Jaime Corzo Galdó, Grupo 1

PRÁCTICA 1: ALGORÍTMICA

→ Titulación: Doble Grado Ingeniería Informática y Matemáticas(DGIIM)
→ Asignatura: Algorítmica(Segundo curso)
→ Práctica 1: Análisis de eficiencia de algoritmos
→ Nombre: Jaime Corzo Galdó
→ Correo: jaimecrz04@correo.ugr.es

Objetivos de la Práctica

El objetivo de esta práctica es que el estudiante comprenda la importancia del análisis de la eficiencia de los algoritmos y se familiarice con las formas de llevarlo a cabo. Para ello se mostrará cómo realizar un estudio teórico, empírico e híbrido. Cada estudiante deberá realizar los análisis empíricos e híbridos de los algoritmos que se detallan más adelante.

Diseño del Estudio

Tamaños e instancias de casos de entrada usadas para los análisis empíricos e híbridos:

→ Orden O(n 2): desde 5000 hasta 125000 con saltos de 5000.
→ Orden O(n log2 n): desde 50000 hasta 1250000 con saltos de 50000.

Entorno de análisis:

→ Hardware: ASUSTeK COMPUTER INC. ZenBook
→ SO: Ubuntu 22.04.2 LTS
→ Compilador: g++
→ Procesador: Intel® Core™ i7-1065G7 CPU @ 1.30GHz × 8

Método de medición del tiempo empleado:

→ Biblioteca de la STL, chrono

Índice

1. Algoritmos O(n^2) ……. pág. 2
   a) Mediciones de tiempo para todos los algoritmos
   b) Ajuste del algoritmo i-ésimo
   c) Comparar las curvas de tiempo de todos los algoritmos del grupo
   d) Análisis de eficiencia empírica e híbrida usando para la compilación -O3
2. Algoritmos O(n*log(n)) ..... pág. 17
   a) Mediciones de tiempo para todos los algoritmos
   b) Ajuste del algoritmo i-ésimo
   c) Comparar las curvas de tiempo de todos los algoritmos del grupo
   d) Análisis de eficiencia empírica e híbrida usando para la compilación -O3
3. Estudio comparativo de todos los algoritmos ..... pág. 27
4. Conclusiones ..... pág. 28

Jaime Corzo Galdó, Grupo 1

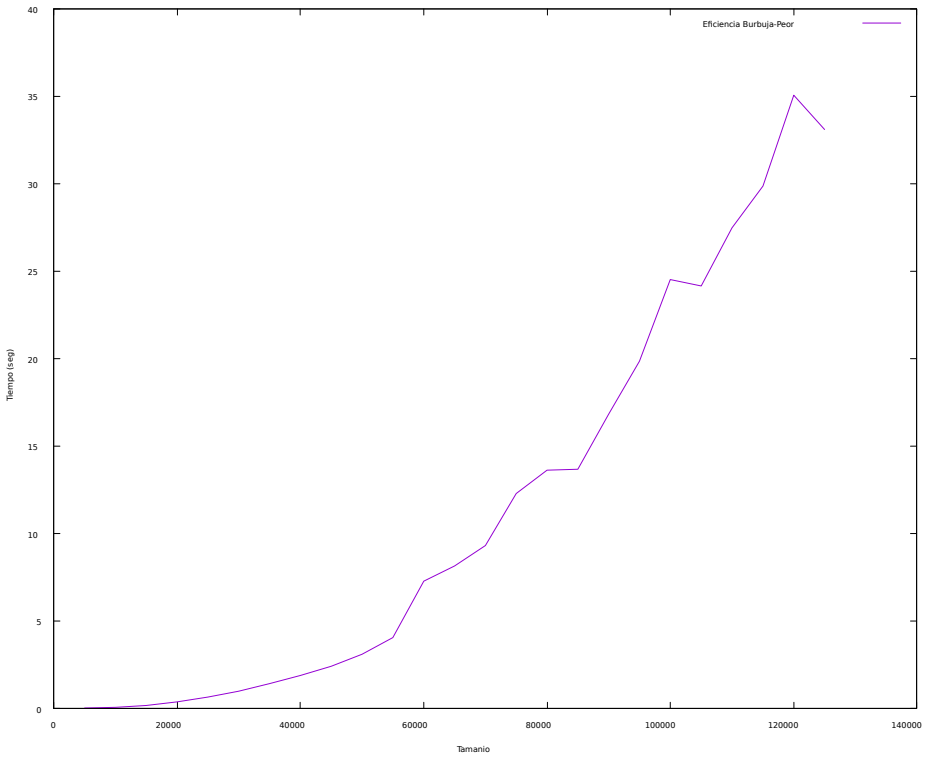
1. Algoritmos O(n^2)

a) Mediciones de tiempo para todos los algoritmos

Burbuja:

Peor Caso: El peor caso es cuando el vector está ordenado de mayor a menor, (O(n^2))

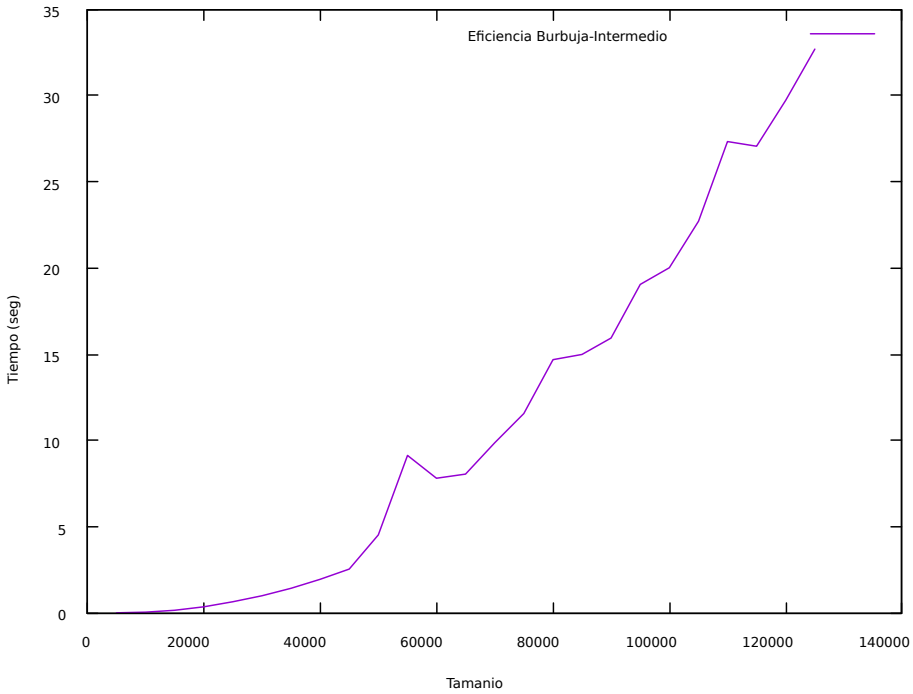
Tamaño	Tiempo(s)
5000	0.0173357
10000	0.0675072
15000	0.150587
20000	0.271562
25000	0.426347
30000	0.608677
35000	0.881532
40000	1.11963
45000	2.51305
50000	3.5479
55000	5.18825
60000	8.62289
65000	9.82359
70000	10.04984
75000	11.19221
80000	13.90897
85000	14.11139
90000	15.09325
95000	17.91385
100000	21.88216
105000	23.0267
110000	25.4044
115000	27.3569
120000	33.3788
125000	35.4374



Jaime Corzo Galdó, Grupo 1

Caso Intermedio: Cuando está ordenado de manera aleatoria, ($O(n^2)$)

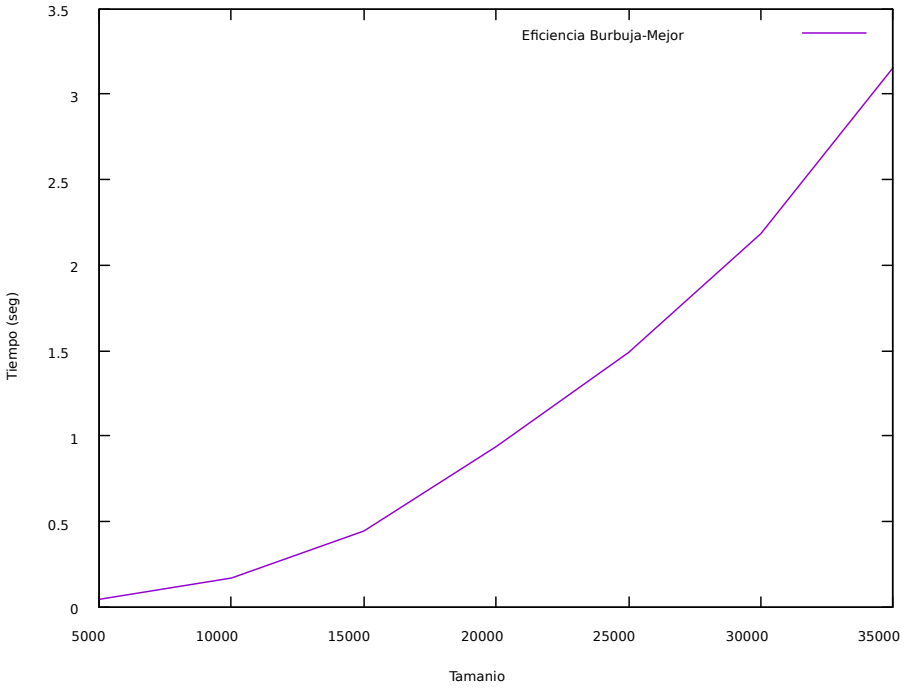
Tamaño Tiempo
000 0.0139577
10000 0.0539736
15000 0.162701
20000 0.365158
25000 0.656898
30000 1.00119
35000 1.44095
40000 1.9545
45000 2.55093
50000 4.53158
55000 9.13026
60000 7.82085
65000 8.04782
70000 9.87186
75000 11.5686
80000 14.68
85000 14.9885
90000 15.9357
95000 19.0476
100000 20.0161
105000 22.7073
110000 27.3188
115000 27.0556
120000 29.72
125000 32.6761



Jaime Corzo Galdó, Grupo 1

Mejor Caso: Cuando ya está ordenado, (O(n))

Tamaño	Tiempo(s)
5000	0.00531141
10000	0.0205015
15000	0.0513062
20000	0.0796221
25000	0.123401
30000	0.178523
35000	0.242776
40000	0.322792
45000	0.404675
50000	0.497454
55000	0.600086
60000	0.752004
65000	2.01136
70000	1.44985
75000	2.20238
80000	2.54142
85000	4.07612
90000	3.33706
95000	4.04992
100000	2.99094
105000	4.96791
110000	7.01499
115000	3.91392
120000	4.29836
125000	4.91862

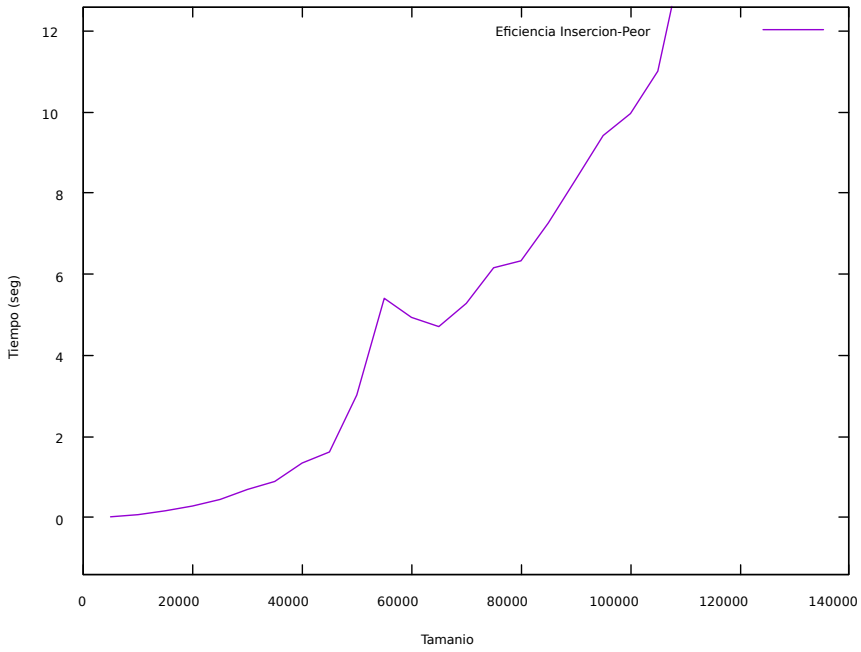


Jaime Corzo Galdó, Grupo 1

Inserción:

Peor Caso: Cuando está ordenado de mayor a menor, ($O(n^2)$)

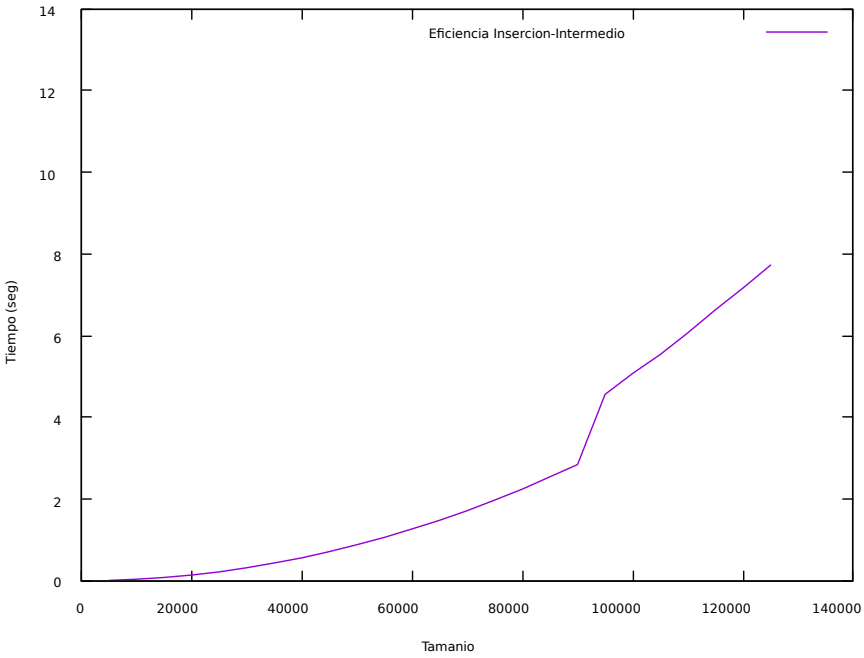
Tamaño	Tiempo(s)
5000	0.01837
10000	0.0708087
15000	0.164721
20000	0.28713
25000	0.444372
30000	0.69228
35000	0.890463
40000	1.34314
45000	1.61456
50000	3.02052
55000	5.40568
60000	4.93789
65000	4.70879
70000	5.27965
75000	6.16051
80000	6.32866
85000	7.2623
90000	8.33503
95000	9.41607
100000	9.96249
105000	11.008
110000	14.1157
115000	14.7387
120000	14.5292
125000	17.157



Jaime Corzo Galdó, Grupo 1

Caso Intermedio: Cuando está ordenado de manera aleatoria, ($O(n^2)$)

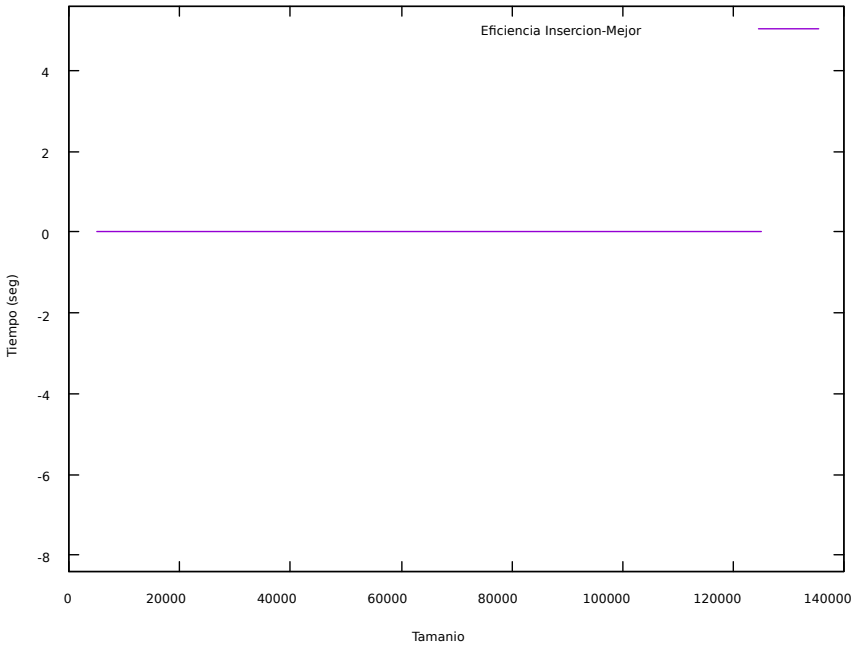
Tamaño	Tiempo(s)
5000	0.0108518
10000	0.0365514
15000	0.0780279
20000	0.138413
25000	0.216607
30000	0.318485
35000	0.4353
40000	0.562038
45000	0.713008
50000	0.881282
55000	1.06285
60000	1.26671
65000	1.48177
70000	1.7191
75000	1.97583
80000	2.24318
85000	2.54505
90000	2.84141
95000	4.5655
100000	5.07769
105000	5.5382
110000	6.06576
115000	6.63164
120000	7.16414
125000	7.72411



Jaime Corzo Galdó, Grupo 1

Mejor Caso: Cuando ya está ordenado, (O(n)), en este caso se puede comprobar que es muy eficiente(el tiempo es prácticamente 0)

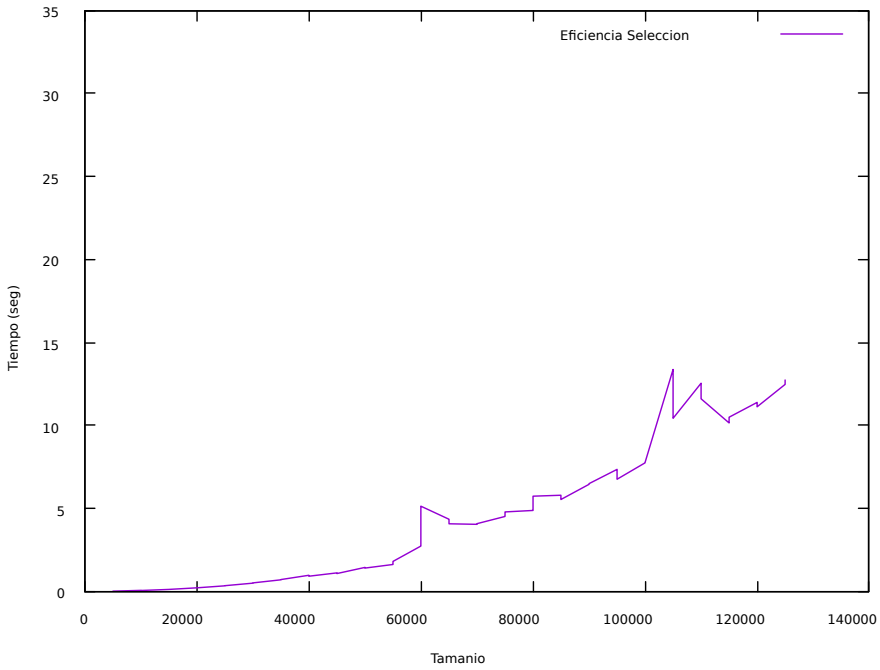
Tamaño Tiempo(s)
5000 6.029e-06
10000 8.055e-06
15000 1.123e-05
20000 1.9651e-05
25000 1.8265e-05
30000 2.1884e-05
35000 2.6264e-05
40000 3.0663e-05
45000 3.492e-05
50000 4.4074e-05
55000 3.9844e-05
60000 4.2416e-05
65000 4.6073e-05
70000 5.0148e-05
75000 5.2556e-05
80000 6.3549e-05
85000 6.2366e-05
90000 6.3115e-05
95000 6.6328e-05
100000 7.3511e-05
105000 7.3067e-05
110000 0.000119856
115000 0.000124862
120000 0.000125417
125000 0.0001306



Jaime Corzo Galdó, Grupo 1

Selección: Ningún caso es particularmente bueno ni malo para el ordenamiento por selección. Se ejecuta en un tiempo $\Theta(n2)$ para todos los casos.

Tamaño	Tiempo(s
5000	0.014574
10000	0.052887
15000	0.117192
20000	0.220347
25000	0.344294
30000	0.51243
35000	0.707295
40000	0.919388
45000	1.07052
50000	1.4025
55000	1.81268
60000	5.13279
65000	4.06981
70000	4.07479
75000	4.78859
80000	5.73207
85000	5.53588
90000	6.49505
95000	6.75194
100000	7.76319
105000	10.4219
110000	11.6024
115000	10.4911
120000	11.1135
125000	12.7314



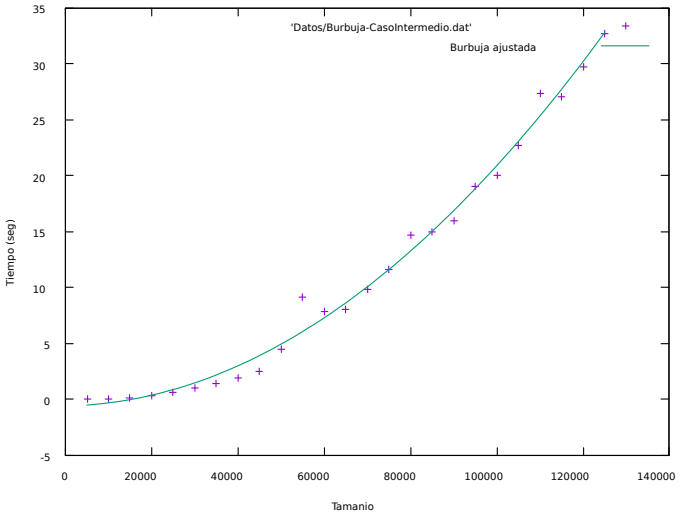
Jaime Corzo Galdó, Grupo 1

b) Ajuste del algoritmo i-ésimo

Burbuja: f(x) = a2*x*x+a1*x+a0

a2 = 2.08532e-09
a1 = 6.13373e-06
a0 = -0.587381

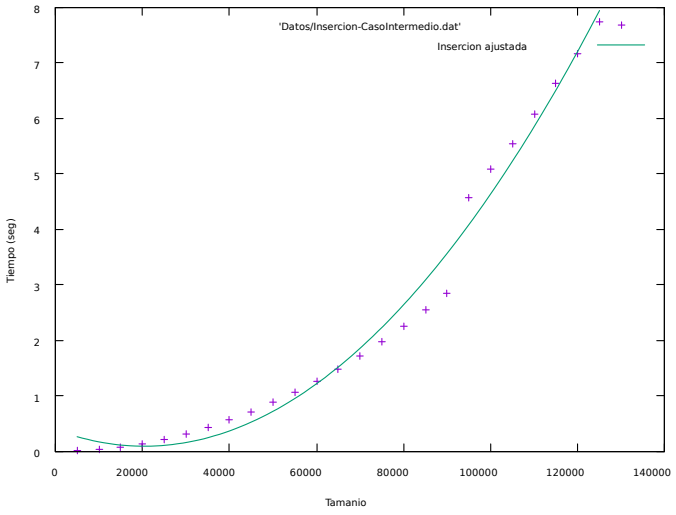
Ajuste de la función a nuestros datos



Inserción: f(x) = a2*x*x+a1*x+a0

a2 = 7.16898e-10
a1 = -2.92579e-05
a0 = 0.392643

Ajuste de la función a nuestros datos

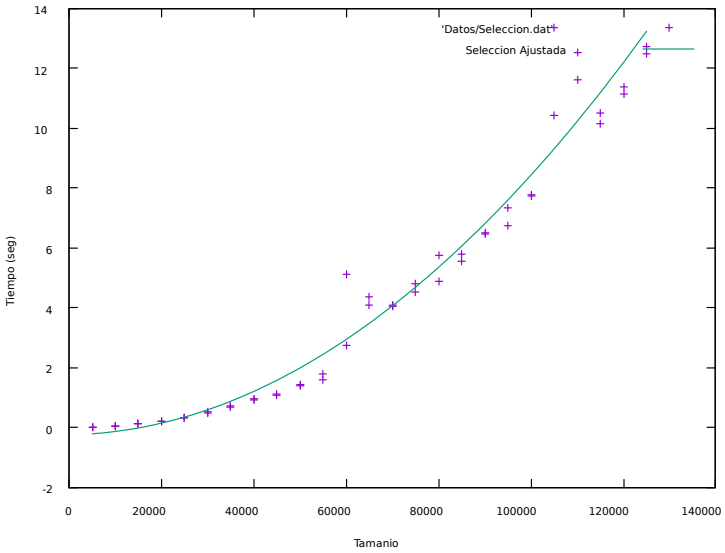


Jaime Corzo Galdó, Grupo 1

Selección: f(x) = a2*x*x+a1*x+a0

a2 = 8.40986e-10
a1 = 2.59368e-06
a0 = -0.240613

Ajuste de la función a nuestros datos

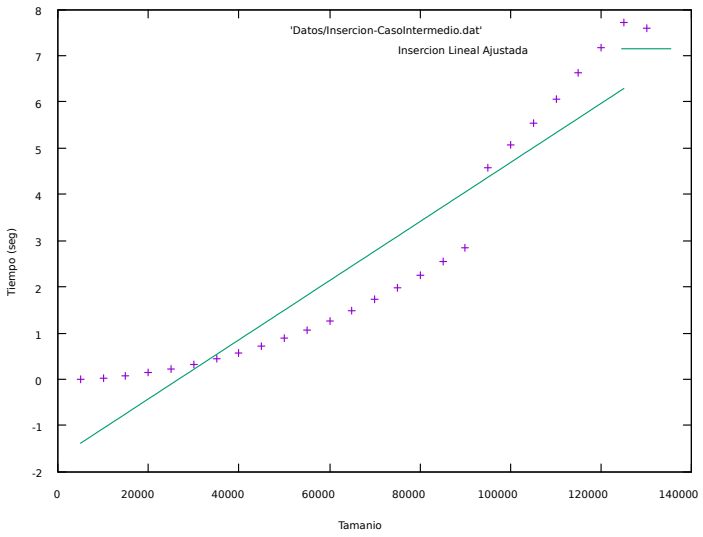


Ajuste a curva lineal: f(x) = a1*x + a0,

Datos empíricos: Inserción

a1 = 6.39388e-05
a0 = -1.70428

Ajuste de la función a los datos



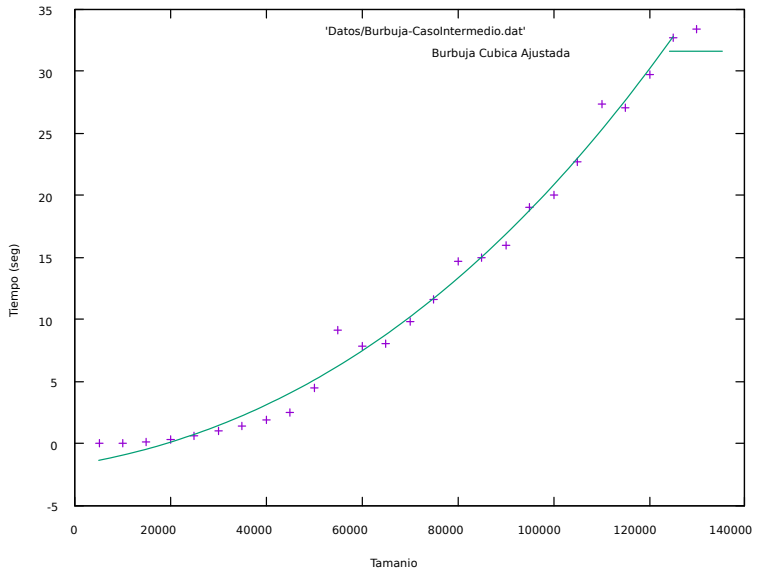
Jaime Corzo Galdó, Grupo 1

Ajuste a curva cúbica: f(x) = a3*x*x*x+a2*x*x+a1*x+a0

Datos empíricos: Burbuja

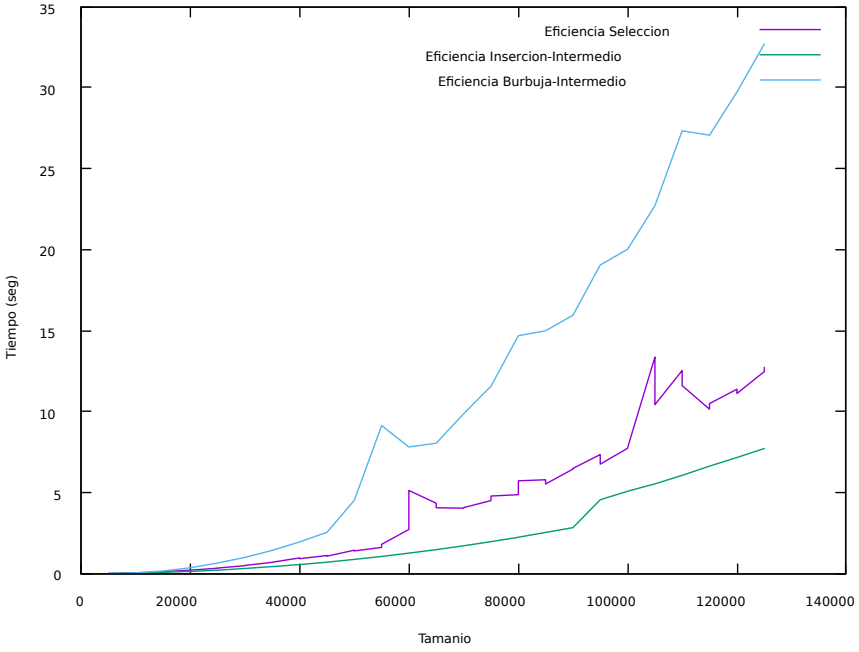
a3 = 3.30084e-15
a2 = 1.28243e-09
a1 = 6.39388e-05
a0 = -1.70428

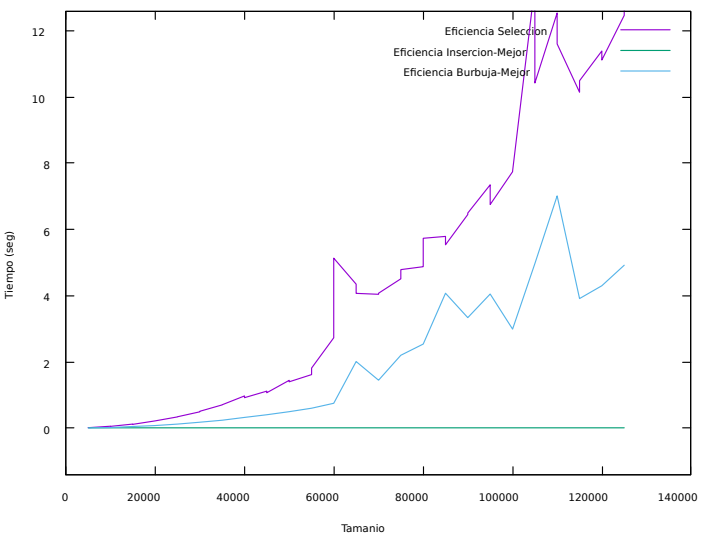
Ajuste de la función a los datos

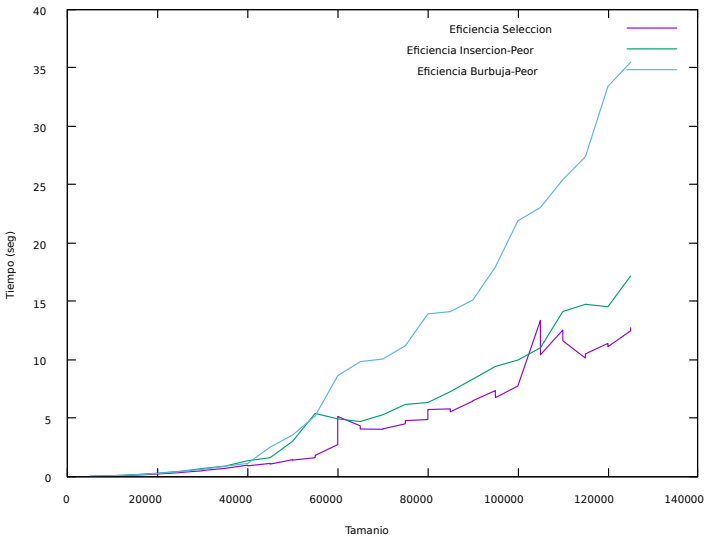


Jaime Corzo Galdó, Grupo 1

c) Comparar las curvas de tiempo de todos los algoritmos del grupo: Como podemos ver en la gráfica, el algoritmo más eficiente es el de inserción, mientras que el burbuja para tamaños grandes se dispara. En caso mejor vemos que al pasar inserción y burbuja a ser lineales, selección no sería tan efectivo y en el peor caso inserción y selección son parecidos(inserción un poco peor) y burbuja de nuevo el menos eficiente.







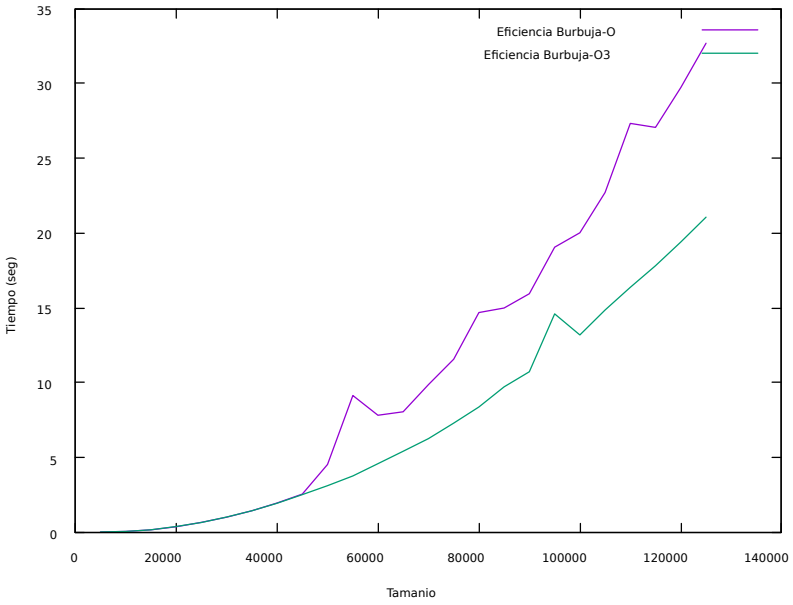
Jaime Corzo Galdó, Grupo 1

d) Análisis de eficiencia empírica e híbrida usando para la compilación -O3

Burbuja

Datos Burbuja(con Compilación -O3) Comparación Compilaciones(-O/-O3)

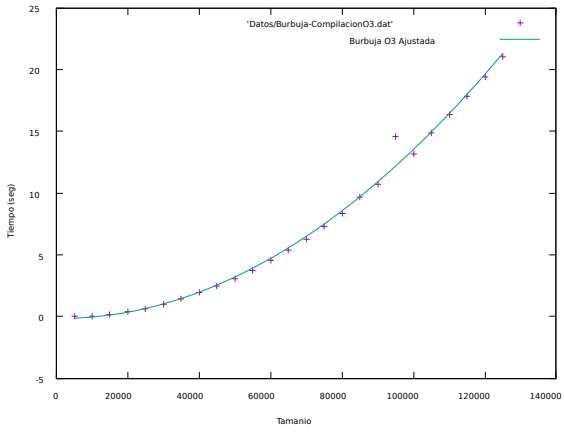
Tamaño	Tiempo(s)
5000	0.0160884
10000	0.0530875
15000	0.161434
20000	0.377414
25000	0.646057
30000	1.0184
35000	1.43597
40000	1.94176
45000	2.51054
50000	3.10964
55000	3.76448
60000	4.57879
65000	5.41308
70000	6.25656
75000	7.28961
80000	8.37026
85000	9.71049
90000	10.7262
95000	14.5948
100000	13.1859
105000	14.8537
110000	16.3676
115000	17.8095
120000	19.4012
125000	21.0579



Ajuste(para O3): f(x) = a2*x*x+a1*x+a0

a2 = 1.39424e-09
a1 = -2.67533e-06
a0 = -0.143252

Ajuste de la función a los datos

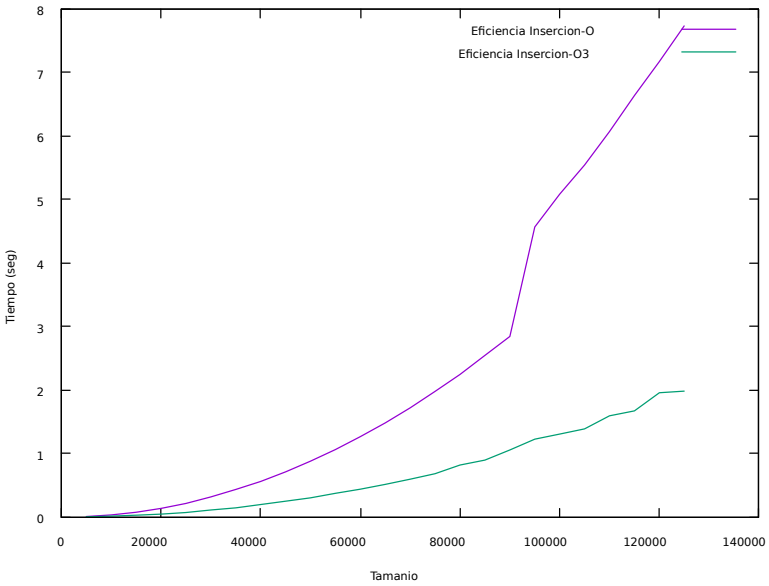


Jaime Corzo Galdó, Grupo 1

Inserción

Datos Inserción(con Compilación -O3) Comparación Compilaciones(-O/-O3)

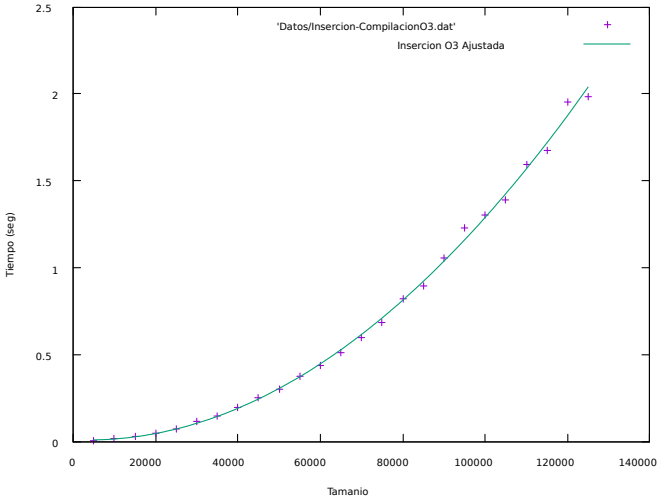
Tamaño	Tiempo(s)
5000	0.00515526
10000	0.0156177
15000	0.0288641
20000	0.0482553
25000	0.0738541
30000	0.113622
35000	0.14707
40000	0.198361
45000	0.2504
50000	0.303595
55000	0.374351
60000	0.439566
65000	0.514262
70000	0.598529
75000	0.683167
80000	0.820109
85000	0.895818
90000	1.05591
95000	1.22727
100000	1.30677
105000	1.38769
110000	1.59268
115000	1.67244
120000	1.95613
125000	1.98119



Ajuste(para O3): f(x) = a2*x*x+a1*x+a0

a2 = 1.37821e-10
a1 = -1.00395e-06
a0 = 0.0114225

Ajuste de la función a los datos



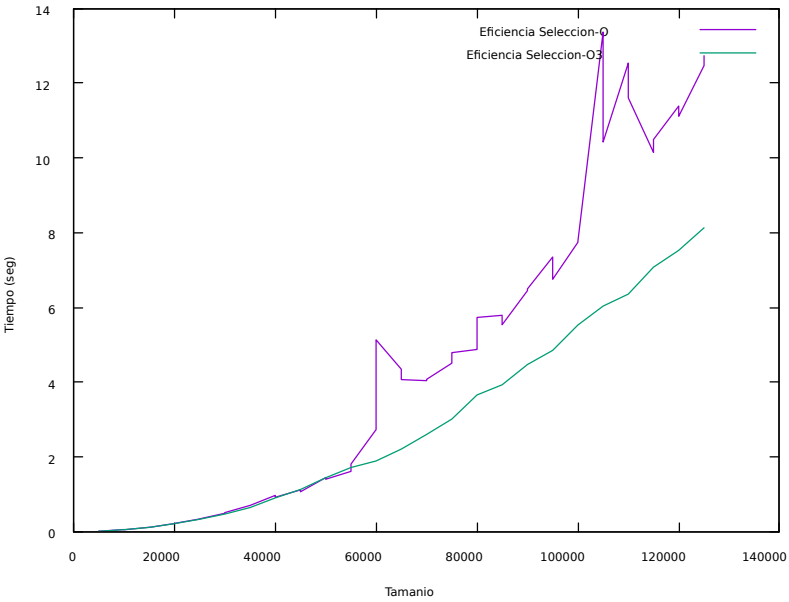
Jaime Corzo Galdó, Grupo 1

Selección

Datos Seleccion(con Compilación -O3)

Tamaño	Tiempo(s)
5000	0.015536
10000	0.056884
15000	0.119349
20000	0.21353
25000	0.334143
30000	0.47294
35000	0.647152
40000	0.902058
45000	1.1273
50000	1.44322
55000	1.71653
60000	1.89346
65000	2.20812
70000	2.60307
75000	3.01145
80000	3.65916
85000	3.93151
90000	4.47216
95000	4.85259
100000	5.52773
105000	6.03719
110000	6.35468
115000	7.0793
120000	7.52425
125000	8.12669

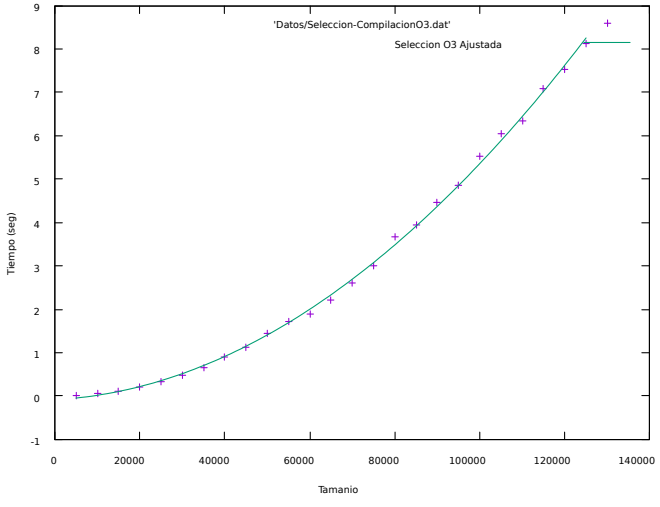
Comparación Compilaciones(-O/-O3)



Ajuste(para O3): f(x) = a2*x*x+a1*x+a0

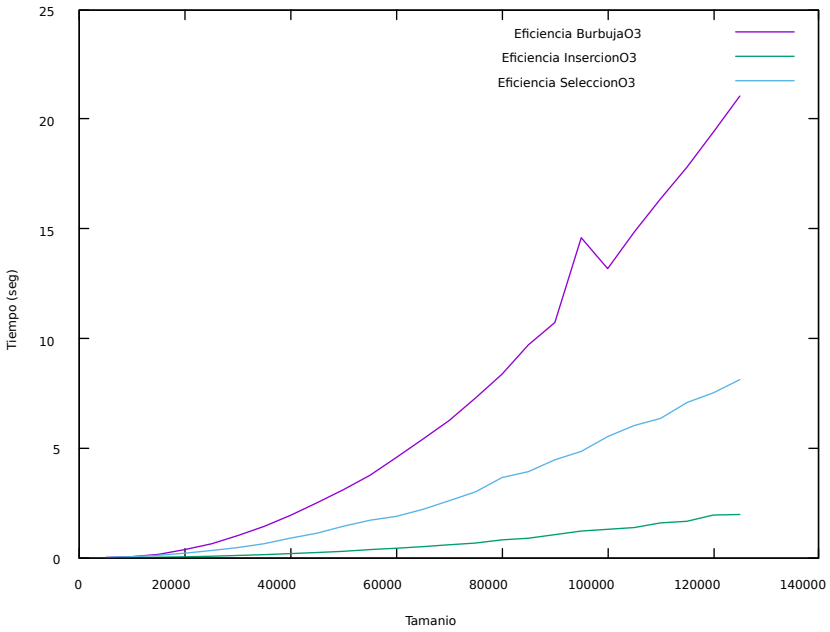
a2 = 4.91225e-10
a1 = 5.31002e-06
a0 = -0.0859668

Ajuste de la función a los datos



Jaime Corzo Galdó, Grupo 1

Comparación de los tres(con Compilación -O3):



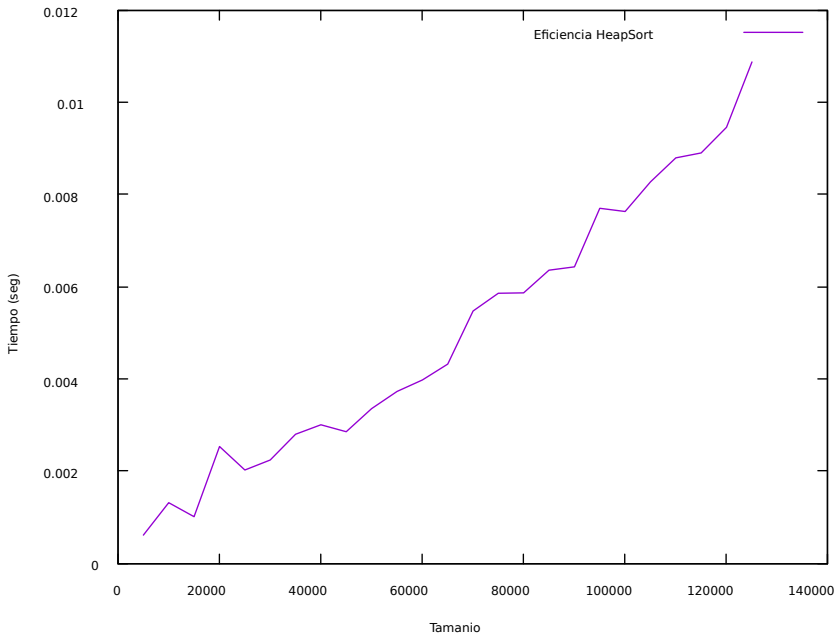
Jaime Corzo Galdó, Grupo 1

2. Algoritmos O(n*log(n))

a) Mediciones de tiempo para todos los algoritmos

Heapsort: Se ejecuta con eficiencia O=(n*log(n)) para todos sus casos

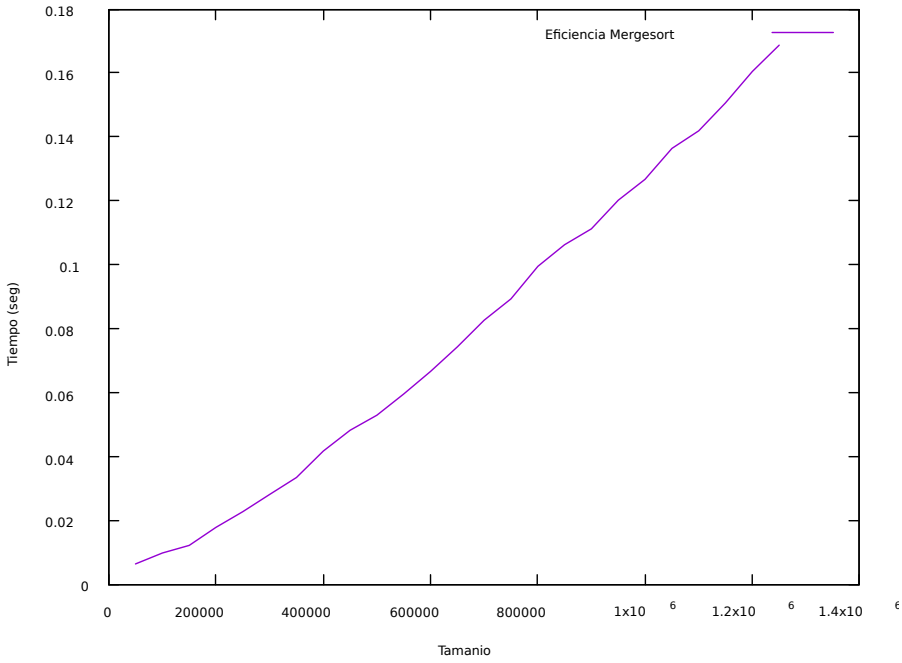
Tamaño	Tiempo(s)
50000	0.00380961
100000	0.00872668
150000	0.0124116
200000	0.0175172
250000	0.0234317
300000	0.029123
350000	0.0351742
400000	0.0404514
450000	0.0471361
500000	0.0526123
550000	0.0588389
600000	0.0696766
650000	0.0773932
700000	0.0958022
750000	0.095502
800000	0.10708
850000	0.104141
900000	0.111182
950000	0.120204
1000000	0.126586
1050000	0.134455
1100000	0.141955
1150000	0.149877
1200000	0.163692
1250000	0.16773



Jaime Corzo Galdó, Grupo 1

Mergesort: Se ejecuta con eficiencia O=(n*log(n)) para todos sus casos

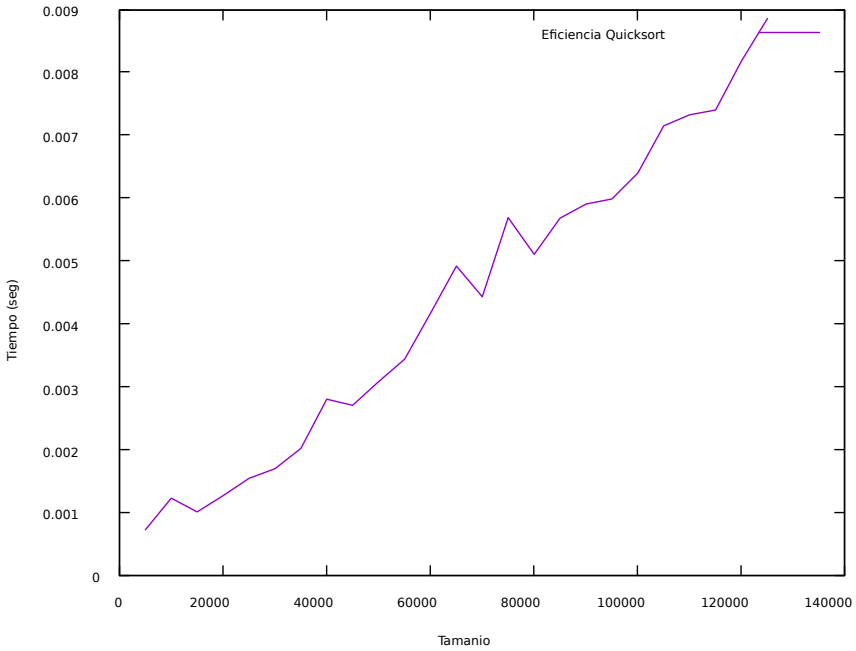
Tamaño	tiempo(s)
50000	0.00640501
100000	0.00983914
150000	0.0122314
200000	0.0178683
250000	0.0227892
300000	0.0281628
350000	0.0334982
400000	0.0417225
450000	0.0482014
500000	0.0529178
550000	0.0595681
600000	0.0666588
650000	0.074357
700000	0.0826758
750000	0.0893611
800000	0.0995064
850000	0.10629
900000	0.111259
950000	0.120222
1000000	0.126687
1050000	0.136408
1100000	0.141898
1150000	0.150631
1200000	0.16046
1250000	0.168694



Jaime Corzo Galdó, Grupo 1

Quicksort: Se ejecuta con eficiencia O=(n*log(n)) en su caso mejor e intermedio, en cambio, en el peor, cuando ya está ordenado, es O(n^2). Por otro lado, si trabajamos con datos muy grandes nos va a dar core dumped, (si llamas a un vector de 650000 elementos tienes una pila de 650000 niveles lo que quiere decir que hay muchas llamadas recursivas) lo hacemos para valores más chicos de tamaño del vector(hasta 125000)

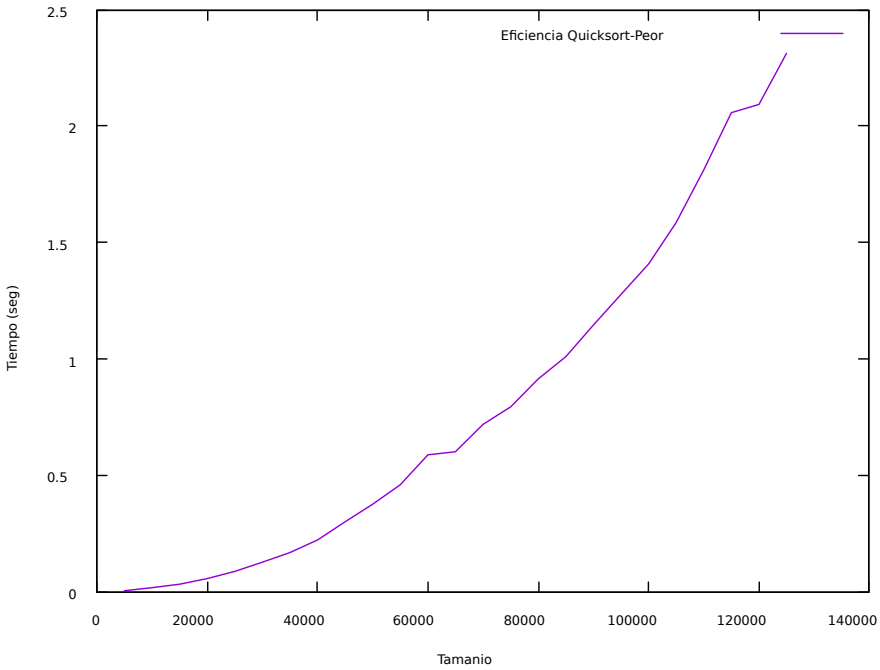
Tamaño	Tiempo(s)
5000	0.000723185
10000	0.00122639
15000	0.00100898
20000	0.0012664
25000	0.0015425
30000	0.0016959
35000	0.00201827
40000	0.00280052
45000	0.0027031
50000	0.00307915
55000	0.00343647
60000	0.00416986
65000	0.00491841
70000	0.00443028
75000	0.00568728
80000	0.0051046
85000	0.00567803
90000	0.00590366
95000	0.00598605
100000	0.00639806
105000	0.00714787
110000	0.0073229
115000	0.00739851
120000	0.00817758
125000	0.00885551



Jaime Corzo Galdó, Grupo 1

Peor caso Quicksort

Tamaño	Tiempo(s)
5000	0.00509543
10000	0.0178125
15000	0.0325659
20000	0.0571489
25000	0.0880482
30000	0.127105
35000	0.168989
40000	0.223136
45000	0.301224
50000	0.376848
55000	0.459285
60000	0.588734
65000	0.601841
70000	0.719256
75000	0.794313
80000	0.914711
85000	1.00979
90000	1.14532
95000	1.27729
100000	1.4073
105000	1.58576
110000	1.81275
115000	2.05709
120000	2.09322
125000	2.31172



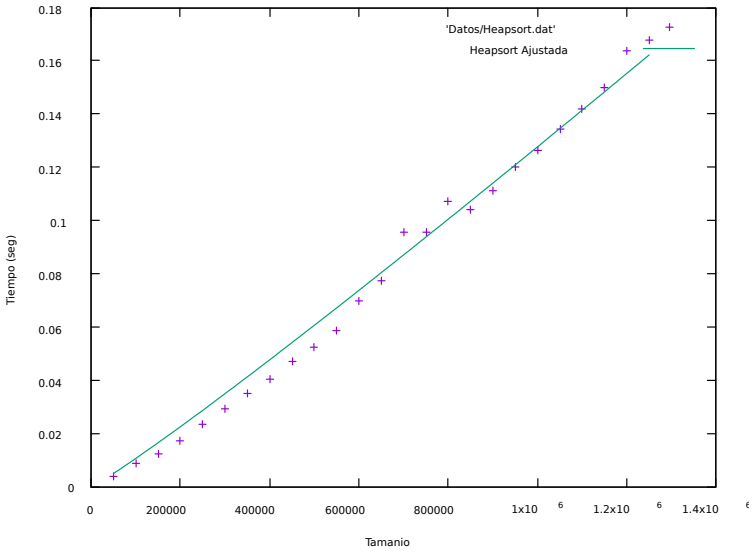
Jaime Corzo Galdó, Grupo 1

b) Ajuste del algoritmo i-ésimo

Heapsort: f(x) = a0*x*log(x)

a0 = 9.24534e-09

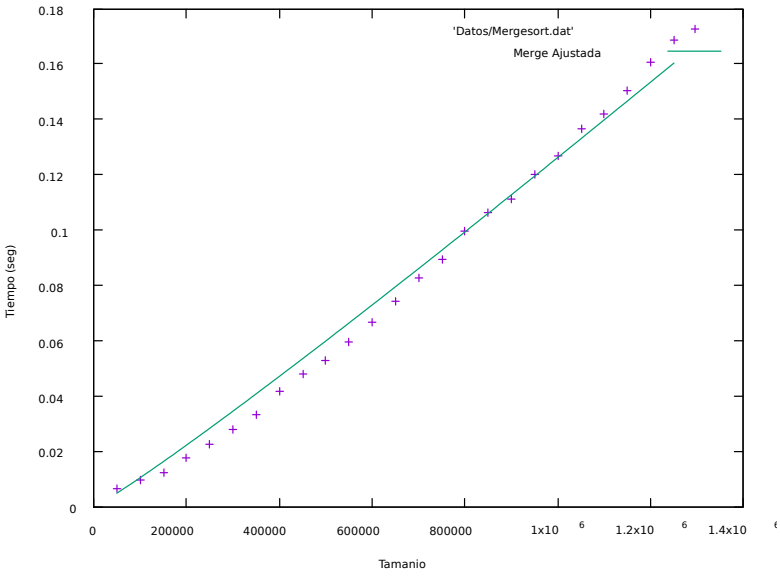
Ajuste de la función a los datos



Mergesort: f(x) = a0*x*log(x)

a0 = 9.13825e-09

Ajuste de la función a los datos

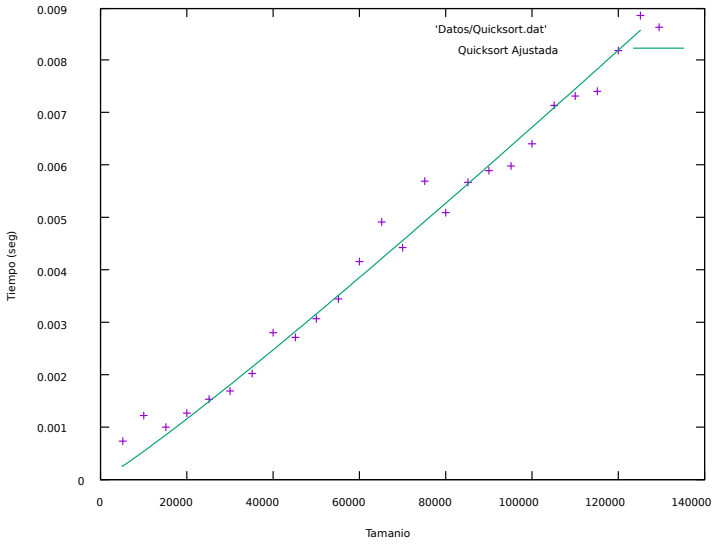


Jaime Corzo Galdó, Grupo 1

Quicksort: f(x) = a0*x*log(x)

a0 = 5.84387e-09

Ajuste de la función a los datos

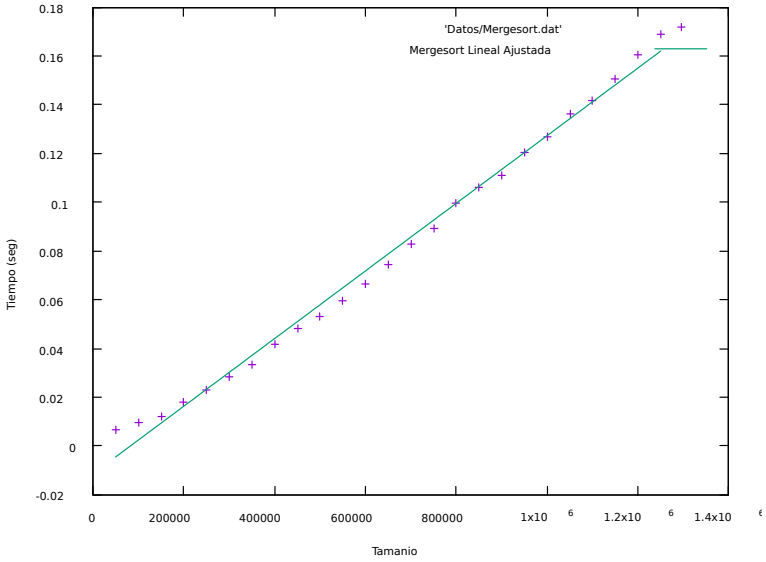


Ajuste a curva lineal: f(x) = a1*x + a0

Datos empíricos: Mergesort

a1 = 1.38753e-07
a0 = -0.0114568

Ajuste de la función a los datos



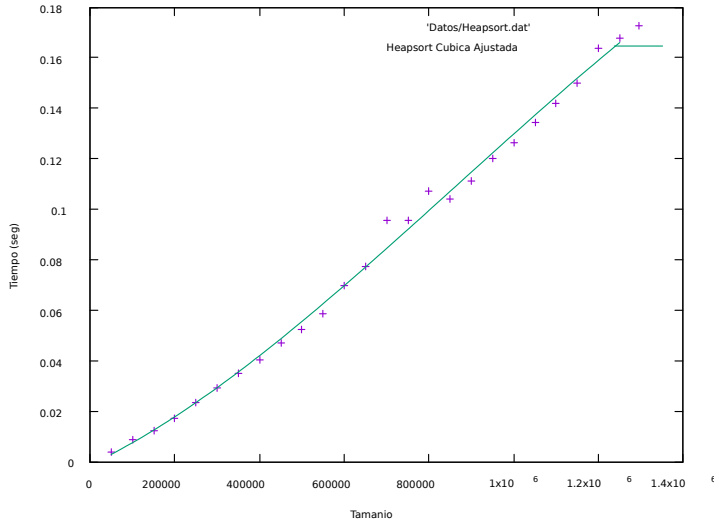
Jaime Corzo Galdó, Grupo 1

Ajuste a curva cúbica: f(x) = a3*x*x*x+a2*x*x+a1*x + a0

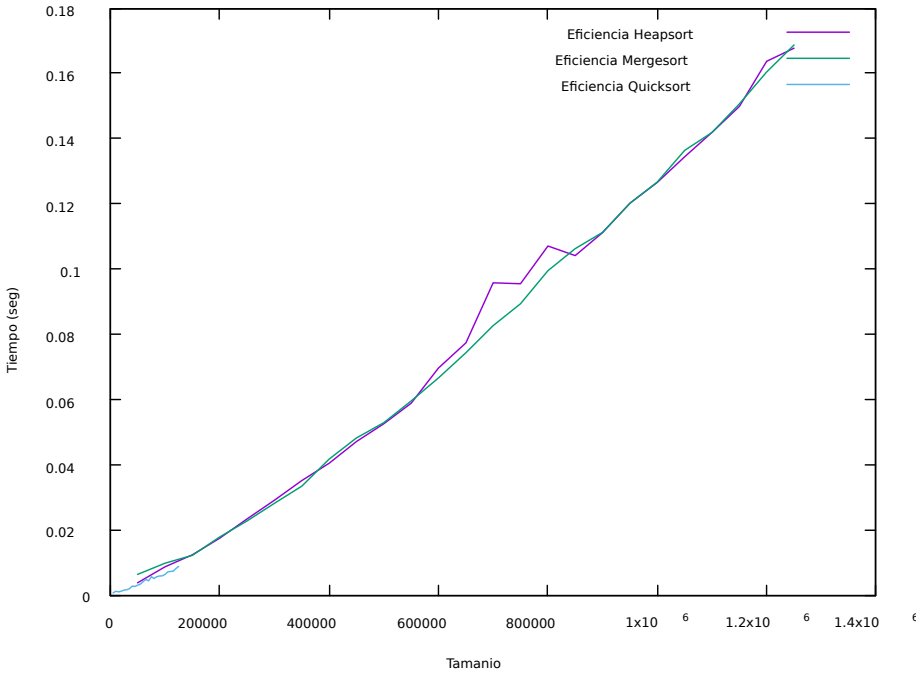
Datos empíricos: Heapsort

a3 = -3.19893e-20
a2 = 8.28582e-14
a1 = 8.01678e-08
a0 = -0.00121354

Ajuste de la función a los datos



c) Comparar las curvas de tiempo de todos los algoritmos del grupo: Como podemos ver, al tener los tres algoritmos la misma eficiencia, O(n*log(n)), (en su caso intermedio) las gráficas son muy parecidas, por lo que en esta caso es difícil quedarse con uno, aunque sabemos que el Quicksort es el más rápido en la práctica.



Jaime Corzo Galdó, Grupo 1

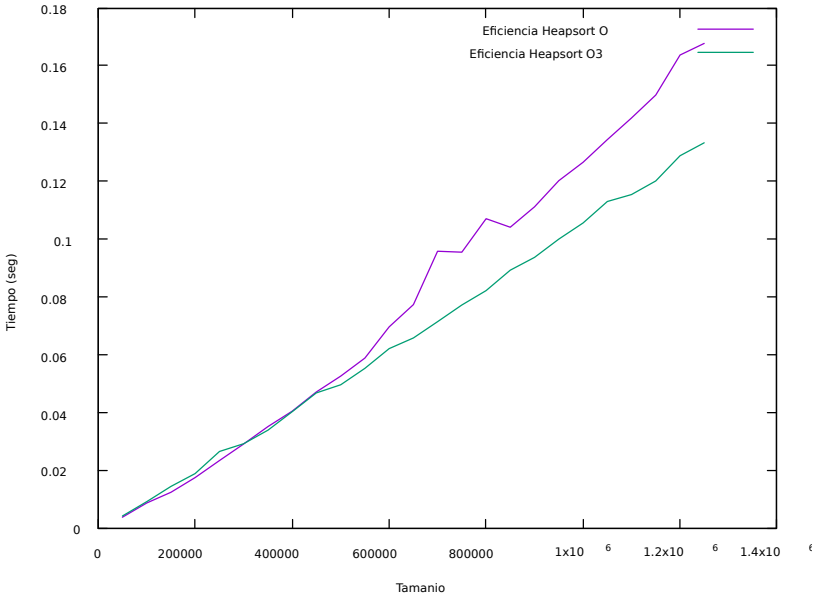
d) Análisis de eficiencia empírica e híbrida usando para la compilación -O3

Heapsort:

Datos Heapsort(con Compilación -O3)

Tamaño	Tiempo(s)
50000	0.00423785
100000	0.00920937
150000	0.0144655
200000	0.0188775
250000	0.0265321
300000	0.0292647
350000	0.0339133
400000	0.0403039
450000	0.0467764
500000	0.0496364
550000	0.0553003
600000	0.0621347
650000	0.0658039
700000	0.0714633
750000	0.077221
800000	0.0821709
850000	0.0892656
900000	0.0937267
950000	0.0999896
1000000	0.105625
1050000	0.113057
1100000	0.115437
1150000	0.120105
1200000	0.12884
1250000	0.133323

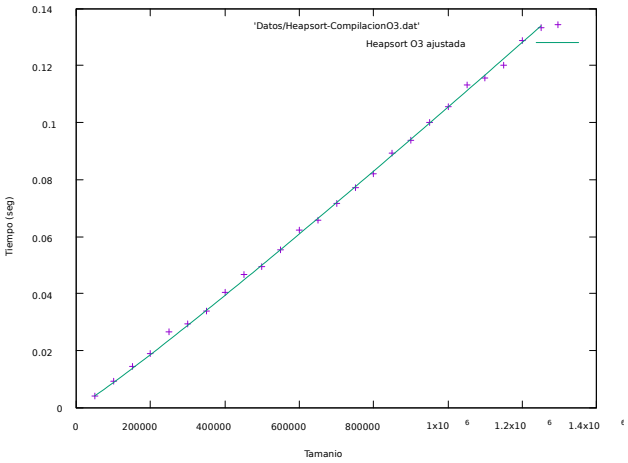
Comparación Compilaciones(-O/-O3)



Ajuste(para O3): f(x) = a0*x*log(x)

a0 = 7.62873e-09

Ajuste de la función a los datos



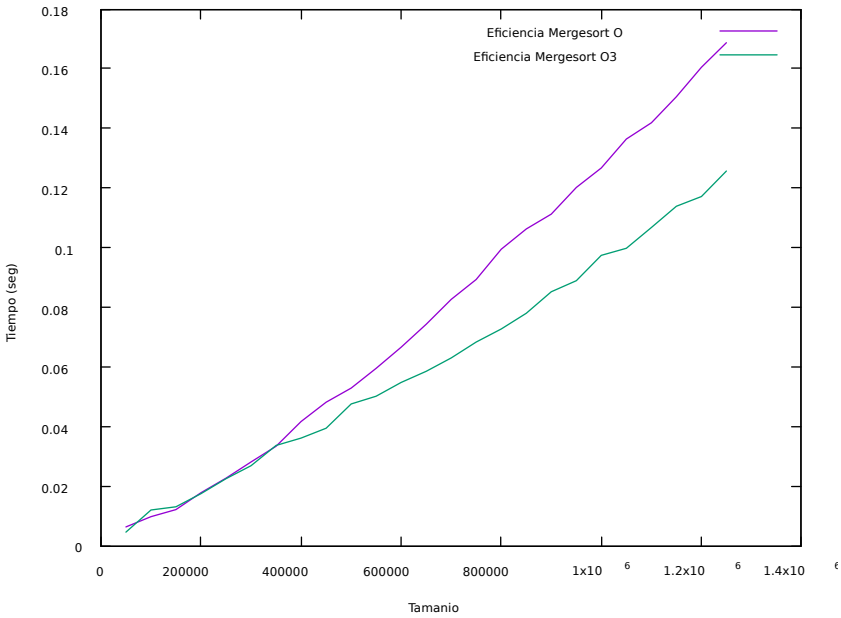
Jaime Corzo Galdó, Grupo 1

Mergesort:

Datos Mergesort(con Compilación -O3)

Tamaño	Tiempo(s)
50000	0.004677
100000	0.01212
150000	0.013185
200000	0.017535
250000	0.022604
300000	0.026936
350000	0.033745
400000	0.036199
450000	0.039482
500000	0.047619
550000	0.050235
600000	0.054846
650000	0.058605
700000	0.06306
750000	0.068407
800000	0.072761
850000	0.078059
900000	0.08521
950000	0.088971
1000000	0.09744
1050000	0.099809
1100000	0.106771
1150000	0.113897
1200000	0.117135
1250000	0.125697

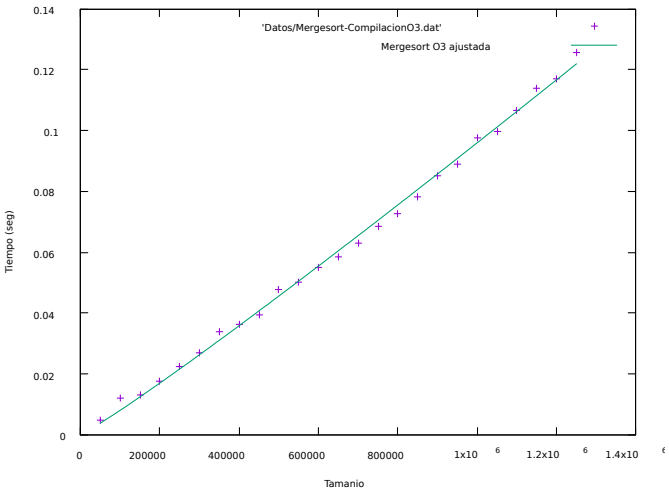
Comparación Compilaciones(-O/-O3)



Ajuste(para O3): f(x) = a0*x*log(x)

a0 = 6.94606e-09

Ajuste de la función a los datos



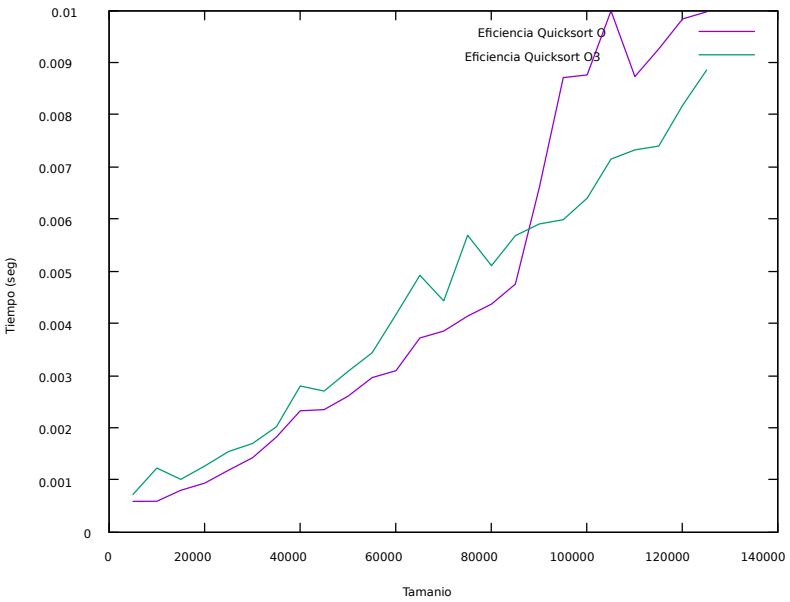
Jaime Corzo Galdó, Grupo 1

Quicksort:

Datos Quicksort(con Compilación -O3)

Tamaño	Tiempo(s)
5000	0.000589044
10000	0.000591014
15000	0.000801088
20000	0.000938625
25000	0.00118792
30000	0.00142354
35000	0.00182121
40000	0.00232583
45000	0.00234804
50000	0.00260423
55000	0.0029616
60000	0.00309314
65000	0.00371995
70000	0.00385246
75000	0.00413894
80000	0.00436809
85000	0.00475222
90000	0.00660845
95000	0.00870845
100000	0.00876012
105000	0.00998664
110000	0.00872876
115000	0.00925808
120000	0.00983499
125000	0.00996858

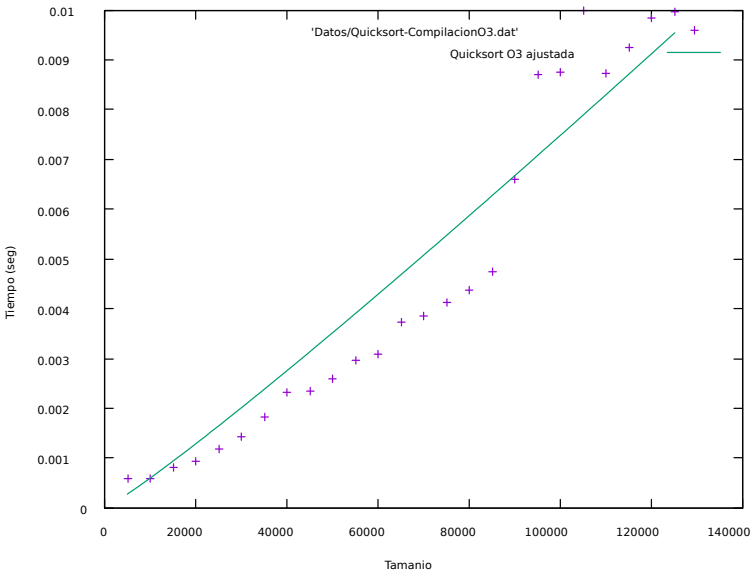
Comparación Compilaciones(-O/-O3)



Ajuste(para O3): f(x) = a0*x*log(x)

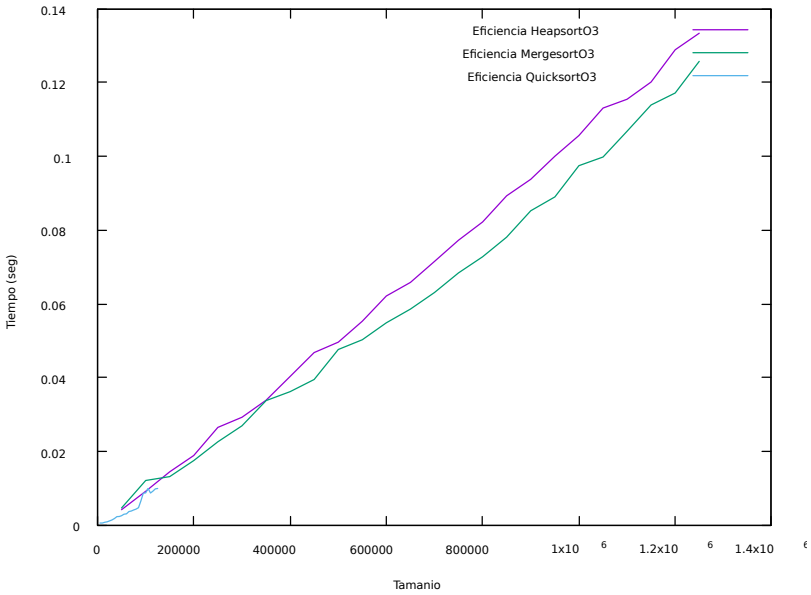
a0 = 6.50325e-09

Ajuste de la función a los datos



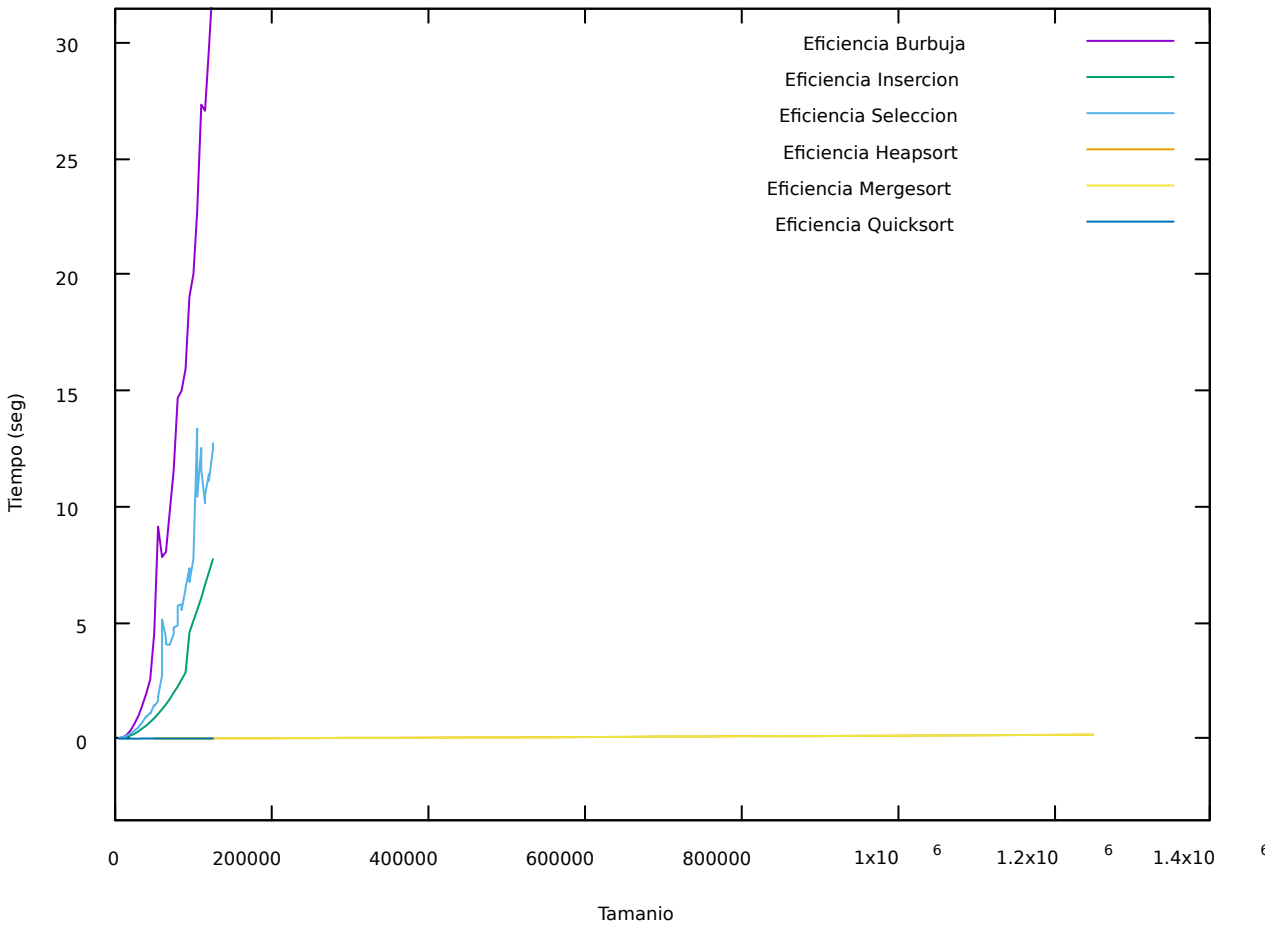
Jaime Corzo Galdó, Grupo 1

Comparación de los tres(con Compilación -O3):



3. Estudio comparativo de todos los algoritmos:

Podemos ver que los algoritmos con eficiencia O(n*log(n)) son más rápidos que los de eficiencia O(n^2), como esperábamos, además de la relación que ya habíamos comentado entre ellos



Jaime Corzo Galdó, Grupo 1

4. Conclusiones

- El análisis teórico es importante para ver si merece la pena implementar un algoritmo, y el análisis empírico e híbrido es la manera de confirmar nuestro análisis teórico.
- Al compilar los mismos algoritmos con una optimización mayor se comprueba que aumenta su eficiencia .
- Se evidencia la importancia de la notación O para elegir que algoritmo usar, especialmente en tamaños grandes.
- Para datos grandes los algoritmos con eficiencia $O(n^2)$ son mucho menos eficientes que los algoritmos O(n*log(n)), para datos pequeños la diferencia no es tan grande.
- De los cuadráticos el algoritmo más eficiente es el de inserción.
- En cambio el burbuja para tamaños grandes se dispara(es mejor evitarlo).
- Los tres algoritmos con O(n*log(n)), (en su caso intermedio), hemos comprobado en la práctica que su eficiencia es similar.
- El Quicksort en su caso peor tiene eficiencia de $O(n^2)$.
- Inserción y burbuja en su mejor caso pasan a ser lineales.
- Si trabajamos en el Quicksort con datos muy grandes nos va a dar core dumped, (al hacer muchas llamadas recursivas tendríamos una pila con muchos niveles).