Universidad De Granada

Práctica 1

Análisis de la eficiencia de Algoritmos

Víctor José Rubia López
B3

Fecha de entrega 26/03/2020

· Contenido

Capítulo 1: Algoritmos de Ordenación

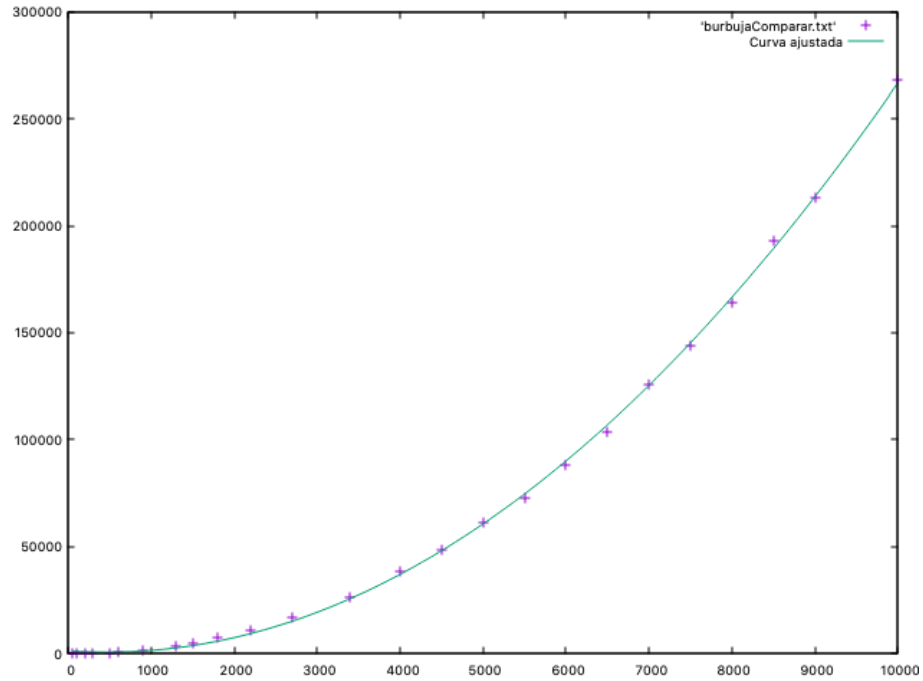
1.1. Algoritmo de la Burbuja

Este algoritmo es muy conocido por ser un método de ordenación sencillo y bastante eficiente. Para abordar el análisis, calcularemos su eficiencia empírica e híbrida, ejecutando el programa con distintos números de elementos del vector que es ordenado. Su orden de eficiencia es de $O(n^2)$.

Parámetros	Tiempo en µs
500	531
1000	2000
2000	9721
5000	61555
6000	90321
7000	124312
8000	160519
9000	210614
10000	265401
11000	322836
12500	433384
14000	535030
16000	712843
18000	906699
20000	1120631
50000	7327132
80000	18982947
90000	24127202
100000	29761368
110000	36295599
120000	43262337
125000	46863077
140000	60266036
150000	67773070
175000	89356080

1: Eficiencia empírica Algorítmo Burbuja

La tabla anterior *(Fig. 1)* nos muestra los tiempos de ejecución en microsegundos para los vectores de tamaño que se muestran en la columna *'Parámetros'*. Por lo tanto, podemos observar que los datos se pueden ajustar a una curva cuadrática, tal y como hemos calculado en la eficiencia híbrida.



2: Gráfica con el tiempo obtenido en función de la cantidad de parámetros y la curva ajustada

La curva ajustada *(Fig.2)* la hemos obtenido a través del uso de la herramienta *gnuplot* y según forma teórica debería ser cuadrática. A continuación expongo los pasos del ajuste obtenido:

```
gnuplot> fit f(x) 'burbujaComparar.txt' via a0,a1,a2
iter      chisq       delta/lim  lambda   a0             a1             a2
   0 3.6775877529e+16   0.00e+00  2.22e+07    1.000000e+00   1.000000e+00   1.000000e+00
   1 6.3675078174e+12  -5.77e+08  2.22e+06    1.560293e-02   9.998762e-01   1.000000e+00
   2 4.9318677664e+08  -1.29e+09  2.22e+05    2.479391e-03   9.998474e-01   1.000000e+00
   3 4.9234878502e+08  -1.70e+02  2.22e+04    2.477982e-03   9.971361e-01   9.999990e-01
   4 4.2826814476e+08  -1.50e+04  2.22e+03    2.509323e-03   7.474541e-01   9.999082e-01
   5 7.4570553145e+07  -4.74e+05  2.22e+02    2.836098e-03  -1.855797e+00   1.000675e+00
   6 7.0679758536e+07  -5.50e+03  2.22e+01    2.874097e-03  -2.158544e+00   1.191750e+00
   7 7.0326150077e+07  -5.03e+02  2.22e+00    2.874865e-03  -2.167208e+00   2.007361e+01
   8 5.9150039565e+07  -1.89e+04  2.22e-01    2.906828e-03  -2.534047e+00   8.534127e+02
   9 5.6446668620e+07  -4.79e+03  2.22e-02    2.931881e-03  -2.821583e+00   1.506598e+03
  10 5.6446502523e+07  -2.94e-01  2.22e-03    2.932078e-03  -2.823854e+00   1.511758e+03
iter      chisq       delta/lim  lambda   a0             a1             a2

After 10 iterations the fit converged.
final sum of squares of residuals : 5.64465e+07
rel. change during last iteration : -2.94256e-06

degrees of freedom    (FIT_NDF)                        : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 1601.8
variance of residuals (reduced chisquare) = WSSR/ndf   : 2.56575e+06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 0.00293208       +/- 4.037e-05    (1.377%)
a1              = -2.82385         +/- 0.3746       (13.26%)
a2              = 1511.76          +/- 641.3        (42.42%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.962  1.000
a2              0.609 -0.754  1.000
```

En este resultado vemos los valores, en negrita, de las constantes ocultas para la fórmula $f(x) = a_1x^2 + a_2x + a_3$ de modo que la función sería $f(x) = 0.00293208x^2 - 2.82385x + 1511.76$

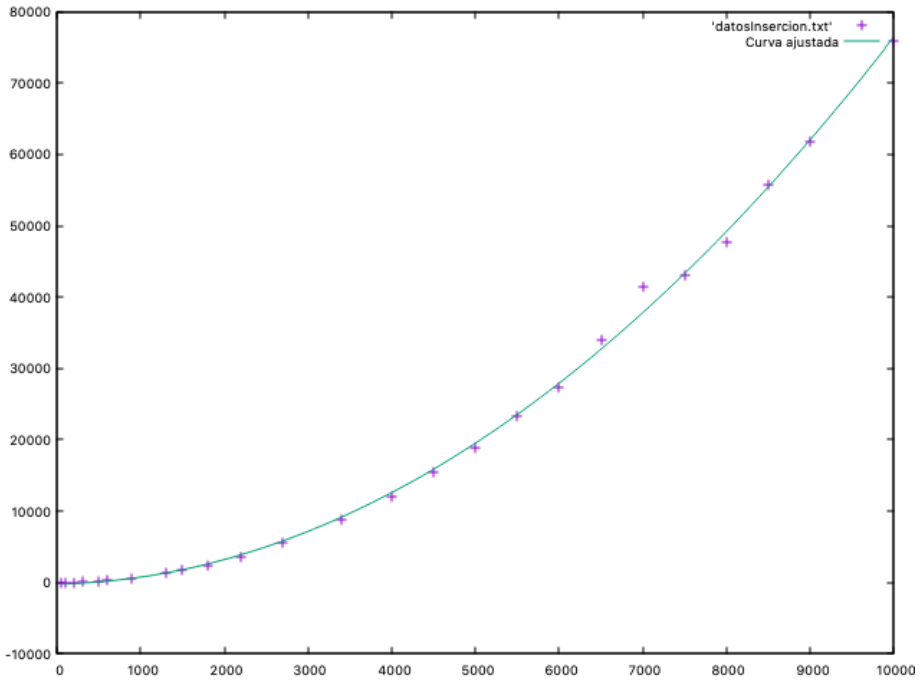
1.2 Algoritmo de Inserción

El ordenamiento a través de este algoritmo se realiza de una forma muy natural para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria. Requiere $O(n^2)$ operaciones para ordenar una lista de n elementos. A continuación, se expone la tabla que contiene el tiempo en microsegundos tardado en ejecutarse en función del número de parámetros.

Parámetros	Tiempo en µs
500	190
1000	765
2000	3038
5000	18711
6000	27434
7000	37546
8000	48666
9000	59956
10000	76671
11000	92715
12500	117333
14000	146678
16000	189645
18000	241778
20000	295452
50000	1840417
80000	4721107
90000	6027389
100000	7804981
110000	9420028
120000	11085752
125000	11798374
140000	15030902
150000	17162835
175000	23564005

3: Tabla que muestra los microsegundos de ejecución en función del número de parámetros

Tras obtener estos datos, expondré la gráfica que muestra el ajuste de la función cuadrática a los resultados empíricos, lo que mostrará también la eficiencia híbrida obtenida tras el cálculo de los coeficientes de la función.



3:Gráfica con el tiempo obtenido en función de la cantidad de parámetros y la curva ajustada

```
gnuplot> fit f(x) 'datosInsercion.txt' via a0,a1,a2
iter      chisq       delta/lim  lambda   a0             a1             a2
   0 1.2465677383e+11   0.00e+00  6.56e+04    2.932078e-03  -2.823854e+00   1.511758e+03
   1 2.8787865409e+08  -4.32e+07  6.56e+03    1.125341e-03  -2.852191e+00   1.520855e+03
   2 3.6463358036e+07  -6.90e+05  6.56e+02    8.357713e-04  -7.094614e-01   1.294261e+03
   3 1.9012764338e+07  -9.18e+04  6.56e+01    7.449997e-04   2.101463e-01  -1.671902e+02
   4 1.8998541653e+07  -7.49e+01  6.56e+00    7.430237e-04   2.323011e-01  -2.148255e+02
   5 1.8998541652e+07  -7.63e-06  6.56e-01    7.430231e-04   2.323081e-01  -2.148407e+02
iter      chisq       delta/lim  lambda   a0             a1             a2

After 5 iterations the fit converged.
final sum of squares of residuals : 1.89985e+07
rel. change during last iteration : -7.63463e-11

degrees of freedom    (FIT_NDF)                        : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 929.285
variance of residuals (reduced chisquare) = WSSR/ndf   : 863570

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 0.000743023      +/- 2.342e-05    (3.152%)
a1              = 0.232308         +/- 0.2173       (93.55%)
a2              = -214.841         +/- 372.1        (173.2%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.962  1.000
a2              0.609 -0.754  1.000
```

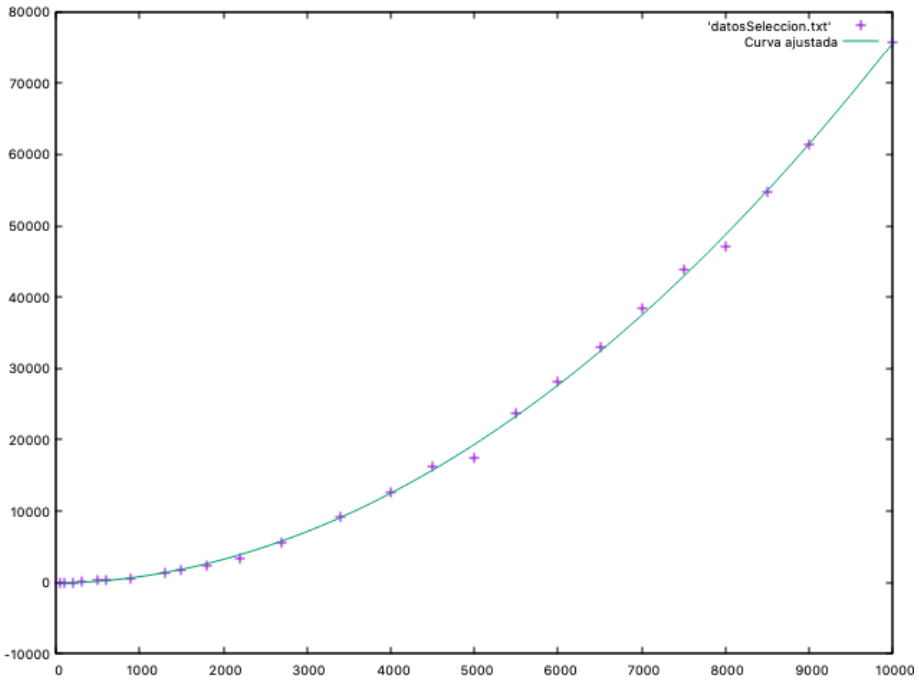
Por lo tanto, la función ajustada sería $f(x) = 0.000743023x^2 + 0.232308x - 214.841$.

1.3 Algoritmo de Selección

El ordenamiento por selección es conocido por su simpleza y por sus ventajas en rendimiento sobre otros algoritmos más complicados en situaciones concretas, particularmente cuando la memoria auxiliar es limitada. Requiere $O(n^2)$ operaciones para ordenar una lista de n elementos. A continuación, se expone la tabla que indica los microsegundos tardados en ordenar un vector tantos parámetros.

Parámetros	Tiempo en µs
500	226
1000	815
2000	3091
5000	19048
6000	28220
7000	38323
8000	47810
9000	61784
10000	76792
11000	90744
12500	118428
14000	155825
16000	198320
18000	249090
20000	302210
50000	1896105
80000	4803923
90000	6008031
100000	7316909
110000	9061501
120000	11030093
125000	12088259
140000	15388992
150000	17646921
175000	24016841

Tras obtener estos datos mostraremos a continuación la representación gráfica de los mismos al mismo tiempo que la curva ajustada tras obtener los coeficientes de la eficiencia híbrida.




```
gnuplot> fit f(x) 'datosSeleccion.txt' via a0,a1,a2
çiter      chisq       delta/lim  lambda   a0             a1             a2
   0 1.1737722551e+07   0.00e+00  1.65e+04    7.430231e-04   2.323081e-01  -2.148407e+02
   1 9.8671007902e+06  -1.90e+04  1.65e+03    7.359929e-04   2.323670e-01  -2.146985e+02
   2 9.8444057734e+06  -2.31e+02  1.65e+02    7.349985e-04   2.380915e-01  -2.039308e+02
   3 9.8013903397e+06  -4.39e+02  1.65e+01    7.369253e-04   2.128275e-01  -1.303094e+02
   4 9.8010001850e+06  -3.98e+00  1.65e+00    7.372646e-04   2.090659e-01  -1.224486e+02
   5 9.8010001845e+06  -5.00e-06  1.65e-01    7.372650e-04   2.090616e-01  -1.224398e+02
iter      chisq       delta/lim  lambda   a0             a1             a2

After 5 iterations the fit converged.
final sum of squares of residuals : 9.801e+06
rel. change during last iteration : -4.9956e-11

degrees of freedom    (FIT_NDF)                        : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 667.458
variance of residuals (reduced chisquare) = WSSR/ndf   : 445500

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 0.000737265      +/- 1.682e-05    (2.281%)
a1              = 0.209062         +/- 0.1561       (74.66%)
a2              = -122.44          +/- 267.2        (218.3%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.962  1.000
a2              0.609 -0.754  1.000
```

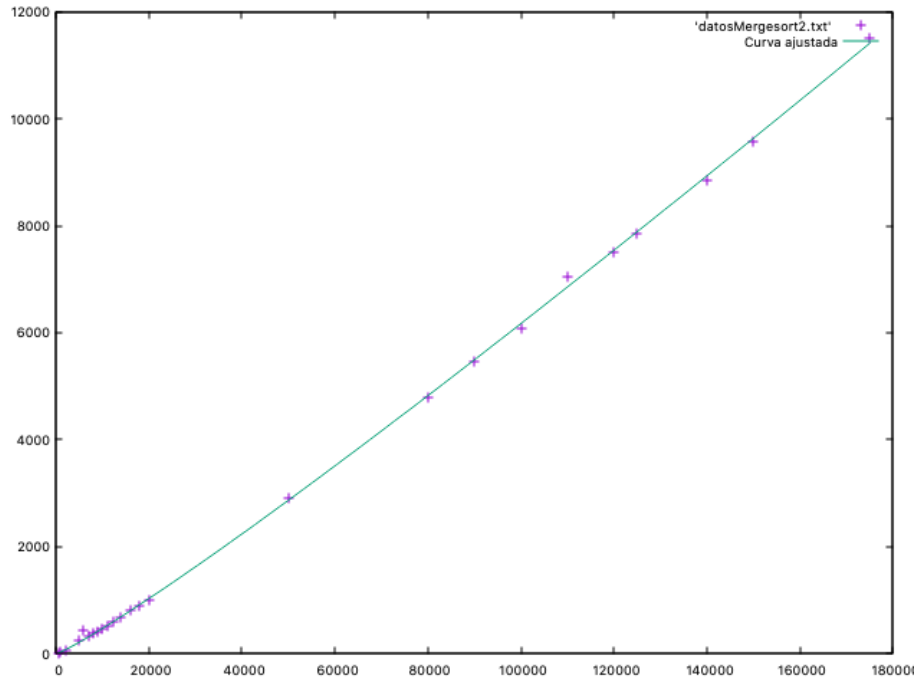
Por lo tanto, nuestra función ajustada sería $f(x) = 0.000737265x^2 + 0.209062x - 122.44$

1.4 Algoritmo Mergesort

Este algoritmo está basado en la técnica divide y vencerás. Es de complejidad O(n log n). A continuación, se expone la tabla con el tiempo en microsegundos tardado en ordenar un vector de tamaño específico.

Parámetros	Tiempo en µs
500	64
1000	139
2000	312
5000	667
6000	845
7000	841
8000	974
9000	1090
10000	1273
11000	1450
12500	2180
14000	1662
16000	1915
18000	2343
20000	3031
50000	7662
80000	12162
90000	14184
100000	16024
110000	15142
120000	16589
125000	17839
500000	82425
1000000	167854
1500000	284813

Expondremos a continuación, la gráfica con los datos de la tabla junto con la curva ajustada por la herramienta usada *gnuplot* a través de las constantes ocultas.




```
gnuplot> f(x)=c1*x+c2*x*(log(x)/log(2))
gnuplot> fit f(x) 'datosMergesort2.txt' via c1,c2
iter      chisq       delta/lim  lambda   c1             c2
   0 4.5932155038e+13   0.00e+00  9.10e+05    1.000000e+00   1.000000e+00
   1 1.7778975592e+10  -2.58e+08  9.10e+04    9.390309e-01  -3.097470e-02
   2 1.1575887840e+08  -1.53e+07  9.10e+03    9.228884e-01  -5.070981e-02
   3 1.7207489208e+07  -5.73e+05  9.10e+02    3.488470e-01  -1.685141e-02
   4 1.0223084197e+05  -1.67e+07  9.10e+01   -7.649063e-03   4.178195e-03
   5 1.0157110664e+05  -6.50e+02  9.10e+00   -9.876702e-03   4.309603e-03
   6 1.0157110664e+05  -2.54e-06  9.10e-01   -9.876841e-03   4.309611e-03
iter      chisq       delta/lim  lambda   c1             c2

After 6 iterations the fit converged.
final sum of squares of residuals : 101571
rel. change during last iteration : -2.53629e-11

degrees of freedom    (FIT_NDF)                        : 23
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 66.454
variance of residuals (reduced chisquare) = WSSR/ndf   : 4416.14

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c1              = -0.00987684      +/- 0.005764     (58.36%)
c2              = 0.00430961       +/- 0.0003402    (7.893%)

correlation matrix of the fit parameters:
                c1     c2
c1              1.000
c2             -1.000  1.000
```

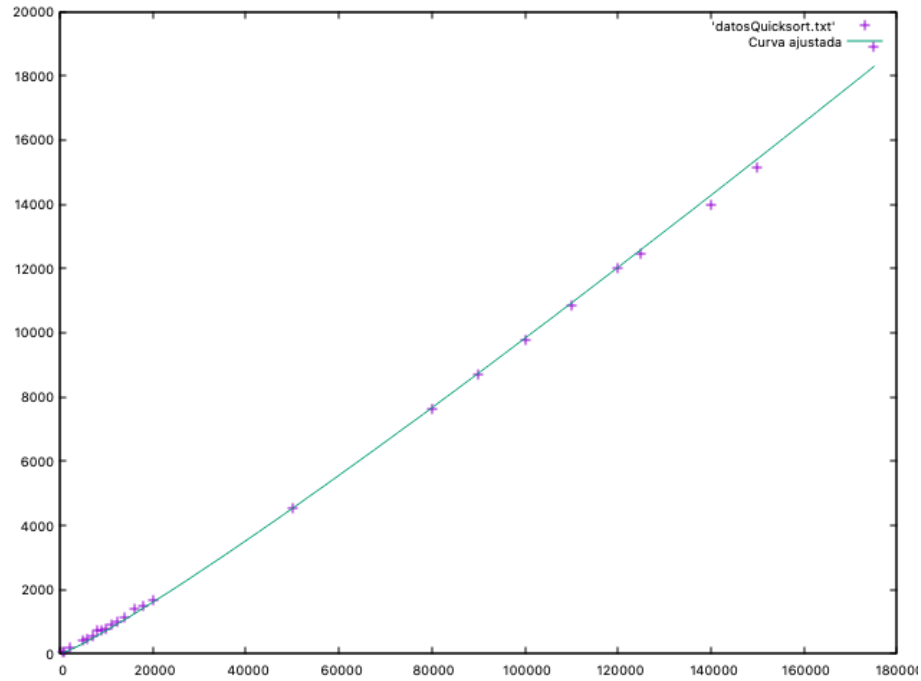
Por lo tanto, la función de forma $f(n) = c_1 n^2 + c_2 n\, log_2(n)$ de modo que la función sería $f(n) = -0.00987684n^2 + 0.00430961n\, log_2(n)$.

1.5 Algoritmo Quicksort

Este algoritmo también se basa en el principio de divide y vencerás y es recursivo. Selecciona un elemento del vector como "pivote" y parte los demás elementos en dos sub-vectores, en función de si son más grandes o pequeños que el pivote. Este algoritmo es de orden O(n log n). A continuación, expongo una tabla con pruebas de tiempo en microsegundos de ejecución para vectores de tanto tamaño.

Parámetros	Tiempo en µs
500	30
1000	63
2000	140
5000	382
6000	464
7000	544
8000	637
9000	721
10000	805
11000	903
12500	1013
14000	1173
16000	1356
18000	1587
20000	1717
50000	4697
80000	8163
90000	9000
100000	9679
110000	11730
120000	12069
125000	12972
500000	55319
1000000	114907
1500000	178243

Exponemos, por lo tanto, la gráfica con los valores representados y la curva aproximada tras haber calculado la eficiencia híbrida.




```
gnuplot> fit f(x) 'datosQuicksort.txt' via c1,c2
iter      chisq       delta/lim  lambda   c1             c2
   0 2.0497427736e+08   0.00e+00  3.95e+03   -9.876841e-03   4.309611e-03
   1 8.2671437579e+05  -2.47e+07  3.95e+02   -9.229082e-03   6.452914e-03
   2 7.4074239560e+05  -1.16e+04  3.95e+01   -1.069836e-02   6.583209e-03
   3 6.9741146592e+05  -6.21e+03  3.95e+00   -2.687905e-02   7.537725e-03
   4 6.9688909893e+05  -7.50e+01  3.95e-01   -2.885895e-02   7.654521e-03
   5 6.9688909815e+05  -1.12e-04  3.95e-02   -2.886138e-02   7.654664e-03
iter      chisq       delta/lim  lambda   c1             c2

After 5 iterations the fit converged.
final sum of squares of residuals : 696889
rel. change during last iteration : -1.12228e-09

degrees of freedom    (FIT_NDF)                        : 23
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 174.068
variance of residuals (reduced chisquare) = WSSR/ndf   : 30299.5

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c1              = -0.0288614       +/- 0.0151       (52.31%)
c2              = 0.00765466       +/- 0.000891     (11.64%)

correlation matrix of the fit parameters:
                c1     c2
c1              1.000
c2             -1.000  1.000
```

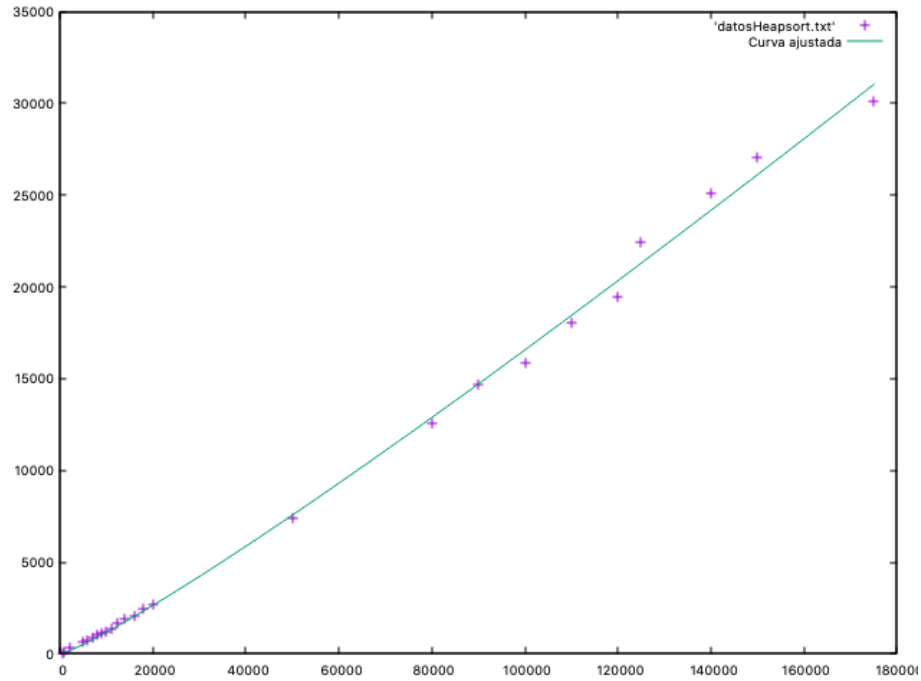
Por lo tanto la función ajustada sería $f(n) = -0.288614n^2 + 0.00765466n\,log_2(n)$.

1.6 Algoritmo Heapsort

Se puede pensar que su funcionamiento es mejorar el algoritmo de selección. Se diferencia por no malgastar tiempo al escanear con un tiempo linear la región no ordenada, manteniendo la zona desordenada en una estructura de datos *"Heap"* para encontrar rápidamente el elemento mayor en cada paso. A continuación, se expone la tabla con los tiempos de ejecución para cada tamaño muestreado.

Parámetros	Tiempo en µs
500	64
1000	136
2000	299
5000	824
6000	729
7000	864
8000	999
9000	1137
10000	1282
11000	1434
12500	1637
14000	1858
16000	2151
18000	2446
20000	2749
50000	7544
80000	12791
90000	14586
100000	16244
110000	18069
120000	19458
125000	21343
500000	95373
1000000	204930
1500000	320141

A su vez, exponemos la gráfica con los valores anteriormente expuestos en la tabla, con su curva ajustada por eficiencia híbrida.




```
gnuplot> fit f(x) 'datosHeapsort.txt' via c1,c2
iter      chisq       delta/lim  lambda   c1             c2
   0 7.0292712508e+08   0.00e+00  7.12e+03   -2.886138e-02   7.654664e-03
   1 6.0875173235e+06  -1.14e+07  7.12e+02   -2.573966e-02   1.150095e-02
   2 5.7246304721e+06  -6.34e+03  7.12e+01   -3.387349e-02   1.206137e-02
   3 5.5185182668e+06  -3.73e+03  7.12e+00   -7.148328e-02   1.428007e-02
   4 5.5180835766e+06  -7.88e+00  7.12e-01   -7.329073e-02   1.438669e-02
   5 5.5180835765e+06  -1.82e-06  7.12e-02   -7.329160e-02   1.438675e-02
iter      chisq       delta/lim  lambda   c1             c2

After 5 iterations the fit converged.
final sum of squares of residuals : 5.51808e+06
rel. change during last iteration : -1.81917e-11

degrees of freedom    (FIT_NDF)                        : 23
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 489.813
variance of residuals (reduced chisquare) = WSSR/ndf   : 239917

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c1              = -0.0732916       +/- 0.04248      (57.96%)
c2              = 0.0143867        +/- 0.002507     (17.43%)

correlation matrix of the fit parameters:
                c1     c2
c1              1.000
c2             -1.000  1.000
```

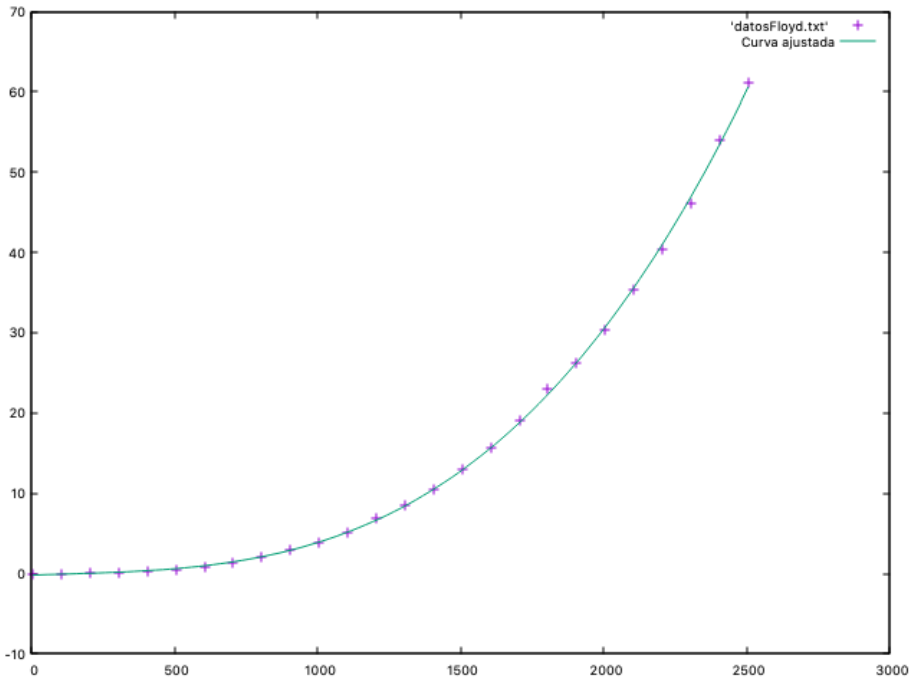
Por lo tanto, la función ajustada obtenida para la eficiencia híbrida sería $f(n) = -0.0732916n^2 + 0.0143867n\,log_2(n)$.

Capítulo 2: Algoritmo de Floyd

Este capítulo analiza la eficiencia del algoritmo de Floyd, creado para calcular el costo del camino mínimo entre cada par de nodos de un grafo dirigido. Exponemos la tabla con el tiempo tardado en microsegundos para tantos números de nodos.

Parámetros	Tiempo en µs
5	2,00E-01
10	4,42E+03
15	3,31E+04
20	1,09E+05
25	2,52E-01
30	4,67E+04
35	7,95E+04
40	1,26E+05
45	1,89E+05
50	2,67E+05
55	3,64E+05
60	4,89E+05
65	6,91E+05
70	8,57E+05
75	1,03E+06
80	1,28E+06
85	1,58E+06
90	1,79E+06
95	2,14E+06
100	2,51E+06
105	3,06E+06
110	3,50E+06
115	4,17E+06
120	4,57E+06
125	5,18E+06

A continuación, se expone la gráfica con los valores anteriores y el estudio de la eficiencia híbrida, junto a su curva ajustada a dichos puntos.




```
gnuplot> f(x)=a1*x*x*x+a2*x*x+a3*x+a4
gnuplot> fit f(x) 'datosFloyd.txt' via a1,a2,a3,a4
iter      chisq       delta/lim  lambda   a1            a2            a3            a4
   0 1.2647734436e+16   0.00e+00  9.71e+06    3.076000e-03   1.000000e+00   1.000000e+00   1.000000e+00
   1 4.8551227403e+12  -2.60e+08  9.71e+05   -3.257089e-04   7.854963e-01   9.998637e-01   9.999999e-01
   2 7.0621659753e+10  -6.77e+06  9.71e+04   -4.921857e-05   1.071852e-01   9.990562e-01   9.999991e-01
   3 7.3482148286e+05  -9.61e+09  9.71e+03    2.473093e-07  -1.002582e-03   9.988685e-01   9.999988e-01
   4 5.4787194788e+05  -3.41e+04  9.71e+02    3.246369e-07  -1.168947e-03   9.930260e-01   9.999817e-01
   5 2.1753028789e+05  -1.52e+05  9.71e+01    2.058957e-07  -7.360947e-04   6.248976e-01   9.989002e-01
   6 6.6503809321e+01  -3.27e+08  9.71e+00    7.004525e-09  -1.107123e-05   8.290744e-03   9.968316e-01
   7 5.3511183228e+00  -1.14e+06  9.71e-01    3.630375e-09   1.218832e-06  -2.136445e-03   9.712027e-01
   8 2.8237133426e+00  -8.95e+04  9.71e-02    4.048277e-09  -6.110798e-07   2.023050e-04   1.725323e-01
   9 2.5511522544e+00  -1.07e+04  9.71e-03    4.237199e-09  -1.438141e-06   1.258939e-03  -1.880264e-01
  10 2.5511466993e+00  -2.18e-01  9.71e-04    4.238056e-09  -1.441892e-06   1.263730e-03  -1.896615e-01
iter      chisq       delta/lim  lambda   a1            a2            a3            a4

After 10 iterations the fit converged.
final sum of squares of residuals : 2.55115
rel. change during last iteration : -2.17748e-06

degrees of freedom    (FIT_NDF)                        : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.340531
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.115961

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a1              = 4.23806e-09      +/- 2.032e-10    (4.794%)
a2              = -1.44189e-06     +/- 7.764e-07    (53.85%)
a3              = 0.00126373       +/- 0.00083      (65.68%)
a4              = -0.189661        +/- 0.2362       (124.6%)

correlation matrix of the fit parameters:
                a1     a2     a3     a4
a1              1.000
a2             -0.985  1.000
a3              0.909 -0.965  1.000
a4             -0.609  0.698 -0.834  1.000
```

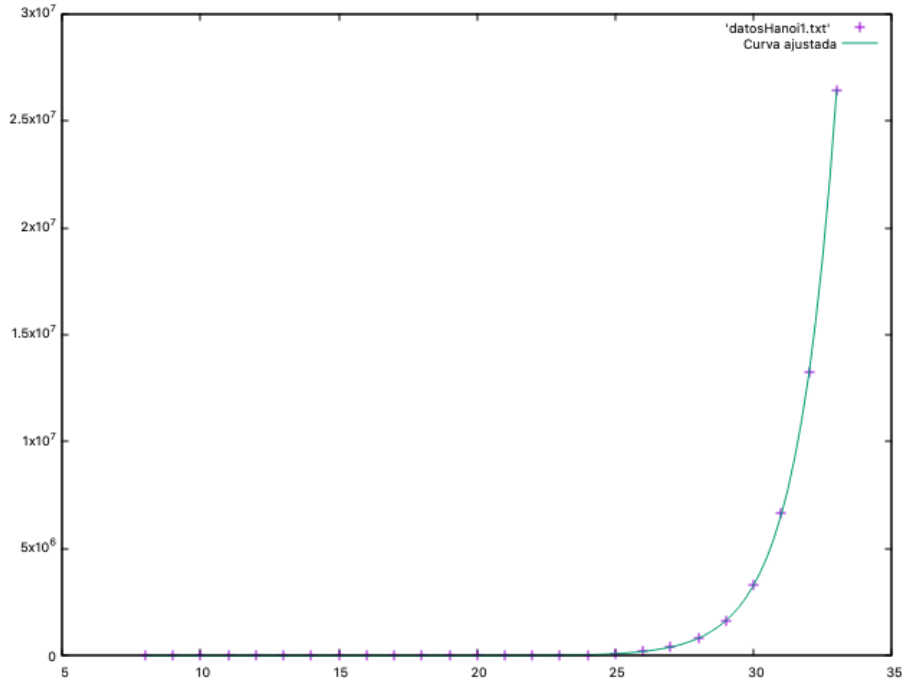
Por lo que la función del algoritmo para eficiencia híbrida sería $f(x) = a_1x^3 + a_2x^2 + a_3x + a_4$ y su ajustada sería $f(x) = 4.23806 * 10^{-9}x^3 - 1.44189 * 10^{-6}x^2 + 0.00126373x - 0.189661$.

Capítulo 3: Algoritmo de Hanoi

En este capítulo se aborda el algoritmo de las torres de Hanoi. Tras haberlo ejecutado, hemos conseguido los siguientes tiempos de ejecución en microsegundos.

Parámetros	Tiempo en µs
8	0
9	1
10	3
11	6
12	12
13	24
14	49
15	99
16	239
17	395
18	793
19	1636
20	3167
21	6398
22	12723
23	26567
24	51675
25	103899
26	208695
27	416331
28	829783
29	1641014
30	3302754
31	6597878
32	12954683
33	25724709

Posteriormente, encontraremos la gráfica que representa estos valores, junto a la curva ajustada a la función teórica.



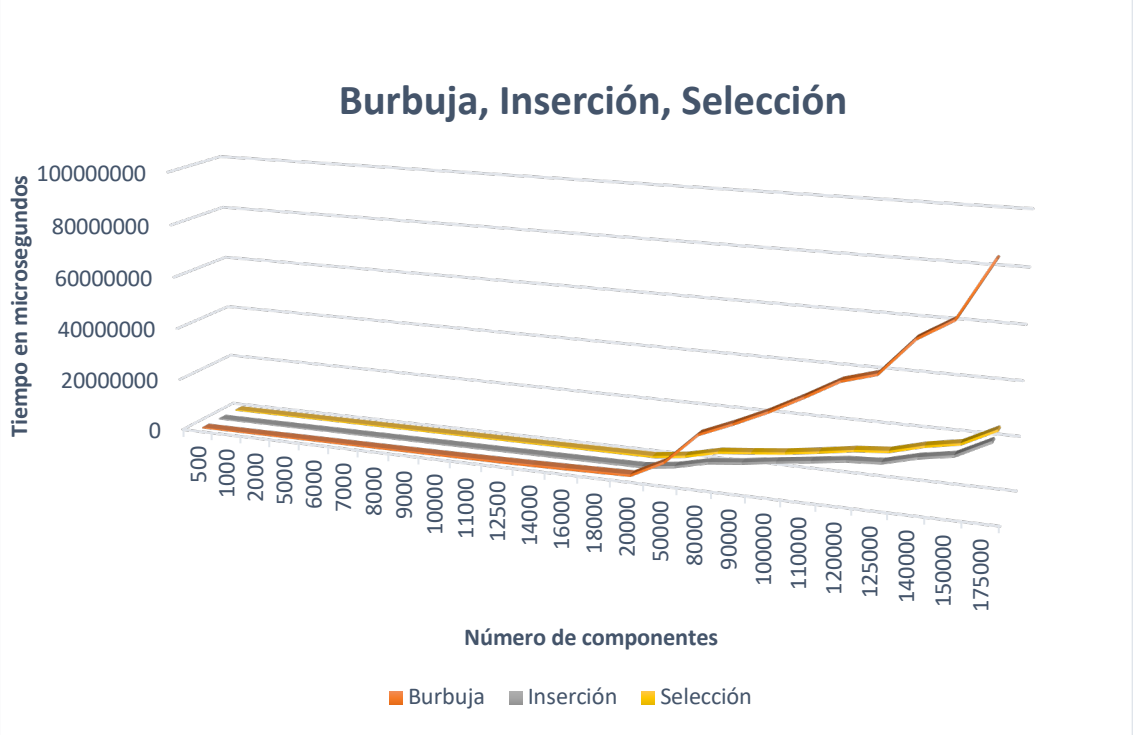

```
gnuplot> f(x)=(2**x)*a1+a2
gnuplot> fit f(x) 'datosHanoi1.txt' via a1,a2
iter      chisq       delta/lim  lambda   a1             a2
   0 9.7778315890e+19   0.00e+00  1.38e+09    1.000000e+00   1.000000e+00
   1 3.4808948961e+16  -2.81e+08  1.38e+08    2.188589e-02   1.000000e+00
   2 9.0296812793e+09  -3.85e+11  1.38e+07    3.079617e-03   1.000000e+00
   3 7.7428633425e+09  -1.66e+04  1.38e+06    3.076000e-03   1.000000e+00
   4 7.7428633425e+09  -2.66e-07  1.38e+05    3.076000e-03   1.000000e+00
iter      chisq       delta/lim  lambda   a1             a2

After 4 iterations the fit converged.
final sum of squares of residuals : 7.74286e+09
rel. change during last iteration : -2.65908e-12

degrees of freedom    (FIT_NDF)                        : 24
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 17961.6
variance of residuals (reduced chisquare) = WSSR/ndf   : 3.22619e+08

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a1              = 0.003076         +/- 1.925e-06    (0.06259%)
a2              = 1                +/- 3745         (3.745e+05%)

correlation matrix of the fit parameters:
                a1     a2
a1              1.000
a2             -0.340  1.000
```

La función teórica para este algoritmo sería $f(x) = 2^n a_1 + a_2$, por lo que la función con las componentes ocultas sería $f(x) = 2^n 0.003076 + 1$

Capítulo 4: Entendiendo los resultados

4.1 Algoritmos con eficiencia $O(n^2)$

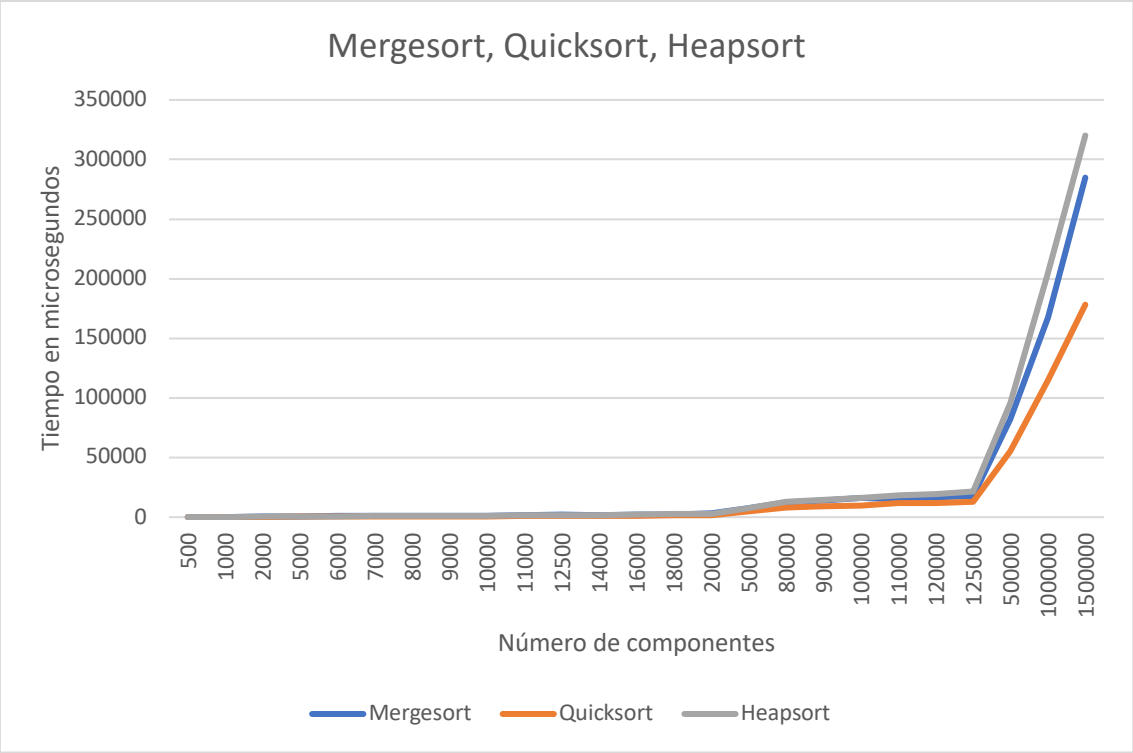
Para comprender los resultados obtenidos en los algoritmos **burbuja**, **inserción** y **selección** nos ayudamos de la composición de las tres gráficas



Así podemos ver fácilmente cómo el algoritmo de la burbuja, inserción y selección son equiparables en términos de eficiencia para vectores con menos de 20.000 elementos. Sin embargo, para vectores con mayores componentes vemos cómo el algoritmo de la burbuja incrementa su tiempo considerablemente más rápido que inserción y selección. Además, se aprecia cómo selección e inserción son dos algoritmos que son muy similares en términos de eficiencia para vectores con muchas componentes.

4.2 Algoritmos con eficiencia $O(n \log n)$

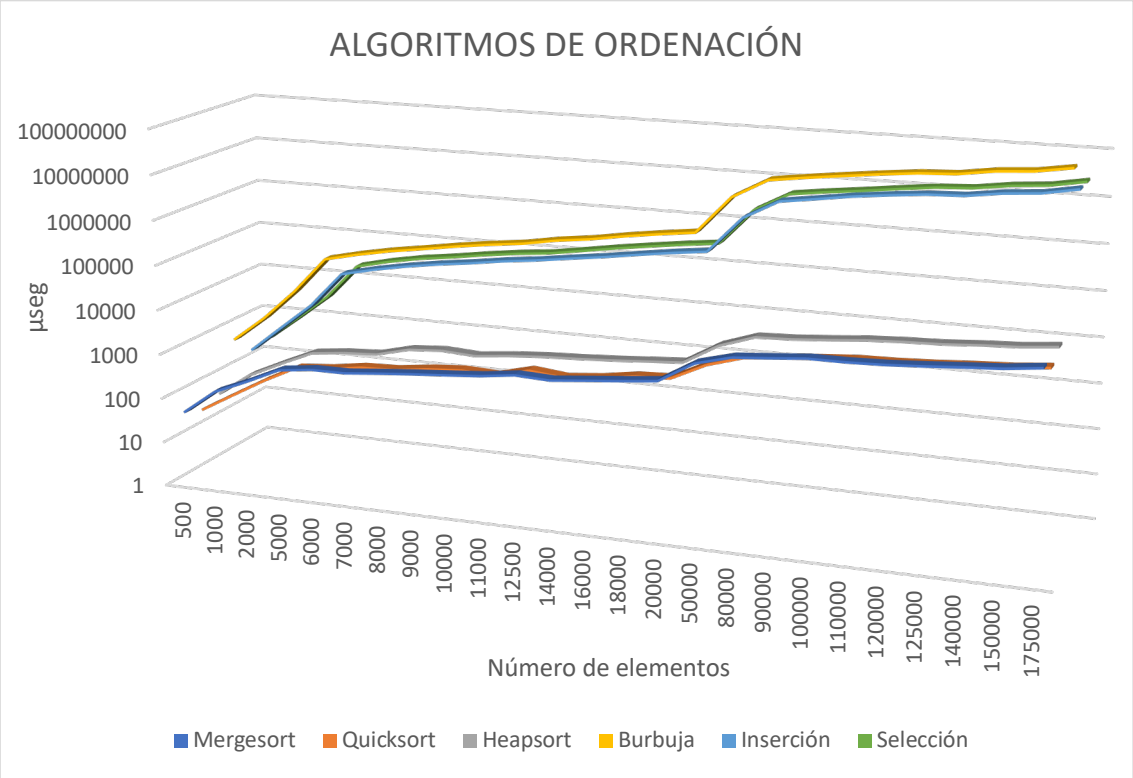
Para comprender los resultados obtenidos en los algoritmos **mergesort**, **quicksort** y **heapsort** nos ayudamos también de la composición de las tres gráficas



Podemos apreciar que los tres algoritmos son bastante similares en términos de eficiencia, sin embargo, al llegar a 20.000 componentes, los algoritmos empiezan a diferenciarse, siendo el Heapsort el que más sufre en eficiencia y el Quicksort el mejor, pues su crecimiento es mas lento a vectores más grandes.

Capítulo 5: Entendiendo resultados globales

En este capítulo abordaremos una comparación global para los algoritmos de ordenación. Comenzaremos exponiendo la gráfica con todos ellos juntos y posteriormente interpretaremos el resultado.



Como se puede observar, claramente los algoritmos de orden de eficiencia $O(n^2)$ son notoriamente menos eficientes que los de orden de eficiencia $O(n \log n)$ y, donde de nuevo, vemos que el mejor de los algoritmos, en términos empíricos, es el Quicksort y el peor sería el Burbuja. Además se define un salto bastante más amplio en vectores más grandes entre los algoritmos de distinto orden de eficiencia, es decir, a más número de componentes en el vector, más notoria será la diferencia en eficiencia al usar un algoritmo de orden $O(n \log n)$ que uno de orden $O(n^2)$.

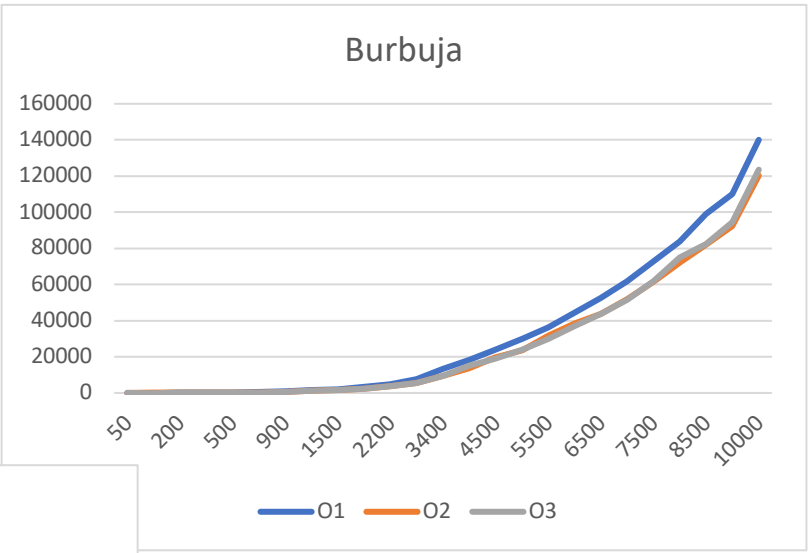
Capítulo 6: Eficiencia de los algoritmos y parámetros externos

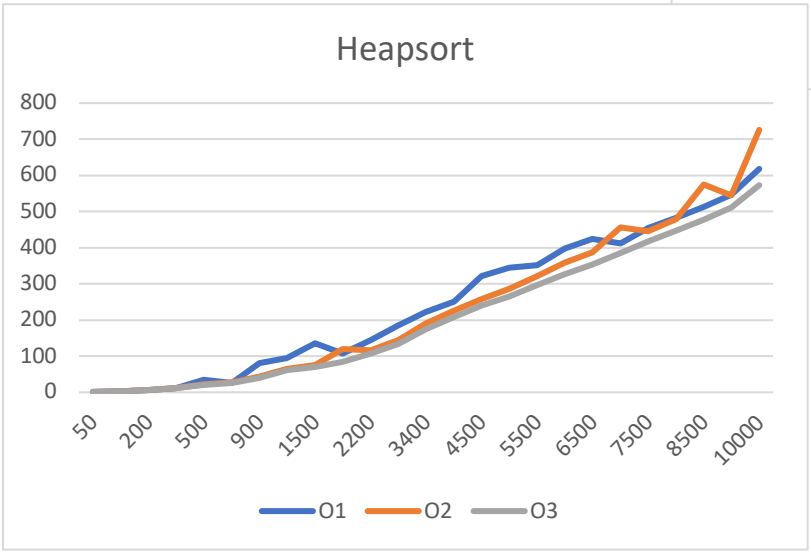
En este capítulo usamos distintas optimizaciones a la hora de compilar para ver cambios en la eficiencia cuando ejecutamos los distintos algoritmos. Asimismo, se incluye también pruebas en distintos computadores y distintos sistemas operativos.

6.1: Optimizaciones

En este apartado diremos que hemos usado el algoritmo de **burbuja** y el algoritmo **Heapsort** para realizar la prueba. A continuación, se expone la tabla y la gráfica con los resultados y una breve explicación.

	BURBUJA			HEAPSORT		
Número de componentes	O1	O2	O3	O1	O2	O3
50	4	3	3	1	1	1
100	14	29	11	3	3	3
200	45	70	37	7	7	7
300	98	124	81	12	12	12
500	273	245	201	35	22	21
600	456	310	285	26	27	25
900	769	601	586	80	43	40
1300	1584	1276	1238	94	65	62
1500	2096	1654	1623	136	75	70
1800	3447	2349	2378	108	120	85
2200	4874	3809	3826	144	116	107
2700	7721	5411	5548	185	145	134
3400	13150	9395	9446	222	190	175
4000	18241	13796	15114	251	225	208
4500	23917	19636	18835	321	258	240
5000	29885	23718	23895	345	287	265
5500	36461	31705	29786	351	322	296
6000	44429	38576	36981	398	358	326
6500	52648	43899	43601	425	387	353
7000	61726	52067	51702	412	457	386
7500	72753	61588	61810	455	445	417
8000	83747	72058	74851	482	479	448
8500	99095	82200	82280	512	574	478
9000	110147	92465	94463	547	545	511
10000	140055	120272	123547	618	726	573



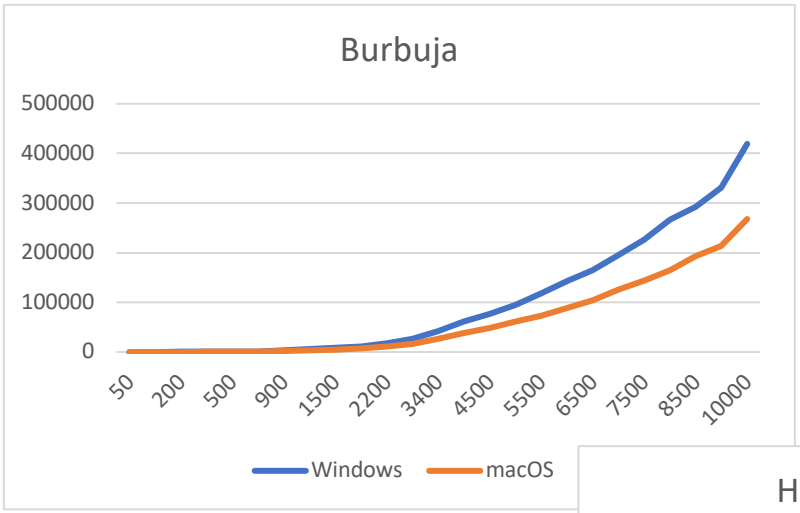


Se puede apreciar que la opción de optimización -O2 en el caso de la burbuja nos arroja una mejor eficiencia respecto a las otras dos opciones de compilación. En cambio, en Heapsort, la opción -O3 nos produce un mejor resultado.

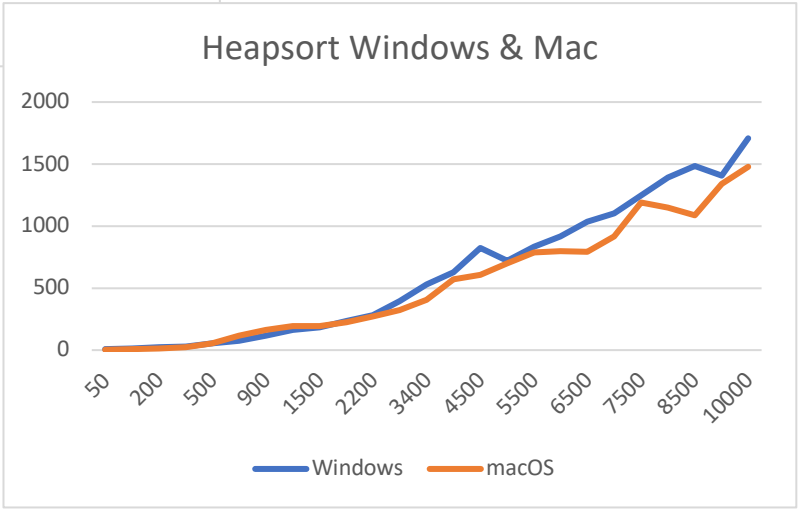
6.2: Computadores y sistemas operativos diferentes

En este apartado compararemos dos computadoras diferentes, con sus especificaciones diferentes (anexadas al final del documento) y sistemas operativos distintos.

HEAPSORT			BURBUJA		
Número de componentes	Windows	macOS	Número de componentes	Windows	macOS
50	8	3	50	15	25
100	12,1	7	100	46	36
200	25,1	15	200	206	98
300	30,5	25	300	406	254
500	54,1	57	500	946,67	583
600	77,3	114	600	1380,3	814
900	113,9	165	900	2753,3	1616
1300	162,5	192	1300	5795,67	3429
1500	185,3	194	1500	7890,3	4902
1800	234,8	227	1800	11166,3	7688
2200	280,1	273	2200	17335	11123
2700	393,2	323	2700	26142,3	16760
3400	527,9	406	3400	42051,3	26779
4000	626,4	573	4000	61300,67	38246
4500	823,6	607	4500	76547	48468
5000	717,9	699	5000	94377,67	61167
5500	834,1	788	5500	117984,7	73060
6000	914,2	798	6000	143069,7	88172
6500	1035,5	791	6500	165193,7	103793
7000	1101,1	918	7000	196116,7	126078
7500	1246,5	1188	7500	226038,7	144133
8000	1390,4	1147	8000	266502,3	163986
8500	1482,9	1089	8500	292099,3	192855
9000	1406	1341	9000	331617	213408
10000	1707,7	1478	10000	419294	267884



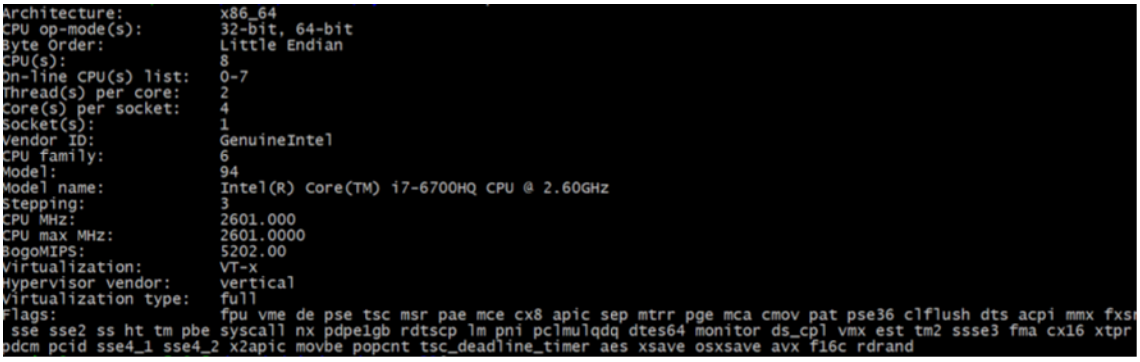
Se observa que en ambos casos la ejecución en macOS nos da un mejor resultado a vectores más grandes.



Anexos

```
Arquitectura:                         x86_64
modo(s) de operación de las CPUs:     32-bit, 64-bit
Orden de los bytes:                   Little Endian
CPU(s):                               6
Lista de la(s) CPU(s) en línea:       0-5
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:               12
«Socket(s)»                           1
Modo(s) NUMA:                         1
ID de fabricante:                     GenuineIntel
Familia de CPU:                       6
Modelo:                               158
Nombre del modelo:                    Intel(R) Core(TM) i7-8700B CPU @ 3.20GHz
Revisión:                             10
CPU MHz:                              3192.000
BogoMIPS:                             6384.00
Fabricante del hipervisor:            KVM
Tipo de virtualización:               lleno
Caché L1d:                            32K
Caché L1i:                            32K
Caché L2:                             256K
Caché L3:                             12288K
CPU(s) del nodo NUMA 0:               0-3
Indicadores:                          fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx rdtscp lm constant_tsc nopl xtopology
nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe
popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch
invpcid_single pti fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 invpcid mpx rdseed adx smap
clflushopt xsaveopt xsavec xsaves dtherm arat pln pts
```

Especificaciones del ordenador usado (macOS)



Especificaciones del ordenador Windows