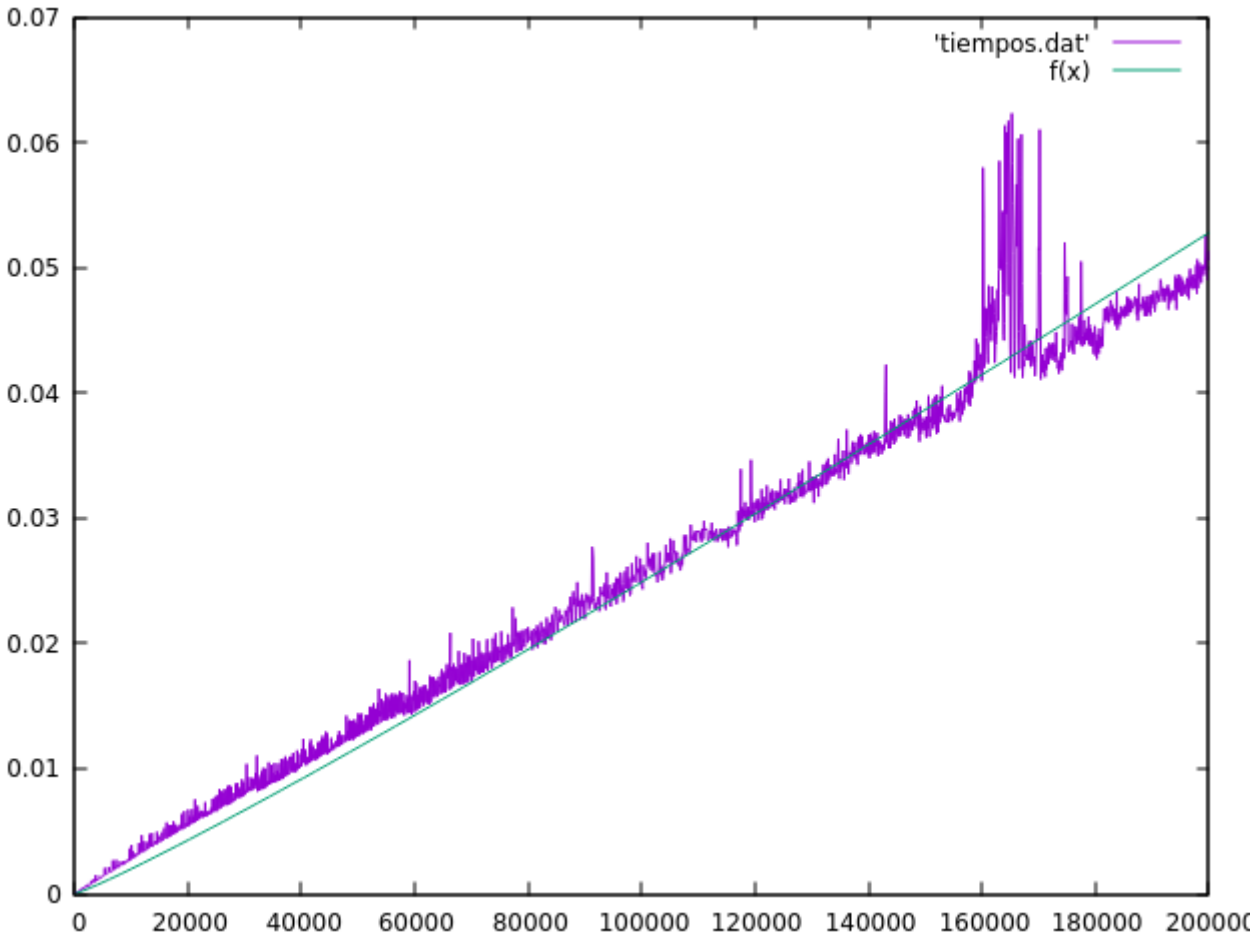
Como hemos visto todo se corresponde con lo calculado teóricamente, luego podemos deducir que hemos calculado correctamente la eficiencia teórica, así como la empírica. Con esto deducimos que estos algoritmos el mejor en uno de los casos sería el de Inserción, que en el caso de que inserten el vector ordenado su eficiencia sería de orden constante, los demás todos sería de N^2 en cualquier caso.

Algoritmos de Ordenación con Estructuras Jerárquicas

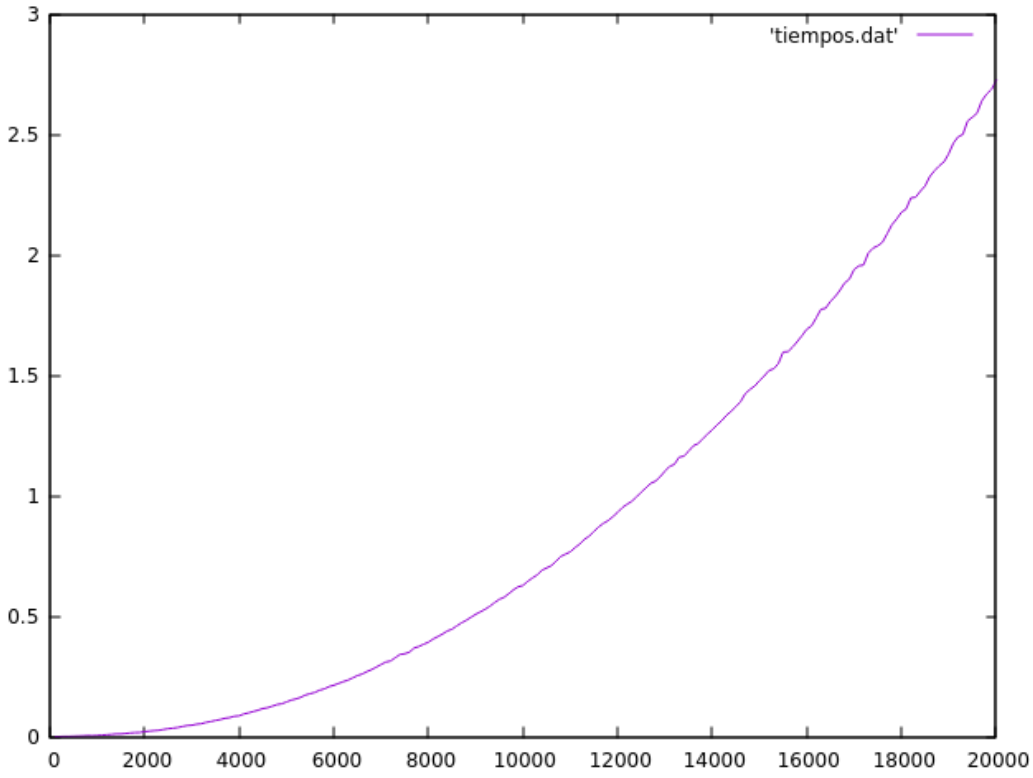
En este caso vamos a comprobar la eficiencia de ordenamiento usando estructuras jerárquicas, para ello vamos a usar el ABB y la estructura APO, la primera funciona insertando los valores en el árbol, insertando como hijo izquierda de la raíz valores más pequeños que la raíz y a la derecha los valores mayores. En el caso del APO, todo nodo debe ser mayor que sus hijos, siendo la raíz principal el valor mayor de todo el árbol, se van insertando de izquierda a derecha por niveles. Voy a explicar primero como he calculado la eficiencia del ABB, para ello destacar que para sacar los valores de un ABB ordenados hay que recorrer el árbol en inorden, para ello debo calcular el tiempo que tarda en insertar los valores de un vector en el árbol y el listarlos en in orden, para ello primero inserto los valores en el árbol con el método Insertar ya incluido en el código y luego se lista en inorden con el operator++ (que no listaremos por pantalla para evitar aumentar el tiempo de ejecución). En el código .cpp se puede ver todo esto. En el caso del ABB teóricamente en el peor de los casos será de N^2 y en el caso promedio y mejor será de nlog(n), como el ejercicio no pedía calcular su eficiencia teórica, tendré en cuenta las eficiencias que dijimos en clase, que son las que he citado antes. Voy a dejar el procedimiento que he realizado primero para el mejor/promedio caso:



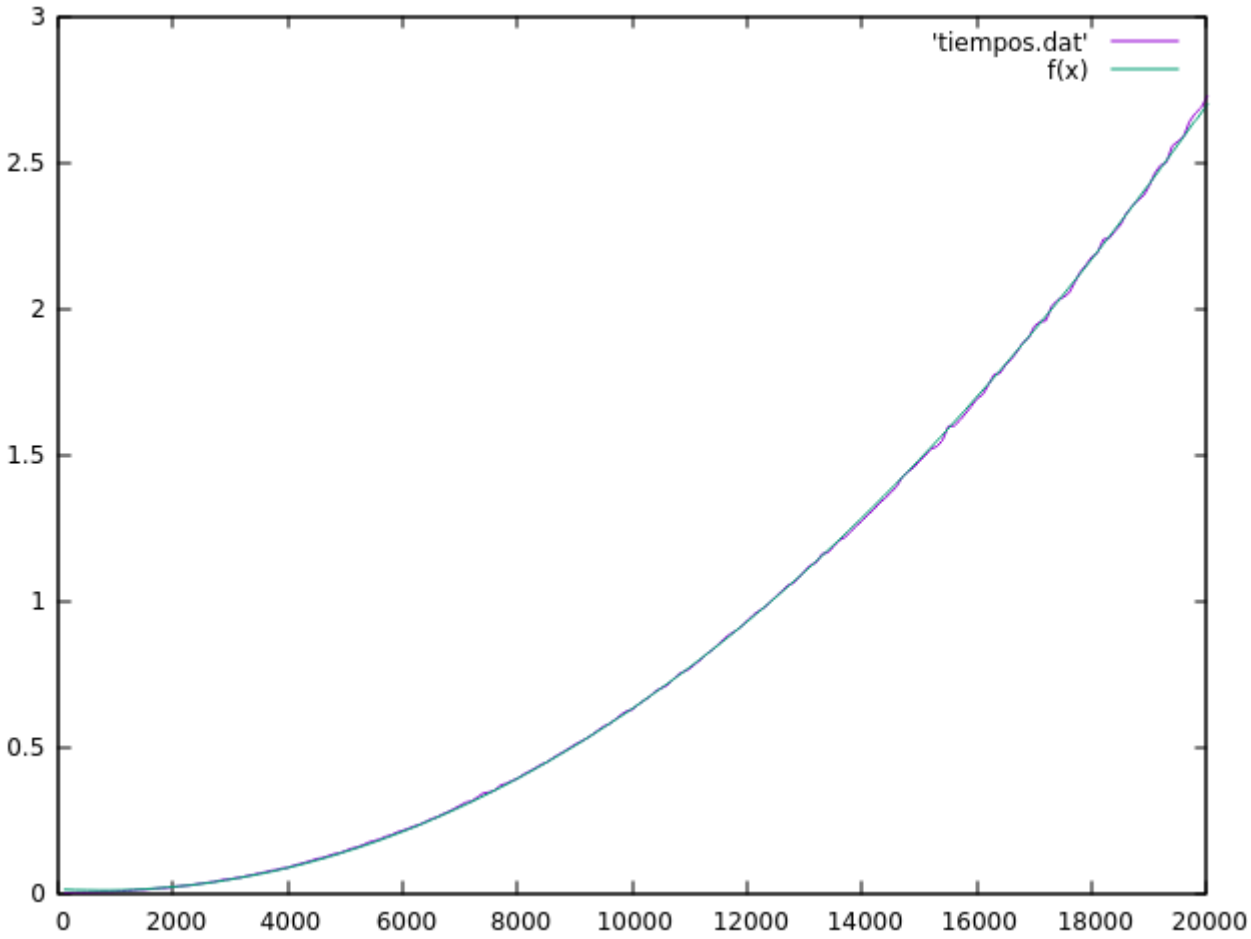

```
gnuplot> f(x)=a*x*log(b*x)
gnuplot> fit f(x) "tiempos.dat" via a,b
iter      chisq       delta/lim  lambda   a             b
   0 3.7649281357e+15   0.00e+00  9.74e+05    1.000000e+00   1.000000e+00
   1 1.8828580067e+11  -2.00e+09  9.74e+04    7.122926e-03   9.182927e-01
   2 1.6889881473e+00  -1.11e+16  9.74e+03    4.304145e-08   9.182880e-01
   3 1.0742257933e-02  -1.56e+07  9.74e+02    2.177592e-08   9.182880e-01
   4 1.0742257933e-02  -5.25e-07  9.74e+01    2.177592e-08   9.182880e-01
iter      chisq       delta/lim  lambda   a             b

After 4 iterations the fit converged.
final sum of squares of residuals : 0.0107423
rel. change during last iteration : -5.24862e-12

degrees of freedom    (FIT_NDF)                        : 1998
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00231873
variance of residuals (reduced chisquare) = WSSR/ndf   : 5.37651e-06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 2.17759e-08      +/- 1.347e-09    (6.184%)
b               = 0.918288         +/- 0.67         (72.96%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -1.000  1.000
gnuplot> plot 'tiempos.dat' w l, f(x) w l
```



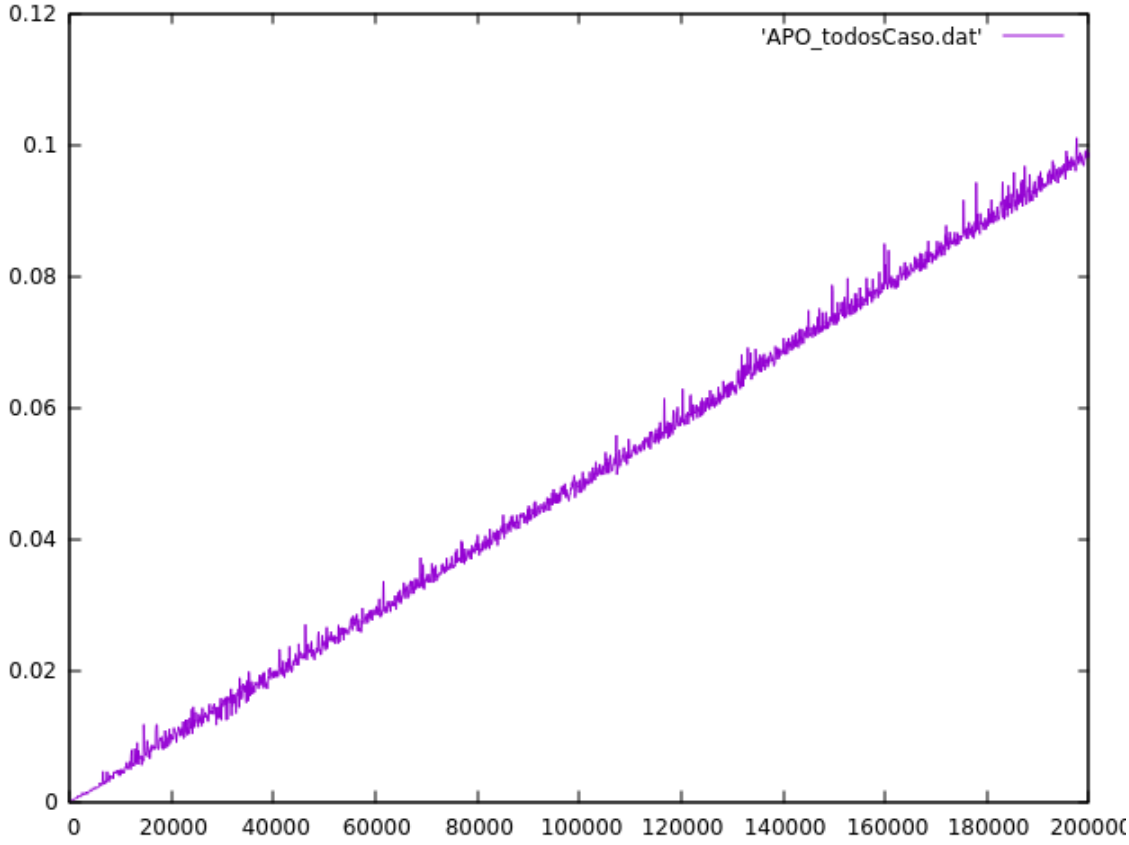
En este caso he calculado tanto el caso promedio como el caso mejor, como vemos nuestra eficiencia teórica hace referencia a lo calculado empíricamente, luego deducimos que hemos realizado bien el procedimiento, como un ABB las inserciones son de eficiencia nlog(n) debido a que metemos n datos y insertar un elemento es log(n) luego nlog(n). La eficiencia es esta también por que el vector se inserta con valores aleatorios, ya que si insertamos un vector ordenado o con el orden al revés la eficiencia empeoraría debido a que el árbol tendría los valores en una única rama. En el caso peor, que es lo que acabo de explicar, al insertar un vector ordenado, el árbol solo tendrá los valores en una rama, luego la eficiencia empeoraría hasta tener una eficiencia de N^2, dejo todo el procedimiento aquí con el que he obtenido este dato (en el código se ve como lo he realizado):



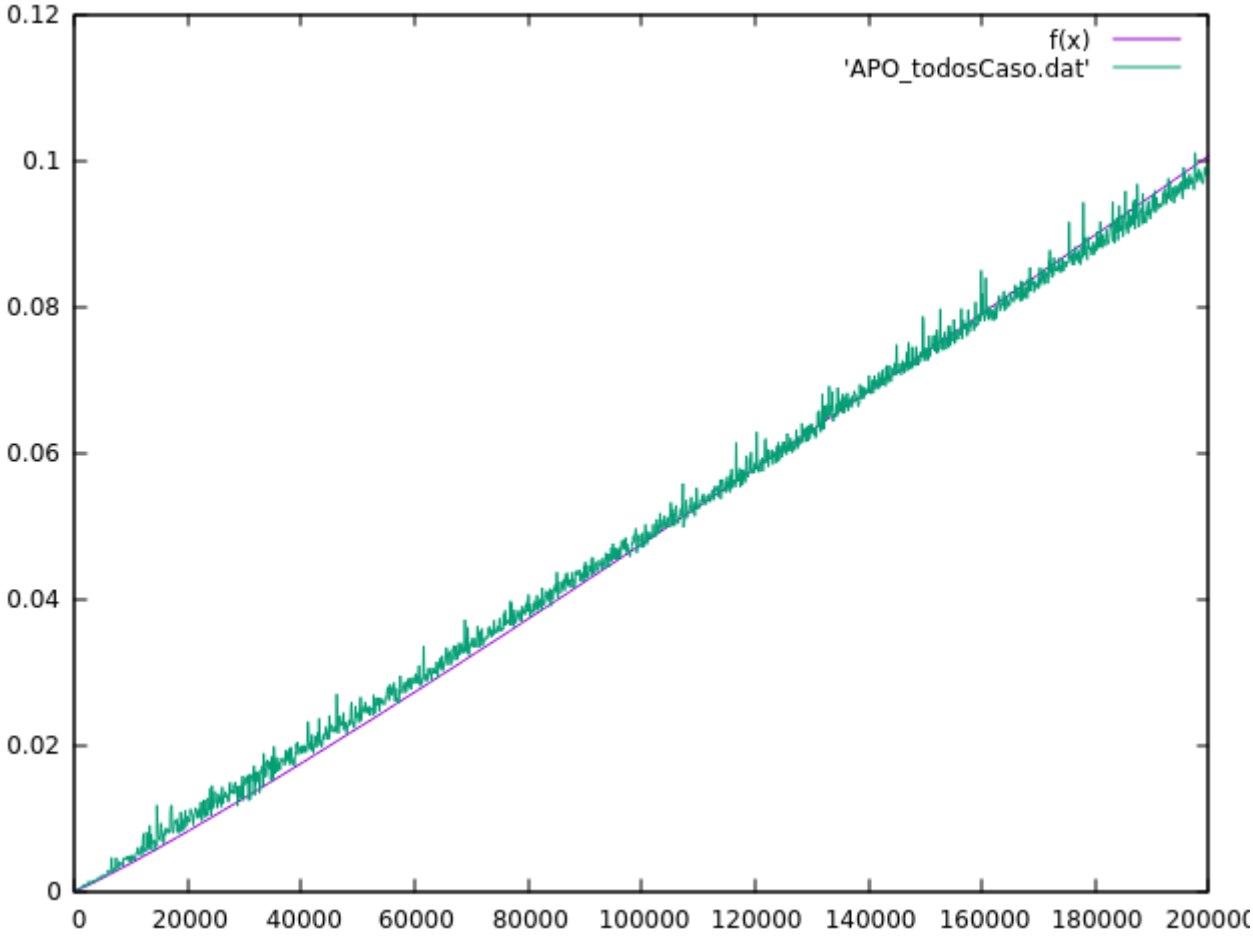

```
gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) "tiempos.dat" via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 6.4810746806e+18   0.00e+00  1.04e+08    1.000000e+00   1.000000e+00   1.000000e+00
   1 1.7944804589e+13  -3.61e+10  1.04e+07    1.601659e-03   9.999376e-01   1.000000e+00
   2 1.6795752745e+09  -1.07e+09  1.04e+06   -6.230957e-05   9.999219e-01   1.000000e+00
   3 1.6743600187e+09  -3.11e+02  1.04e+05   -6.224051e-05   9.983695e-01   9.999997e-01
   4 1.2541000289e+09  -3.35e+04  1.04e+04   -5.386393e-05   8.640103e-01   9.999729e-01
   5 4.5803682194e+06  -2.73e+07  1.04e+03   -3.240729e-06   5.202082e-02   9.998107e-01
   6 2.3119942593e+01  -1.98e+10  1.04e+02    1.333692e-08  -1.739545e-04   9.997804e-01
   7 2.1142057376e+01  -9.36e+03  1.04e+01    1.541350e-08  -2.071305e-04   9.977940e-01
   8 1.4644527852e+01  -4.44e+04  1.04e+00    1.404288e-08  -1.741531e-04   8.324922e-01
   9 4.3087527927e-02  -3.39e+07  1.04e-01    7.567369e-09  -1.835242e-05   5.153300e-02
  10 1.0423681651e-02  -3.13e+05  1.04e-02    7.246426e-09  -1.063053e-05   1.282666e-02
  11 1.0423673627e-02  -7.70e-02  1.04e-03    7.246267e-09  -1.062670e-05   1.280747e-02
iter      chisq       delta/lim  lambda   a             b             c

After 11 iterations the fit converged.
final sum of squares of residuals : 0.0104237
rel. change during last iteration : -7.69758e-07

degrees of freedom    (FIT_NDF)                        : 197
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00727407
variance of residuals (reduced chisquare) = WSSR/ndf   : 5.2912e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 7.24627e-09      +/- 1.725e-11    (0.2381%)
b               = -1.06267e-05     +/- 3.58e-07     (3.369%)
c               = 0.0128075        +/- 0.001559     (12.17%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.969  1.000
c               0.749 -0.868  1.000
gnuplot> plot 'tiempos.dat' w l, f(x) w l
```



Finalmente, como dijimos teóricamente la gráfica refleja claramente que esta estructura en el peor de los casos conseguir los datos ordenados puede llegar a tener una eficiencia de N^2. Por ultimo en cuanto a estructuras jerárquicas voy a comprobar que el APO es capaz de conseguir los datos ordenados con una eficiencia de nlog(n), para ello he realizado los mismos pasos que con el ABB, pero en este caso da igual como inserte los datos que la eficiencia siempre es la misma, luego dejando claro esto dejo todo el procedimiento que he realizado:



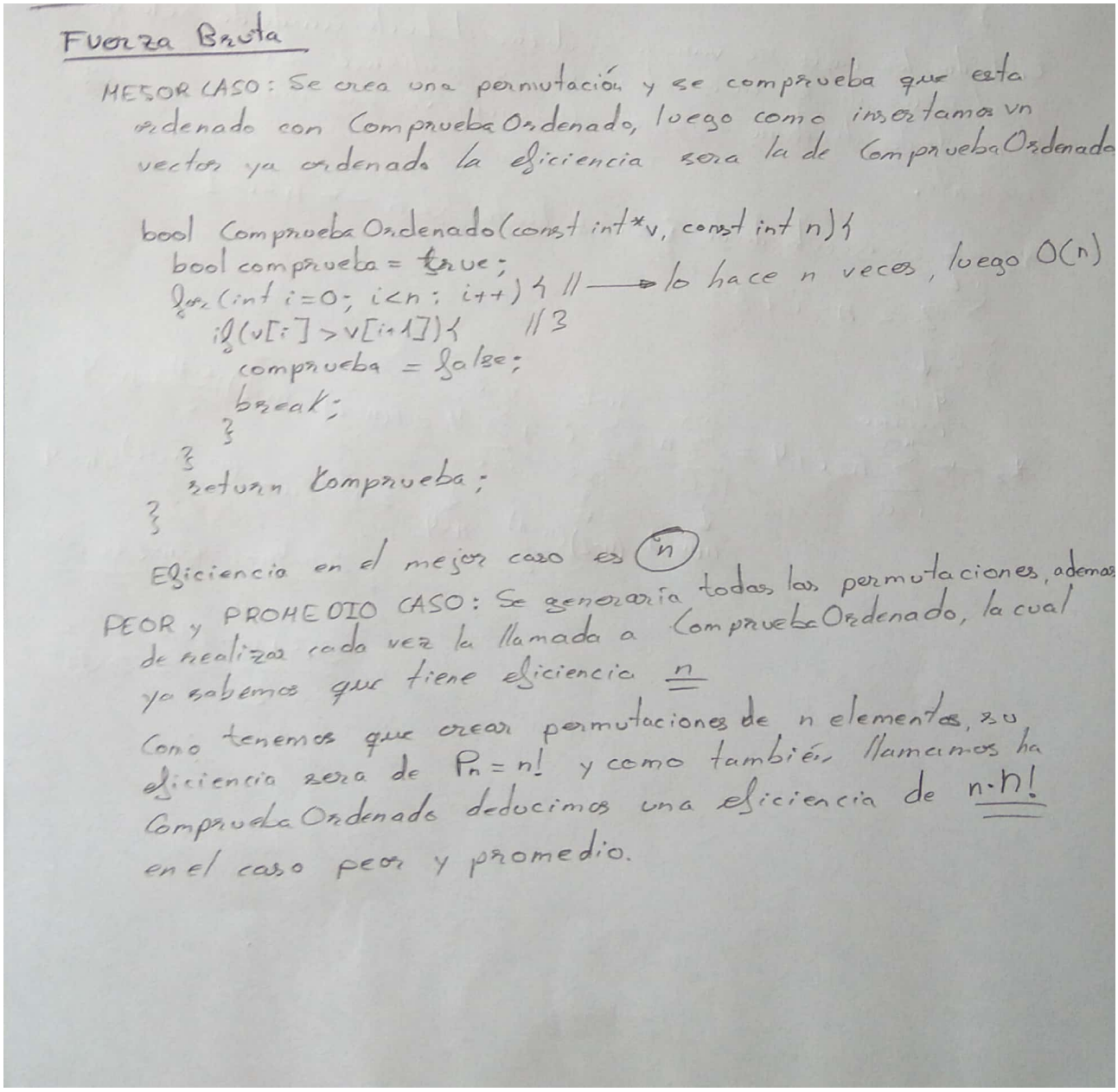

```
gnuplot> f(x)=a*x*log(b*x)
gnuplot> fit f(x) "APO_todosCaso.dat" via a,b
iter      chisq       delta/lim  lambda   a             b
   0 3.7649279881e+15   0.00e+00  9.74e+05    1.000000e+00   1.000000e+00
   1 1.8828578147e+11  -2.00e+09  9.74e+04    7.122945e-03   9.182927e-01
   2 1.6830690101e+00  -1.12e+16  9.74e+03    6.277857e-08   9.182880e-01
   3 4.9081678966e-03  -3.42e+07  9.74e+02    4.151357e-08   9.182880e-01
   4 4.9081678963e-03  -5.42e-06  9.74e+01    4.151357e-08   9.182880e-01
iter      chisq       delta/lim  lambda   a             b

After 4 iterations the fit converged.
final sum of squares of residuals : 0.00490817
rel. change during last iteration : -5.4248e-11

degrees of freedom    (FIT_NDF)                        : 1998
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00156734
variance of residuals (reduced chisquare) = WSSR/ndf   : 2.45654e-06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 4.15136e-08      +/- 9.102e-10    (2.193%)
b               = 0.918288         +/- 0.2375       (25.87%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -1.000  1.000
gnuplot> plot f(x) w l, 'APO_todosCaso.dat' w l
```



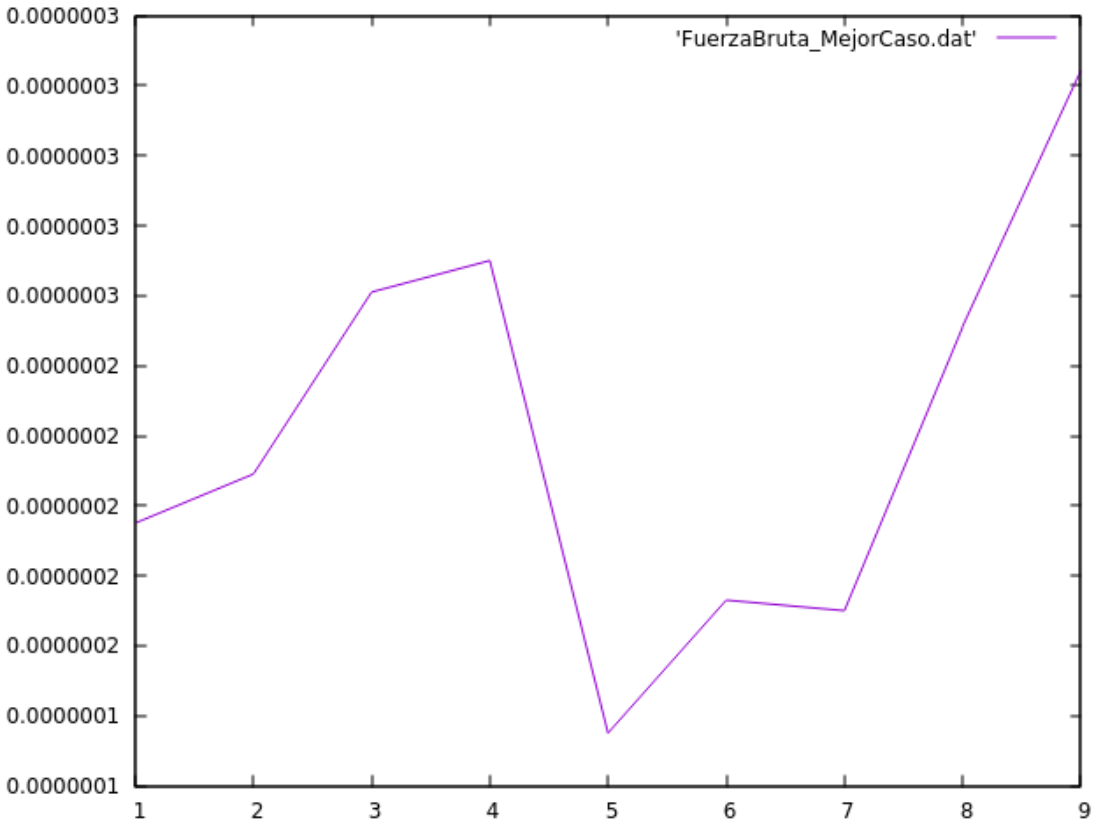
Como vemos igualmente llegamos a la misma conclusión que llegamos con la eficiencia teórica, luego deducimos que este algoritmo tiene una eficiencia en todos sus casos de nlog(n).

Algoritmo de Ordenación con Fuerza Bruta

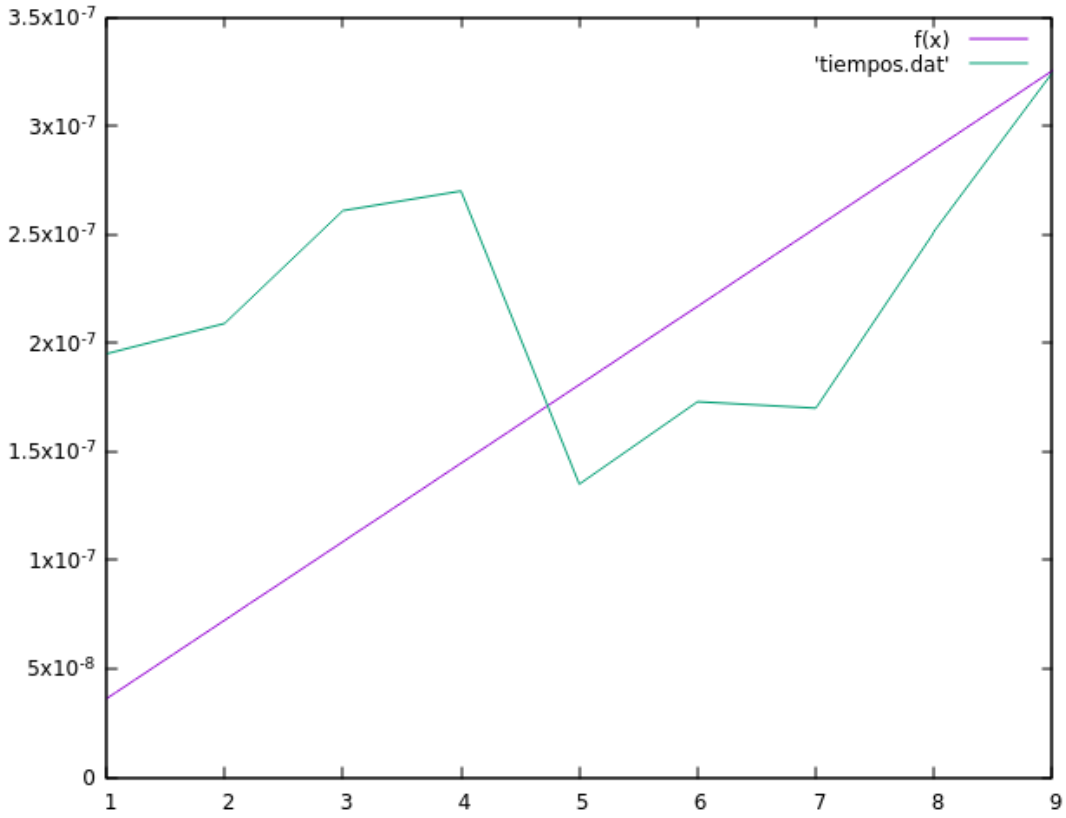
El algoritmo de ordenación por fuerza bruta consiste en coger todas las posibles combinaciones de n numero, es decir, si tenemos números de tres cifras podemos llegar a tener Pn=n!, es decir, que con tres cifras podremos realizar 1*2*3=6 posibles números. Este algoritmo va a consistir en sacar todas las permutaciones de n cifras y comprobar si están ordenados, si están ordenados terminamos si no seguimos creando permutaciones. Teniendo en cuenta esto, para llegar al mejor de los casos de este algoritmo no es más que insertar un vector ordenado que corresponda con la permutación que hemos creado, es decir si la primera permutación es 0 1 2, y el vector es 3 5 7, i=0 corresponde a 3, i=1 corresponde a 5 y i=2 corresponde a 7, como vemos que esta ordenado (lo comprobaríamos con una función) no se seguirían creando permutaciones, esto sería en el mejor de los casos. En el peor de los casos sería encontrar el vector ordenado en la ultima permutación, haciendo que el algoritmo cree todas las posibles permutaciones y encuentre el vector ordenado en la ultima permutación. En cuanto a la eficiencia si nos vamos al mejor de los casos, la eficiencia sería simplemente la que nos cueste la función de comprobación de si el vector esta o no ordenado, luego en mi caso esa función es de orden N, luego en el mejor de los casos la eficiencia del algoritmo es orden N. En el peor de los casos, se van a tener que realizar n! Permutaciones y n comprobaciones del orden del vector, luego la eficiencia en el peor de los casos y en el caso promedio es de N*N!. Dejo el análisis realizado teóricamente a continuación:



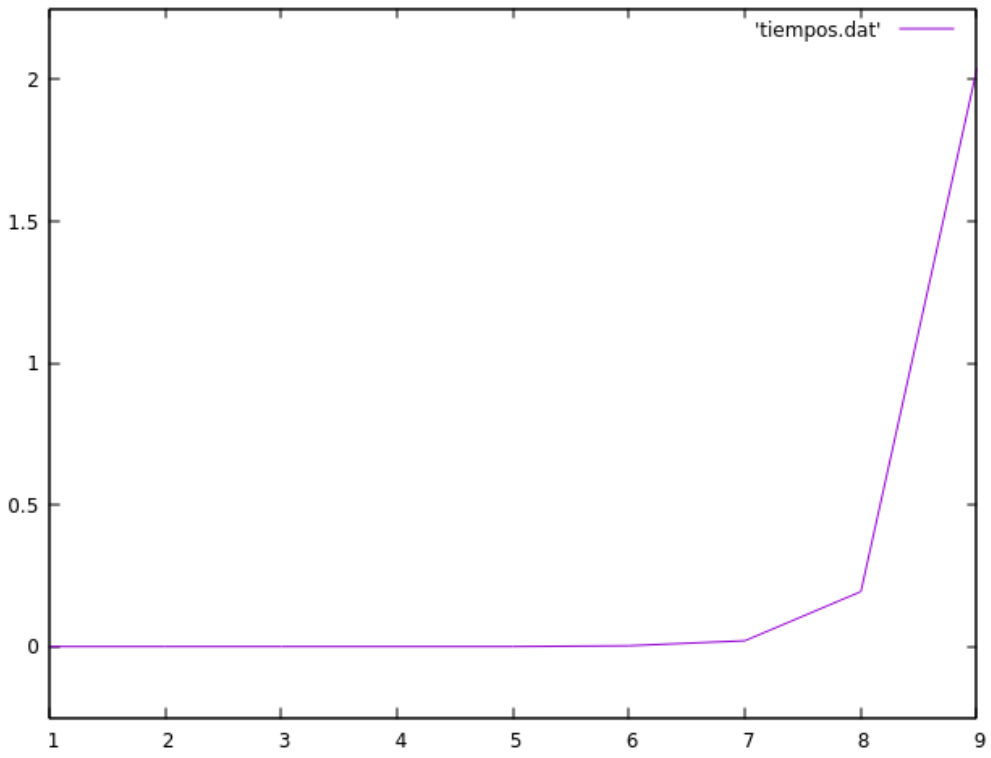
En cuanto a la eficiencia empírica, voy a dejar todo el procedimiento realizado primero del mejor de los casos, lance el programa creando una permutacion ya ordenada y llamando a la función de comprobación del orden una sola vez debido a que inserto el vector ordenado respecto a la permutación.



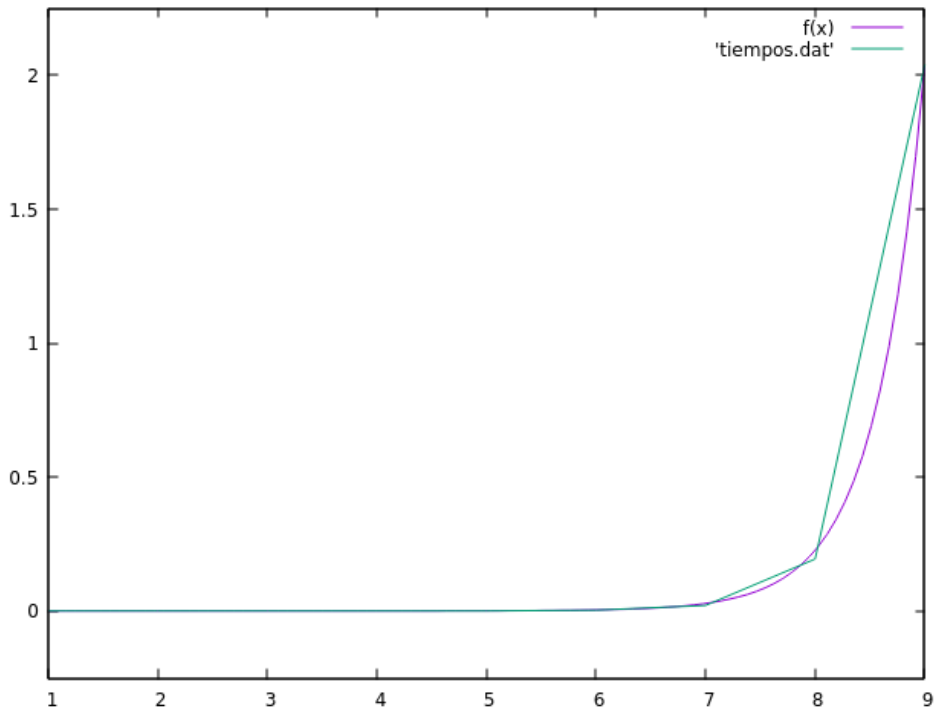

```
gnuplot> f(x)=a*x
gnuplot> fit f(x) "tiempos.dat" via a
iter      chisq       delta/lim  lambda   a
   0 2.8499997939e+02   0.00e+00  5.63e+00    1.000000e+00
   1 2.8499997939e+00  -9.90e+06  5.63e-01    1.000000e-01
   2 3.5107124416e-06  -8.12e+10  5.63e-02    1.110239e-04
   3 9.5728923915e-14  -3.67e+12  5.63e-03    3.738406e-08
   4 9.5295512281e-14  -4.55e+02  5.63e-04    3.615088e-08
   5 9.5295512281e-14   0.00e+00  5.63e-05    3.615088e-08
iter      chisq       delta/lim  lambda   a

After 5 iterations the fit converged.
final sum of squares of residuals : 9.52955e-14
rel. change during last iteration : 0

degrees of freedom    (FIT_NDF)                        : 8
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 1.09142e-07
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.19119e-14

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 3.61509e-08      +/- 6.465e-09    (17.88%)
gnuplot> plot f(x) w l, 'tiempos.dat' w l
```



Como no tenemos una gran cantidad de datos recogidos el ajuste no es exacto del todo, pero podemos ver como nuestros datos recopilados tienen una tendencia a ser de eficiencia N (constante), luego tiene sentido debido a que solo realizamos la comprobación del orden del vector. Ahora dejo el estudio empírico del peor y promedio caso:



```
gnuplot> f(x)=a*x*b*gamma(x)
gnuplot> fit f(x) "tiempos.dat" via a, b
iter      chisq       delta/lim  lambda   a             b
   0 1.3333203742e+11   0.00e+00  1.22e+05    1.000000e+00   1.000000e+00
   1 1.0230954871e+10  -1.20e+06  1.22e+04    5.263184e-01   5.263184e-01
   2 6.4454356661e+08  -1.49e+06  1.22e+03    2.636913e-01   2.636913e-01
   3 4.0290354774e+07  -1.50e+06  1.22e+02    1.318668e-01   1.318668e-01
   4 2.5165565418e+06  -1.50e+06  1.22e+01    6.595485e-02   6.595485e-02
   5 1.5687987766e+05  -1.50e+06  1.22e+00    3.301991e-02   3.301991e-02
   6 9.7044611615e+03  -1.52e+06  1.22e-01    1.659481e-02   1.659481e-02
   7 5.8209592856e+02  -1.57e+06  1.22e-02    8.466255e-03   8.466255e-03
   8 3.0915639434e+01  -1.78e+06  1.22e-03    4.564089e-03   4.564089e-03
   9 1.0335150468e+00  -2.89e+06  1.22e-04    2.895946e-03   2.895990e-03
  10 8.1924097553e-03  -1.25e+07  1.22e-05    2.396426e-03   2.434648e-03
  11 1.1115595095e-03  -6.37e+05  1.22e-06    2.312092e-03   2.424163e-03
   * 1.6557931866e+07   1.00e+05  1.22e-05   -1.007832e-01   1.105164e-01
   * 1.6669624988e-01   9.93e+04  1.22e-04    1.280949e-03   3.504905e-03
  12 1.1114569256e-03  -9.23e+00  1.22e-05    2.301591e-03   2.434790e-03
   * 1.1117886876e-03   2.98e+01  1.22e-04    2.340277e-03   2.393914e-03
  13 1.1114552653e-03  -1.49e-01  1.22e-05    2.302002e-03   2.434404e-03
iter      chisq       delta/lim  lambda   a             b

After 13 iterations the fit converged.
final sum of squares of residuals : 0.00111146
rel. change during last iteration : -1.49381e-06

degrees of freedom    (FIT_NDF)                        : 7
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0126008
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000158779

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 0.002302         +/- 5.863e+08    (2.547e+13%)
b               = 0.0024344        +/- 6.2e+08      (2.547e+13%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -1.000  1.000
gnuplot> plot f(x) w l, 'tiempos.dat' w l
```



Como vemos decidimos que la eficiencia teórica de este algoritmo era de N*N! y lo podemos ver perfectamente reflejado en la gráfica, luego tiene sentido debido a que creamos todas las permutaciones posibles, así como todas las veces se llama al método de comprobar el orden, luego finalmente podemos asegurar que nuestra eficiencia teórica está correcta.

Algoritmos de Ordenación por Mezcla

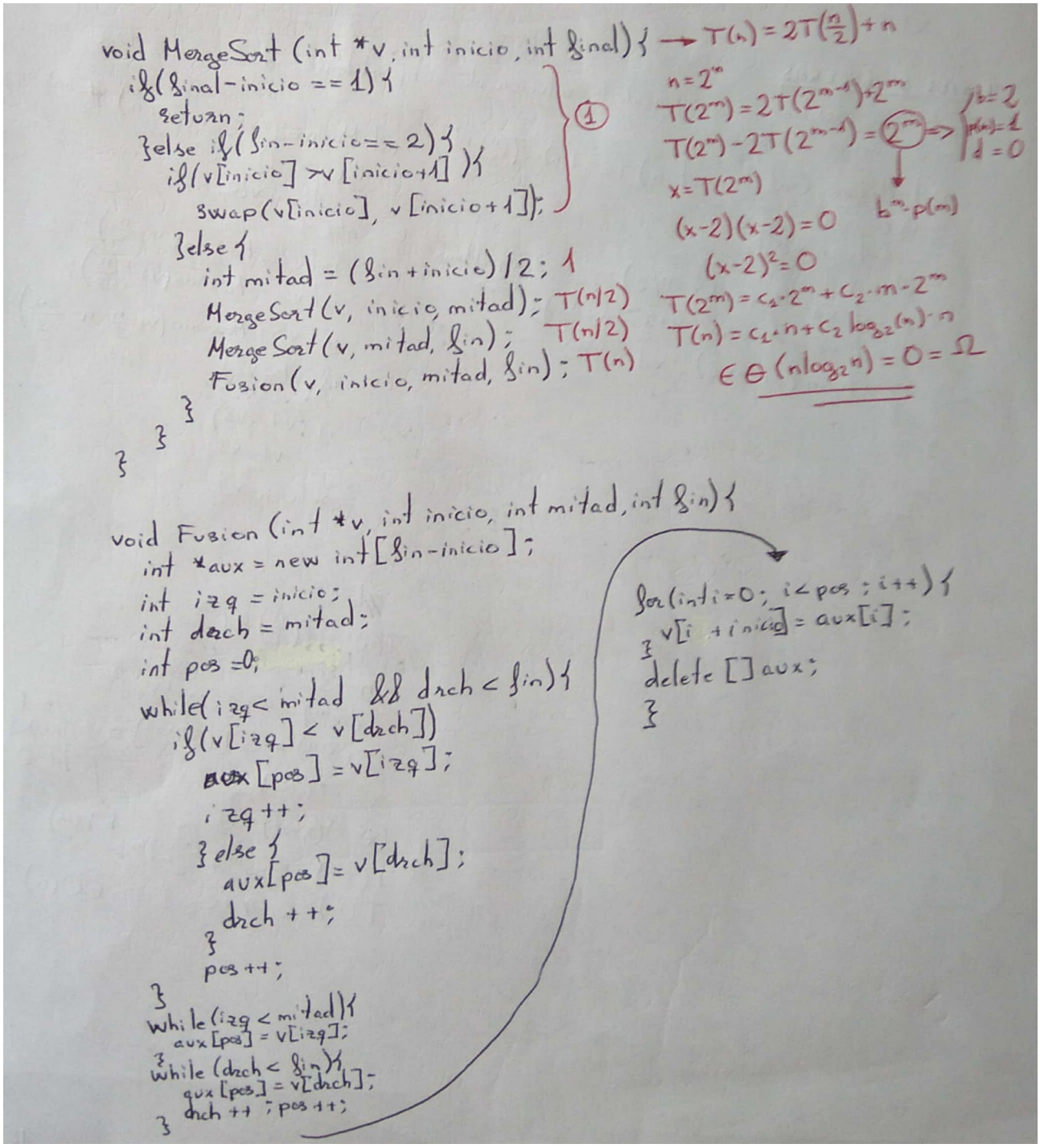
Se trata de un algoritmo basado en la técnica divide y vencerás, en el que su funcionamiento es el siguiente:

Si la longitud de la lista es 0 ó 1, entonces ya está ordenada. En otro caso:

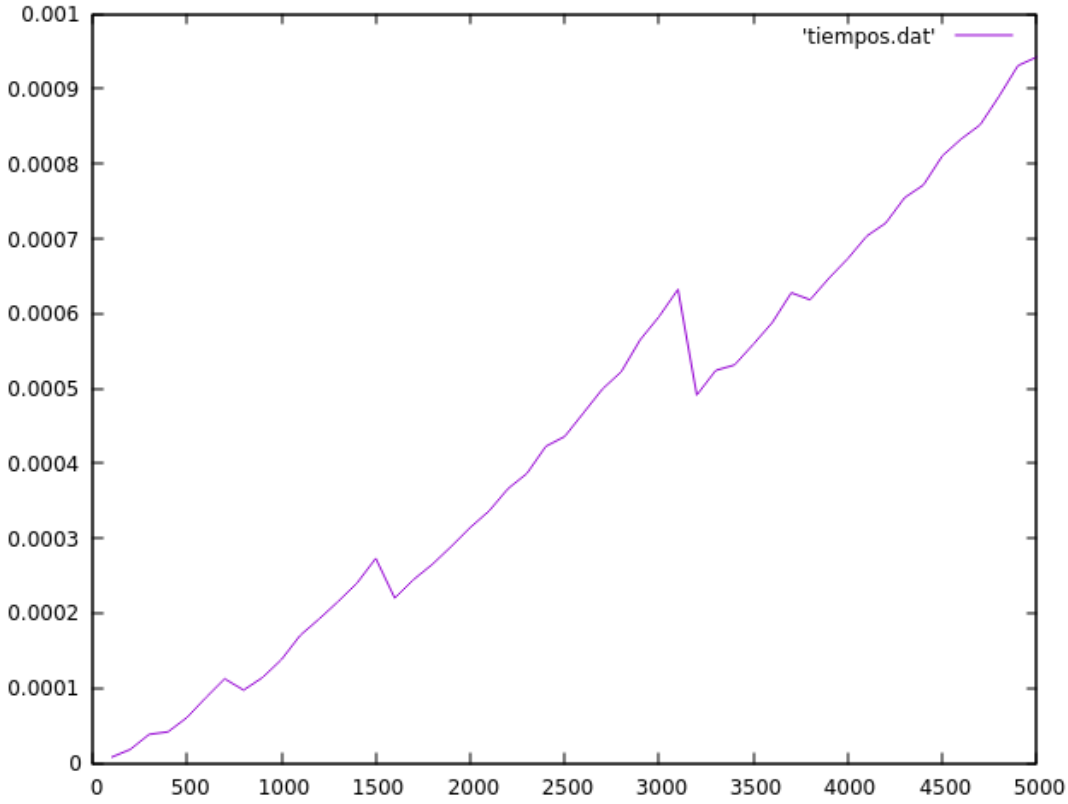
- Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.
-Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.

-Mezclar las dos sublistas en una sola lista ordenada.

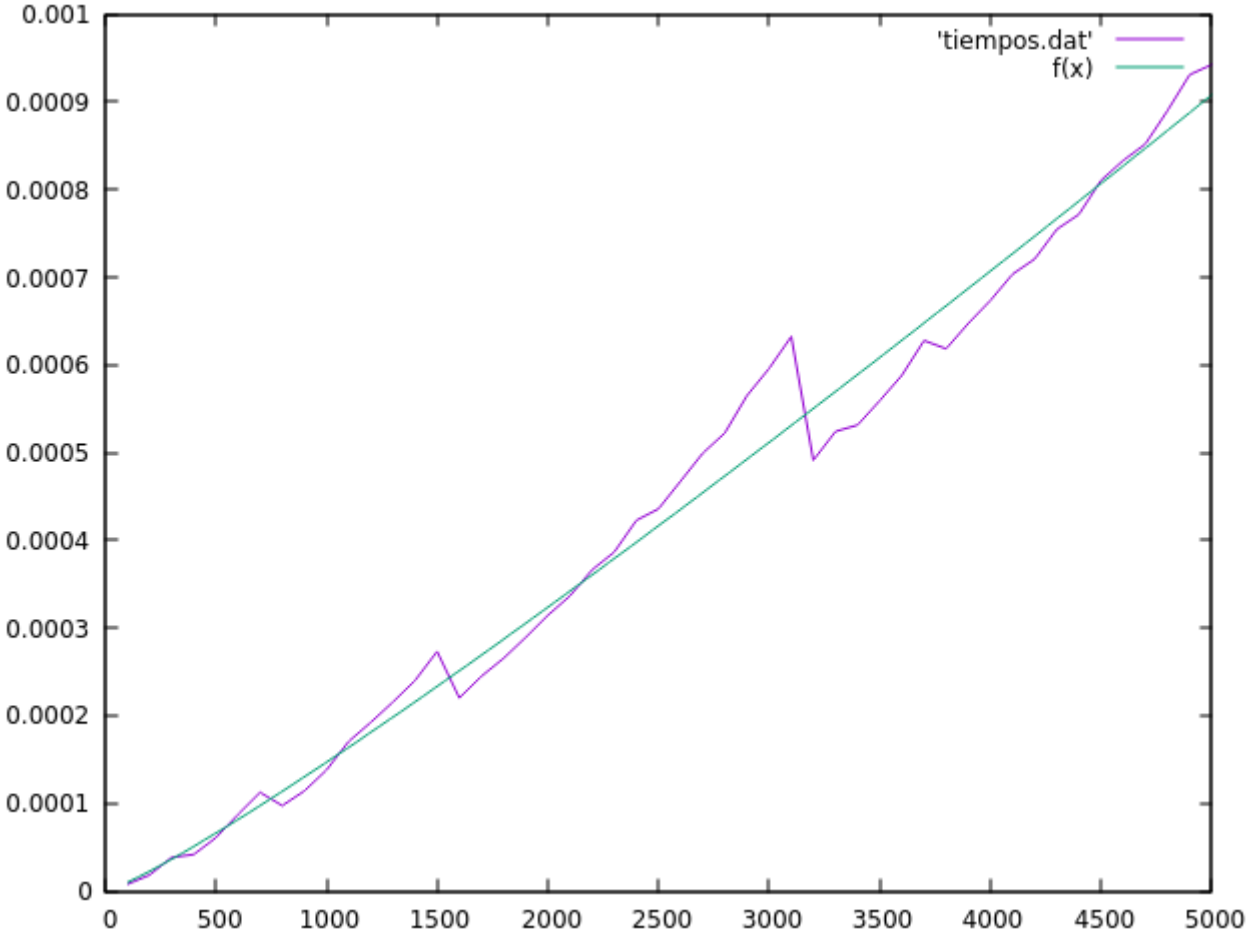
Este algoritmo en todos sus casos es de orden N*log(N), pero en este caso como lo hemos programado, tiene un valor llamado UMBRAL_MS que condiciona el tamaño mínimo del vector para utilizar el algoritmo MergeSort, ya que si el tamaño del vector es mayor que este valor se realizaría el ordenamiento del vector mediante MergeSort, en cambio, si el tamaño del vector es menor que el UMBRAL_MS el vector se ordenará mediante Inserción. En el ejercicio nos dice que comprobemos también usando los otros algoritmos estudiados, es decir, en vez de usar Inserción usar ABB, APO y Fuerza Bruta. En el caso de comparar graficando ABB y los tiempos del MergeSort no van a coincidir en ningún punto, puesto que la eficiencia de ambos algoritmos es prácticamente la misma, luego las dos gráficas nunca se cruzaran. En el caso del APO, pasa lo mismo ya que APO y MergeSort tienen la misma eficiencia de n*log(n) luego tampoco se cruzarían en ningún punto. En el caso de comparar Fuerza Bruta y MergeSort no he podido graficarlo puesto que con Fuerza Bruta tengo solo 9 posibles tamaños y con MergeSort tengo hasta 5000 datos, ya que este ultimo es más rápido y no podría ver ambos datos en una misma gráfica, como no se realizar esto ultimo he dejado este algoritmo fuera de mi estudio, pero al ser n*n! entiendo que no deberían cruzarse debido a que nlogn crece más rápido que n*n!. Voy a dejar la eficiencia teórica estudiada en clase del algoritmo MergeSort, la única diferencia entre este y el usado en la practica es el uso de UMBRAL_MS para la elección de un algoritmo u otro dependiendo del tamaño del vector.


void MergeSort (int *v, int inicio, int final){
if(final-inicio == 1){
return;
}else if(fin-inicio==2){
if(v[inicio] > v[inicio+1]){
swap(v[inicio], v[inicio+1]);
}else{
int mitad = (fin+inicio)/2; 1
MergeSort(v, inicio, mitad); T(n/2)
MergeSort(v, mitad, fin); T(n/2)
Fusion(v, inicio, mitad, fin); T(n)
}
}
}
T(n) = 2T(n/2) + n
n = 2^m
T(2^m) = 2T(2^{m-1}) + 2^m
T(2^m) - 2T(2^{m-1}) = 2^m
x = T(2^m)
(x-2)(x-2) = 0
(x-2)^2 = 0
T(2^m) = c1·2^m + c2·m·2^m
T(n) = c1·n + c2 log2(n)·n
∈ Θ(n log2 n) = O = Ω
void Fusion (int *v, int inicio, int mitad, int fin){
int *aux = new int[fin-inicio];
int izq = inicio;
int dcha = mitad;
int pos = 0;
while(izq < mitad && dcha < fin){
if(v[izq] < v[dcha])
aux[pos] = v[izq];
izq++;
}else{
aux[pos] = v[dcha];
dcha++;
}
pos++;
}
while(izq < mitad){
aux[pos] = v[izq];
}
while(dcha < fin){
aux[pos] = v[dcha];
dcha++; pos++;
}
for(int i=0; i<pos; i++){
v[i+inicio] = aux[i];
}
delete [] aux;
}

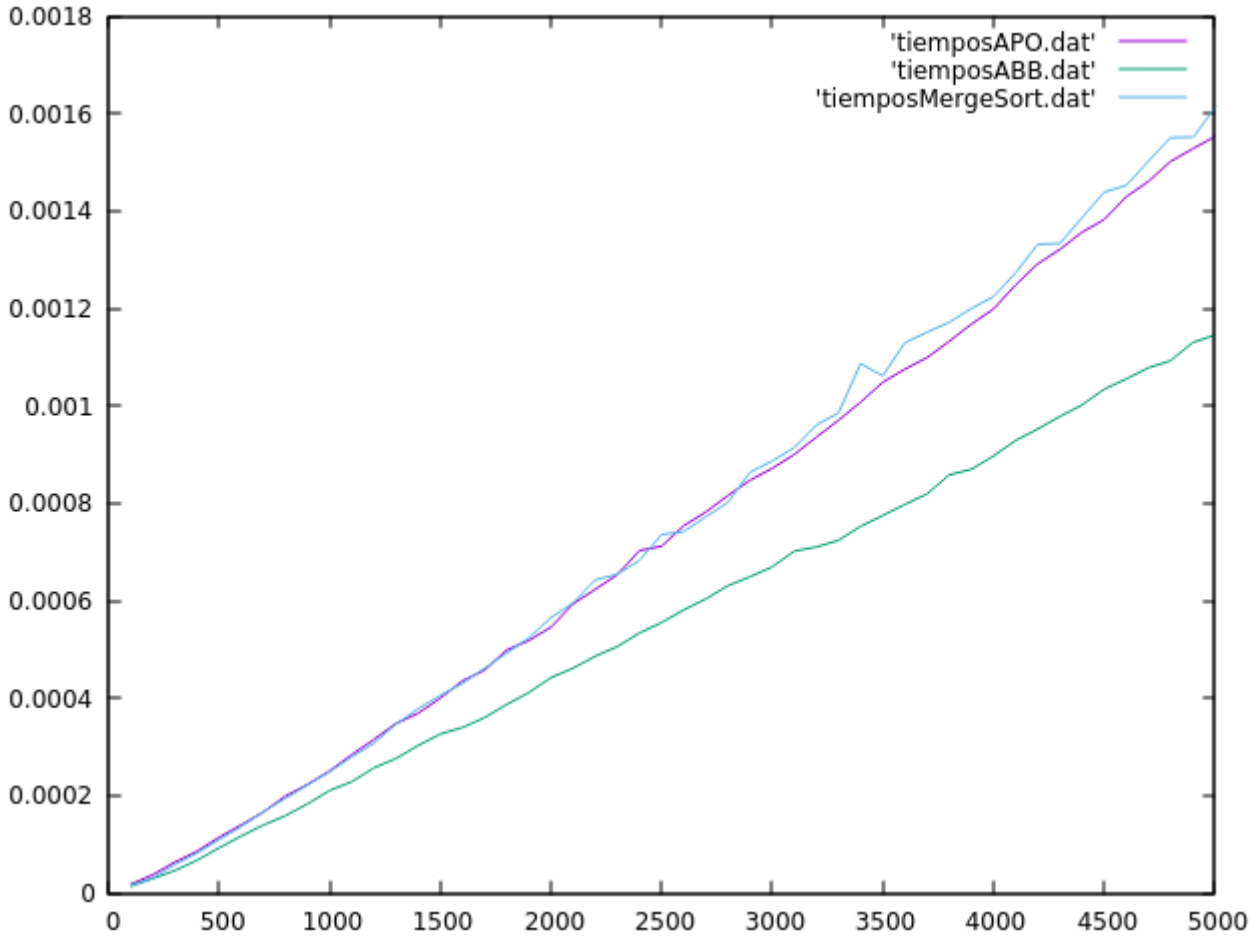
Como vemos el algoritmo MergeSort tiene una eficiencia de N*log(N) para todos sus posibles casos (mejor, peor y promedio), esto lo vamos a ver reflejado perfectamente en el estudio realizado empíricamente.




```
gnuplot> f(x)=a*x*log(b*x)
gnuplot> fit f(x) "tiempos.dat" via a,b
iter      chisq       delta/lim  lambda   a             b
   0 2.8866318671e+10   0.00e+00  1.71e+04    1.000000e+00   1.000000e+00
   1 1.6575092011e+07  -1.74e+08  1.71e+03    2.433664e-02   8.814682e-01
   2 2.2427535310e-01  -7.39e+12  1.71e+02    2.852564e-06   8.813852e-01
   3 5.8887667554e-08  -3.81e+11  1.71e+01    2.164391e-08   8.813852e-01
   4 5.8887421789e-08  -4.17e-01  1.71e+00    2.164094e-08   8.813852e-01
iter      chisq       delta/lim  lambda   a             b

After 4 iterations the fit converged.
final sum of squares of residuals : 5.88874e-08
rel. change during last iteration : -4.17348e-06

degrees of freedom    (FIT_NDF)                        : 48
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 3.5026e-05
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.22682e-09

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 2.16409e-08      +/- 5.073e-09    (23.44%)
b               = 0.881385         +/- 1.669        (189.4%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -0.999  1.000
gnuplot> plot 'tiempos.dat' w l, f(x) w l
```



Como vemos, se puede ver perfectamente el ajuste realizado a una eficiencia de orden N*log(N), luego podemos concluir que nuestra eficiencia teórica hace referencia a la empírica. Ahora voy a comparar MergeSort con el ABB y el APO, destacar, que adjunto los códigos de ambos y que para usar la ordenación con ABB o APO en vez de Inserción los he añadido en la misma parte del código donde se comprueba si el tamaño del vector es menor que el UMBRAL_MS, y he puesto el UMBRAL_MS con un valor de 6000 para que cuando yo lance el script con hasta 5000 datos de vector siempre se vaya al método del ABB o APO, ya que el UMBRAL_MS es mayor que el máximo tamaño de vector usado, esto nos va a dar gráficas como estas:



Como podemos observar y que ya he dicho con anterioridad, tanto usando APO como MergeSort los tiempos son prácticamente iguales debido a que ambos procedimientos en cualquier caso tienen una eficiencia de N*log(N). En el caso de usar MergeSort y ABB podemos ver que el ABB puede llegar a tener mejores tiempos de ejecución, pero la diferencia no es muy superior ya que la eficiencia del ABB en el mejor o caso promedio es de N*log(N), luego la diferencia es mínima entre este y MergeSort. Luego concluimos que UMBRAL_MS no es más que un valor que nos va a permitir dejar de usar MergeSort cuando lanzamos su código y esto afecta a la eficiencia del algoritmo debido a que si usamos en vez de Inserción por ejemplo un algoritmo todavía peor, aunque ya un algoritmo de N^2 es un algoritmo ineficiente de por si, podemos hacer que MergeSort ,que es un algoritmo relativamente rápido, pase a ser lento para ciertos tamaños de vector.