Los tres algoritmos : mergesort, quicksort y heapsort son algoritmos de ordenación de vectores recursivos.

Calculo de la Eficiencia Empírica - Mergesort

Para ello medimos los recursos empleados (tiempo) para cada tamaño dado de las entradas. En este caso el tamaño viene dado por el número de componentes del vector a ordenar.

Para nuestro caso hemos utilizado valores de n a empezar en 1000 y a terminar en 100000 (muestras tomada de 1000 en 1000).

Para mostrar la eficiencia empírica usamos tablas para recoger el tiempo invertido para cada caso y también de gráficos.

En anexo siguen las tablas determinadas.

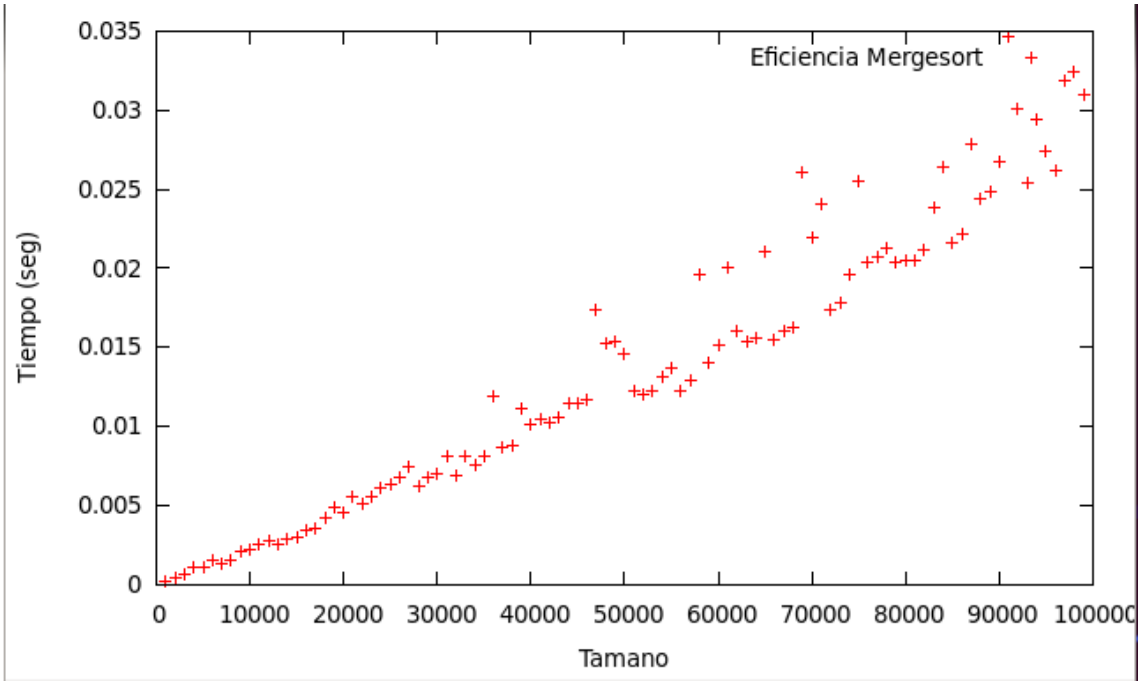

Figure 1- Gráfico Eficiencia del algoritmo Mergesort

Calculo de la Eficiencia Híbrida - Mergesort

Para el siguiente estudio he utilizado **gnuplot> f(x)=a0*x+a1*x*log(x)** en gnuplot y con la gráfica que se genera podemos comprobar como se produce un ajuste **muy aproximado**, esto nos dice que la información teórica y las pruebas empíricas coinciden.

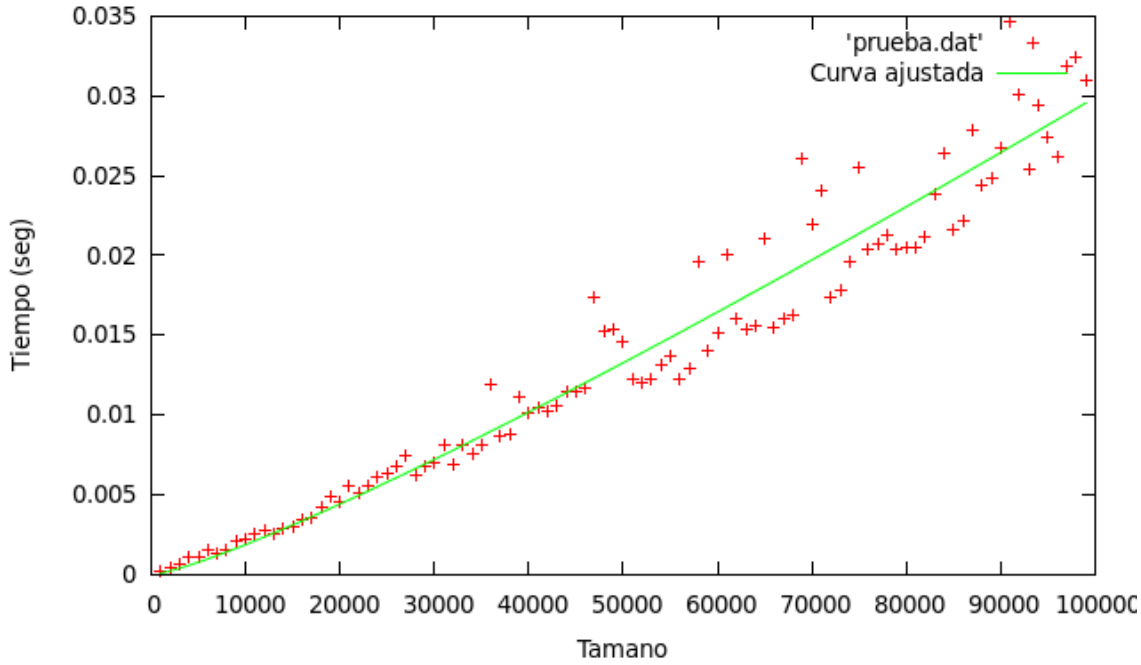

Figure 2-Curva Ajustada del Mergesort

variance of residuals (reduced chisquare) = WSSR/ndf : 4.45026e-06

Final set of parameters Asymptotic Standard Error

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

a0 = -2.61613e-07 +/- 1.235e-07 (47.2%)

a1 = 4.87065e-08 +/- 1.104e-08 (22.68%)

Por ejemplo, podemos estimar usando esa función ajustada que el tiempo de ejecución del algoritmo para una entrada de tamaño n = 100000 sería de f(100000) = 0,0299093segundos.

Otro aspecto interesante a analizar mediante este tipo de estudio es la variación de la eficiencia empírica en función de parámetros externos tales como: la opción de compilación utilizada (con optimización): **g++ -O2 -o fichero fichero.cpp**

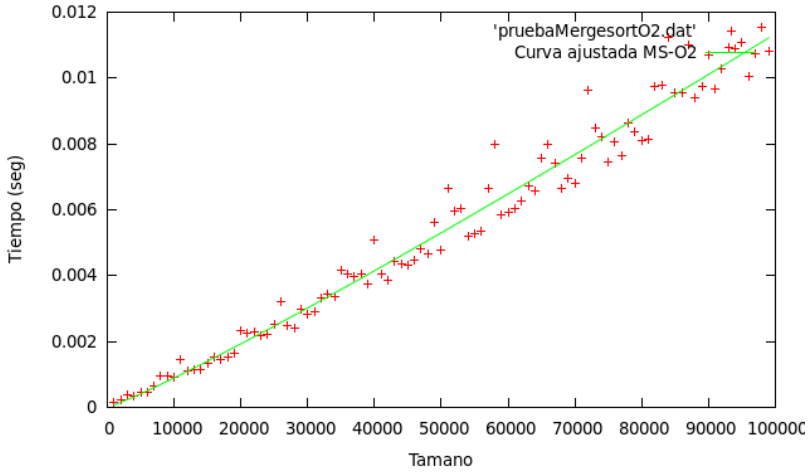

Figure 3- Curva ajustada del Mergesort con uso de compilación optimizada.

Se observa que el tiempo de ejecución con la compilación optimizada ha sido mejor: T(100000)≤0,012 segundos. Un 2,492 mejor que el tiempo de ejecución sin la optimización.

Mergesort se ejecuta en un tiempo de O(nlogn) en el peor caso, donde n es el tamaño del array a ordenar. La cantidad de comparaciones realizadas es cerca de óptima. Es un ejemplo bueno de algoritmo recursivo.

Calculo de la Eficiencia Empírica – Quicksort

Para nuestro caso hemos utilizado valores de n a empezar en 1000 y a terminar en 100000 (muestras tomada de 1000 en 1000).

Para mostrar la eficiencia empírica usamos tablas para recoger el tiempo invertido para cada caso y también de gráficos.

En anexo siguen las tablas determinadas.

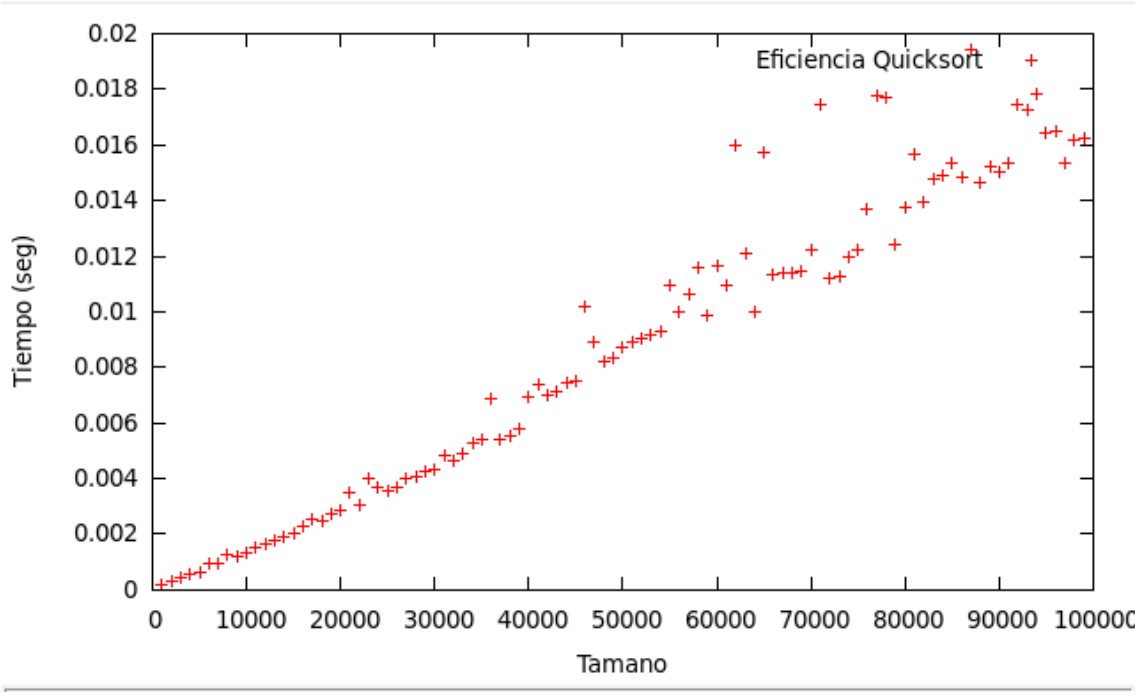

Figure 4 - Gráfico Eficiencia del algoritmo Quicksort

Calculo de la Eficiencia Híbrida - Quicksort

Para el siguiente estudio he utilizado **gnuplot> f(x)=a0*x+a1*x*log(x)** en gnuplot y con la gráfica que se genera podemos comprobar como se produce un ajuste **muy aproximado**, esto nos dice que la información teórica y las pruebas empíricas coinciden.

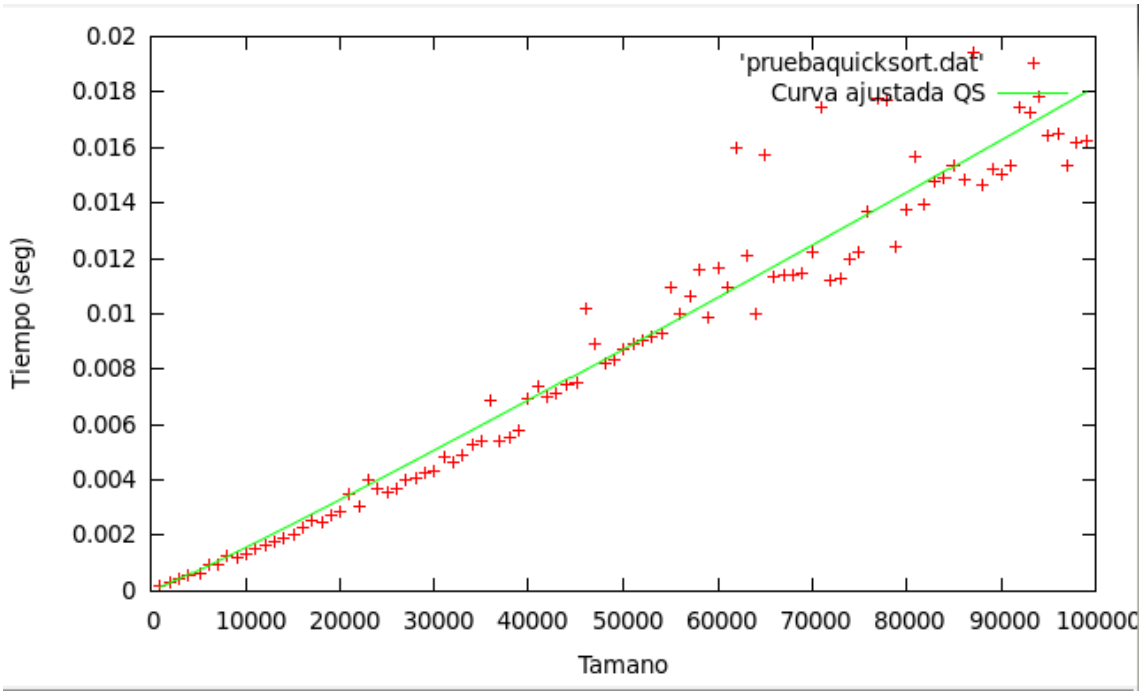

Figure 5 - Curva Ajustada del Quicksort

Haciendo la determinación de los parámetros a0 y a1:

gnuplot> f(x)=a0*x+a1*x*log(x)

gnuplot> fit f(x) 'prueba.dat' via a0,a1

Obtenemos alguna información relevante:

variance of residuals (reduced chisquare) = WSSR/ndf : 1.72421e-06

Final set of parameters Asymptotic Standard Error

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

a0 = 5.91681e-08 +/- 7.686e-08 (129.9%)

a1 = 1.0663e-08 +/- 6.875e-09 (64.47%)

Por ejemplo, podemos estimar usando esa función ajustada que el tiempo de ejecución del algoritmo para una entrada de tamaño n = 100000 sería de f(100000) = 0,018193segundos.

Otro aspecto interesante a analizar mediante este tipo de estudio es la variación de la eficiencia empírica en función de parámetros externos tales como: la opción de compilación utilizada (con optimización): **g++ -O2 -o fichero fichero.cpp**

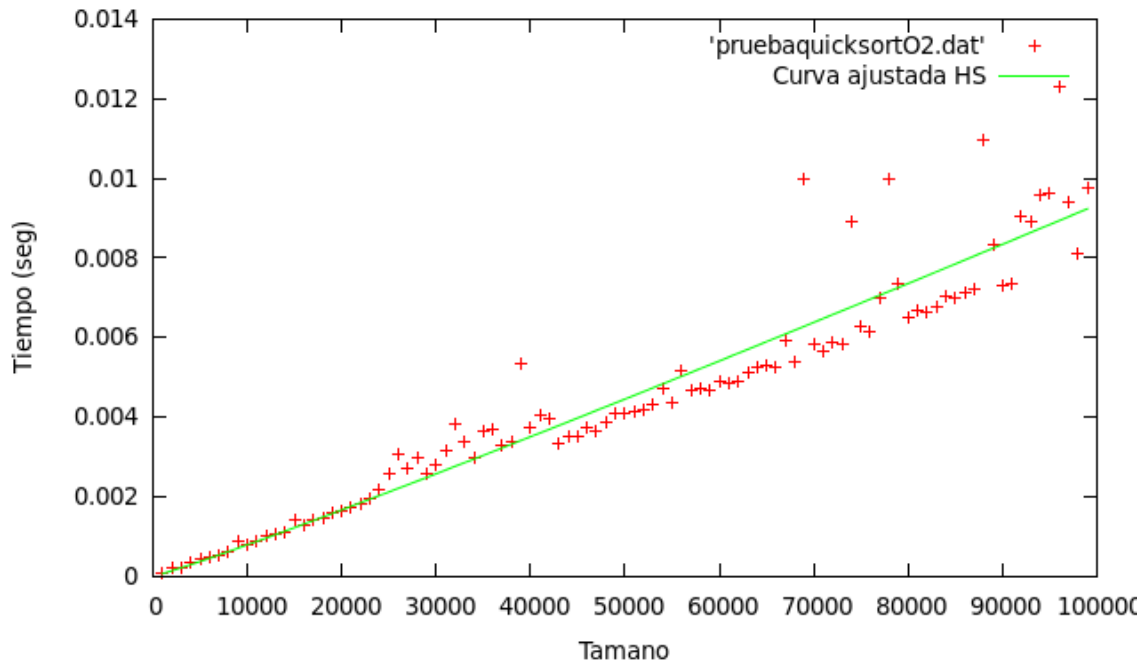

Figure 6 -Curva ajustada del Quicksort con uso de compilación optimizada.

Se observa que el tiempo de ejecución con la compilación optimizada ha sido mejor: T(100000)≤0,01 segundos. Un 1,8193 mejor que el tiempo de ejecución sin la optimización.

Calculo de la Eficiencia Empírica – Heapsort

Para nuestro caso hemos utilizado valores de n a empezar en 1000 y a terminar en 100000 (muestras tomada de 1000 en 1000).

Para mostrar la eficiencia empírica usamos tablas para recoger el tiempo invertido para cada caso y también de gráficos.

En anexo siguen las tablas determinadas.

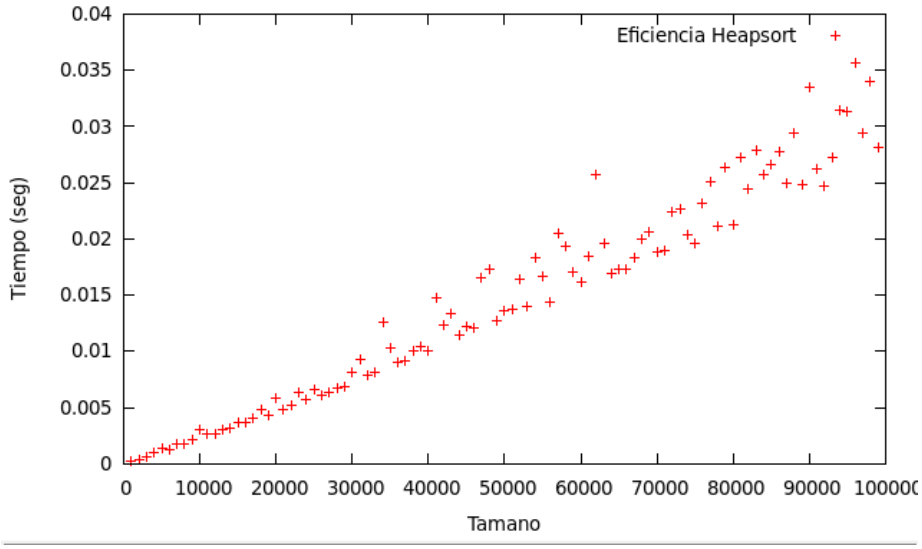

Figure 7-Gráfico Eficiencia del algoritmo Heapsort

Calculo de la Eficiencia Híbrida - Heapsort

Para el siguiente estudio he utilizado **gnuplot> f(x)=a0*x+a1*x*log(x)** en gnuplot y con la gráfica que se genera podemos comprobar como se produce un ajuste **muy aproximado**, esto nos dice que la información teórica y las pruebas empíricas coinciden.

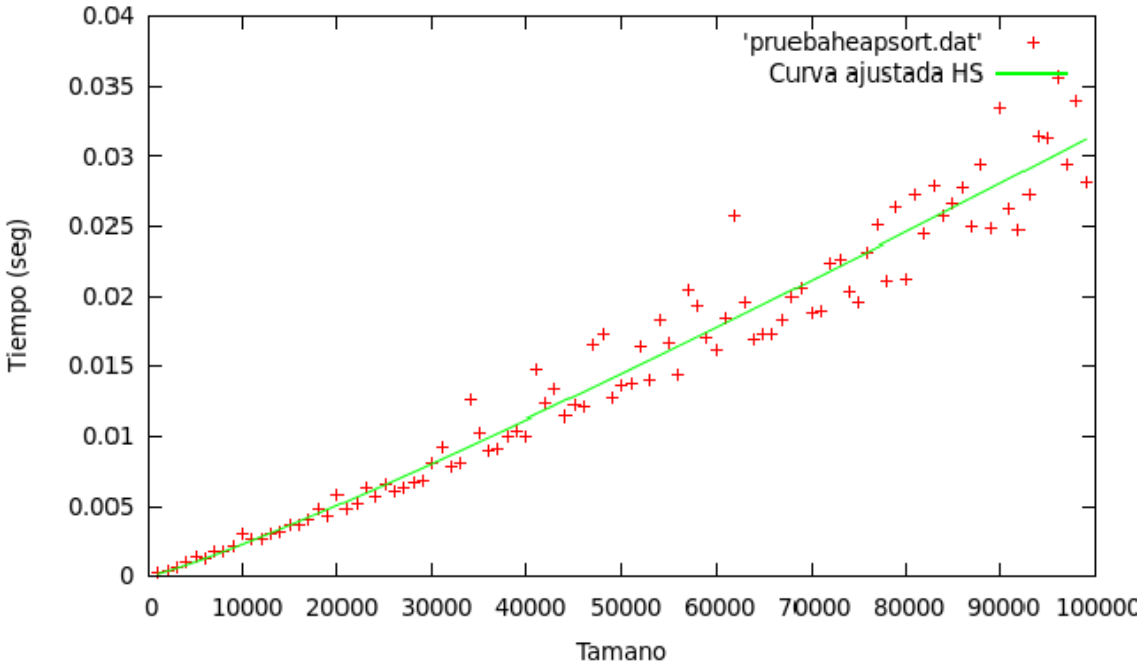

Figure 8 - Curva Ajustada Heapsort

Haciendo la determinación de los parámetros a0 y a1:

gnuplot> f(x)=a0*x+a1*x*log(x)

gnuplot> fit f(x) 'pruebaheapsort.dat' via a0,a1

Obtenemos alguna información relevante:

variance of residuals (reduced chisquare) = WSSR/ndf : 3.80271e-06

Final set of parameters Asymptotic Standard Error

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

a0 = -1.26563e-07 +/- 1.141e-07 (90.18%)

a1 = 3.84354e-08 +/- 1.021e-08 (26.56%)

Por ejemplo, podemos estimar usando esa función ajustada que el tiempo de ejecución del algoritmo para una entrada de tamaño n = 100000 sería de f(100000) = 0,032240segundos.

Otro aspecto interesante a analizar mediante este tipo de estudio es la variación de la eficiencia empírica en función de parámetros externos tales como: la opción de compilación utilizada (con optimización): **g++ -O2 -o fichero fichero.cpp**

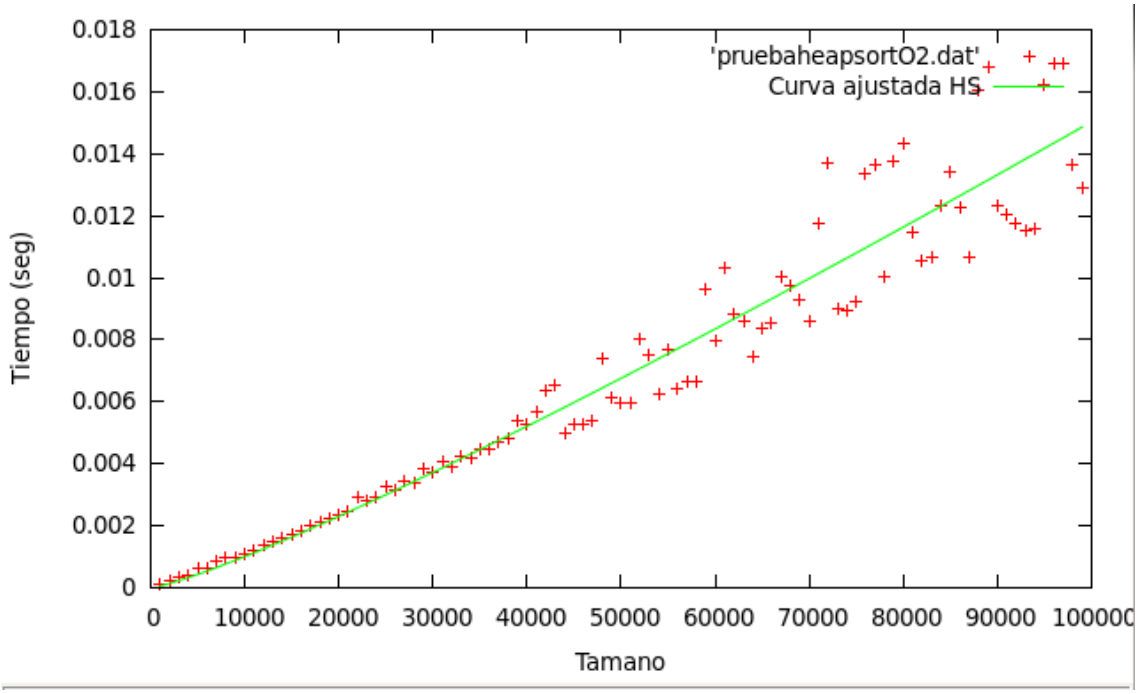

Figure 9-Curva ajustada del Heapsort con uso de compilación optimizada.

Se observa que el tiempo de ejecución con la compilación optimizada ha sido mejor: T(100000)≤0,016 segundos. Un 2,015 mejor que el tiempo de ejecución sin la optimización.

Comparación entre los tiempos de ejecución de los tres algoritmos O(nlogn) analisados (sin optimización):

El más rápido a ejecutar-se fue el Quicksort, en seguida el Mergesort y por último el Heapsort.

Anexos

Tabla de datos recogidos por implementación de los algoritmos.

Mergesort	Heapsort	Quicksort
n T(s)	n T(s)	n T(s)
1000 0.000175178	1000 0.000250926	1000 0.000170374
2000 0.000442308	2000 0.000402057	2000 0.000322886
3000 0.000633921	3000 0.000653078	3000 0.00041809
4000 0.0011282	4000 0.000999073	4000 0.000549007
5000 0.00113243	5000 0.0013462	5000 0.000630285
6000 0.00150932	6000 0.00124487	6000 0.000969374
7000 0.00137207	7000 0.00182228	7000 0.000946149
8000 0.00154617	8000 0.0017346	8000 0.0012776
9000 0.00206508	9000 0.00214404	9000 0.00117494
10000 0.00216235	10000 0.00305577	10000 0.00134822
11000 0.00249995	11000 0.00266762	11000 0.00150661
12000 0.00279164	12000 0.00264154	12000 0.00162207
13000 0.00251344	13000 0.00304628	13000 0.00176468
14000 0.00289217	14000 0.00312263	14000 0.00189339
15000 0.00299122	15000 0.00364498	15000 0.00203562
16000 0.00342269	16000 0.0036175	16000 0.00230065
17000 0.00356351	17000 0.00410623	17000 0.00256544
18000 0.00420246	18000 0.004766	18000 0.00247717
19000 0.00486726	19000 0.00434456	19000 0.00271923
20000 0.00450186	20000 0.00585006	20000 0.00283952
21000 0.00551136	21000 0.004869	21000 0.00347014
22000 0.00510252	22000 0.00523208	22000 0.00307753
23000 0.00550632	23000 0.0063121	23000 0.00401545
24000 0.00607925	24000 0.00567988	24000 0.00369443
25000 0.00635277	25000 0.00664691	25000 0.00356303
26000 0.00673021	26000 0.0061219	26000 0.00370557
27000 0.00748844	27000 0.0063834	27000 0.00398112
28000 0.00626344	28000 0.00670571	28000 0.00405798
29000 0.0067992	29000 0.00689368	29000 0.00428009
30000 0.00702867	30000 0.0081659	30000 0.00434262
31000 0.00808129	31000 0.0093012	31000 0.00480411
32000 0.00691551	32000 0.00789431	32000 0.00461417
33000 0.00809563	33000 0.00808421	33000 0.00491369
34000 0.00759279	34000 0.0125616	34000 0.00530059
35000 0.0081486	35000 0.0102412	35000 0.00543509
36000 0.011933	36000 0.00903575	36000 0.00686069
37000 0.00863342	37000 0.00918761	37000 0.00542906
38000 0.00883814	38000 0.0100057	38000 0.00556151
39000 0.0111385	39000 0.0104511	39000 0.0057731
40000 0.0101272	40000 0.0100597	40000 0.0069592
41000 0.0104254	41000 0.0147371	41000 0.00736244
42000 0.0102505	42000 0.0123688	42000 0.00698552

43000 0.0105821	43000 0.0134109	43000 0.007114
44000 0.0114274	44000 0.0113879	44000 0.00744926
45000 0.0114185	45000 0.0121787	45000 0.00750141
46000 0.0116659	46000 0.0120745	46000 0.0101614
47000 0.017348	47000 0.0165896	47000 0.00893694
48000 0.0152724	48000 0.0173401	48000 0.00818758
49000 0.0153457	49000 0.012671	49000 0.00833376
50000 0.0145559	50000 0.0136326	50000 0.00871449
51000 0.0122475	51000 0.0137016	51000 0.00890256
52000 0.0120474	52000 0.0164808	52000 0.0090708
53000 0.0122787	53000 0.014016	53000 0.00914137
54000 0.013172	54000 0.0183499	54000 0.00929994
55000 0.013723	55000 0.0166612	55000 0.0109753
56000 0.012289	56000 0.0143278	56000 0.0100007
57000 0.0128762	57000 0.0205238	57000 0.0106584
58000 0.0195736	58000 0.0193324	58000 0.0115674
59000 0.0140501	59000 0.0170726	59000 0.00987749
60000 0.015113	60000 0.0161459	60000 0.011645
61000 0.0200164	61000 0.018482	61000 0.0109535
62000 0.0160787	62000 0.025749	62000 0.0160042
63000 0.0153156	63000 0.0195826	63000 0.0120888
64000 0.0155544	64000 0.016925	64000 0.00997022
65000 0.0210374	65000 0.0172974	65000 0.0157133
66000 0.0154368	66000 0.0172647	66000 0.0113379
67000 0.0159868	67000 0.018275	67000 0.0113999
68000 0.0162406	68000 0.0199835	68000 0.0114216
69000 0.0260927	69000 0.0206701	69000 0.0114788
70000 0.0219008	70000 0.0188052	70000 0.0122216
71000 0.0241246	71000 0.0189669	71000 0.0174474
72000 0.0173413	72000 0.0224462	72000 0.0111894
73000 0.017877	73000 0.0226306	73000 0.0112718
74000 0.019653	74000 0.0203651	74000 0.0119711
75000 0.0255592	75000 0.0196607	75000 0.0122557
76000 0.0204343	76000 0.023118	76000 0.0137194
77000 0.0207364	77000 0.0251172	77000 0.0177631
78000 0.021281	78000 0.0211488	78000 0.0176811
79000 0.0203644	79000 0.0264081	79000 0.0124247
80000 0.0205175	80000 0.0212095	80000 0.013762
81000 0.0204872	81000 0.0272433	81000 0.0157014
82000 0.0211638	82000 0.0244121	82000 0.0139424
83000 0.0238633	83000 0.0278484	83000 0.0148034
84000 0.0263793	84000 0.0257143	84000 0.0149172
85000 0.0215608	85000 0.0266066	85000 0.0153769
86000 0.0221625	86000 0.0277125	86000 0.01487
87000 0.0279117	87000 0.0249807	87000 0.0194087
88000 0.024428	88000 0.029476	88000 0.0146585
89000 0.0248304	89000 0.0247789	89000 0.0152167
90000 0.0267225	90000 0.0334886	90000 0.0150106
91000 0.0346341	91000 0.0262223	91000 0.0153575

92000 0.0300427	92000 0.0247547	92000 0.0174473
93000 0.0253986	93000 0.0271967	93000 0.0172858
94000 0.0294273	94000 0.0314156	94000 0.0178208
95000 0.0274326	95000 0.0313135	95000 0.0164283
96000 0.0261405	96000 0.0357141	96000 0.0165204
97000 0.031828	97000 0.0293845	97000 0.0153487
98000 0.0324402	98000 0.0340577	98000 0.0162114
99000 0.0310407	99000 0.028154	99000 0.0162525