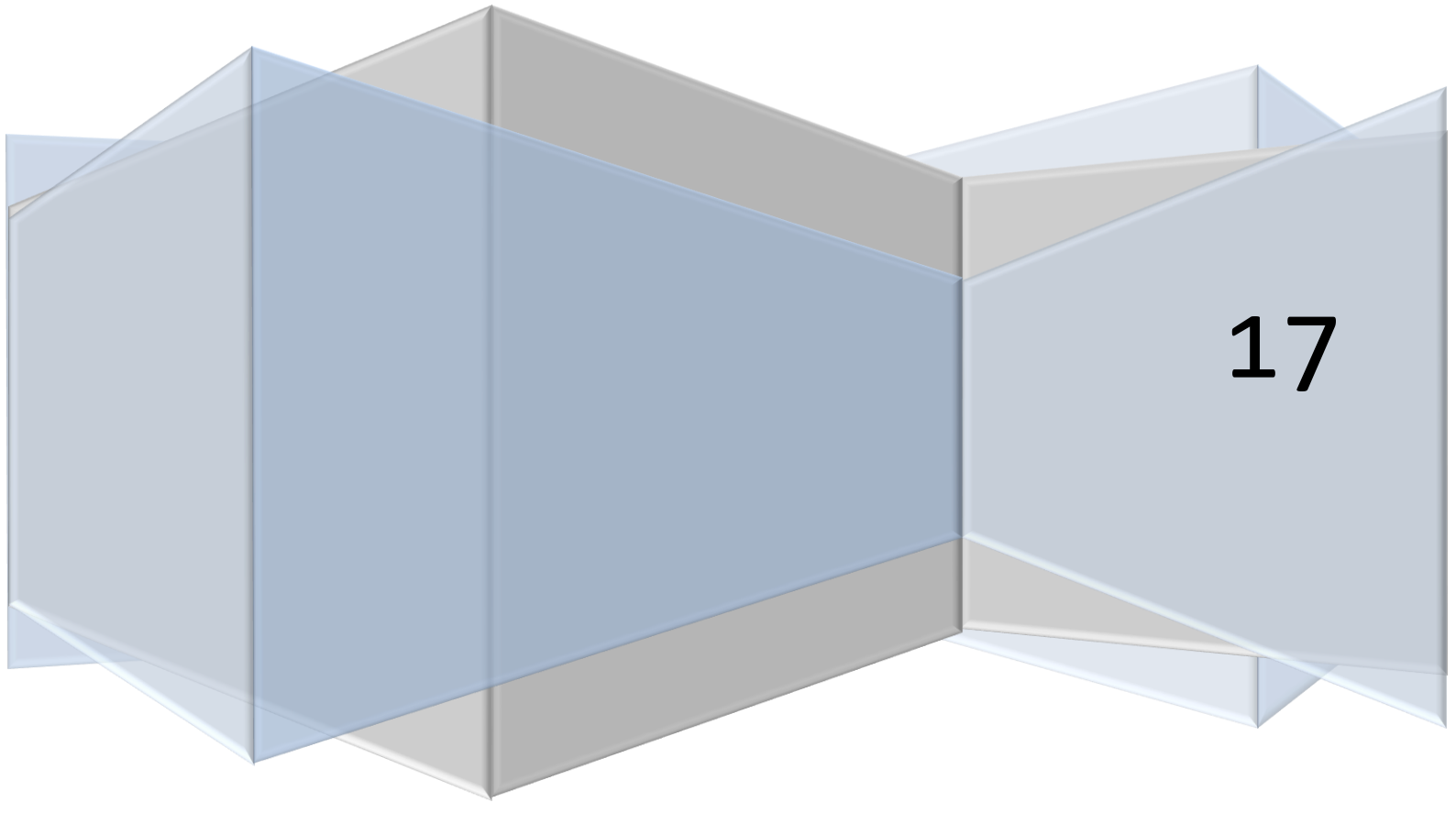
Práctica 1 - Eficiencia

Integrantes:

Guillermo Gómez Trenado, Miguel Ángel Rispal Martínez, Juan Carlos Ruiz García, Ignacio Irurita Contreras, Joaquín Fernández León, Daniel García Martos.



Contenido

Ejercicio 1. Calcule la eficiencia empírica siguiendo las indicaciones de la sección 3.

Para realizar la comparación entre algoritmos de distinto orden utilizaremos los algoritmos de ***Burbuja, Hanoi, Floyd y Quicksort***.

Para tomar los diferentes tiempos hemos aumentado el número de entradas de **100 en 100 hasta** un número de entradas máximo de **3900**. En el algoritmo de *Hanoi* hemos hecho una excepción. Al ser un algoritmo exponencial, hemos incrementado de **1 en 1 hasta** un número de entradas igual a **37**, puesto que era aquí donde empezaba a necesitar un tiempo muy considerable para su ejecución. Una vez tomados los tiempos se ha creado la siguiente tabla a partir de los resultados obtenidos con la **función *time*** que ofrece C++.

Hanoi (2^n)		Floyd (n^3)		Burbuja (n^2)	QuickSort ($n*\log_2(n)$)
Iteraciones	Tiempo (s)	**Iteraciones**	Tiempo (s)	Tiempo (s)	Tiempo (s)
1	1,00E-06	**100**	0,012076	3,90E-05	3,00E-05
2	1,00E-06	**200**	0,048968	0,000106	4,10E-05
3	1,00E-06	**300**	0,160928	0,000199	8,60E-05
4	2,00E-06	**400**	0,379565	0,000348	8,30E-05
5	2,00E-06	**500**	0,740481	0,000535	0,000105
6	2,00E-06	**600**	128.333	0,000866	0,000127
7	4,00E-06	**700**	203.369	0,001058	0,000194
8	6,00E-06	**800**	302.536	0,001424	0,000165
9	1,20E-05	**900**	426.672	0,001744	0,000175
10	1,70E-05	**1000**	59.076	0,002075	0,000269
11	4,10E-05	**1100**	791.481	0,002498	0,000223
12	5,80E-05	**1200**	102.998	0,002979	0,00023
13	0,000115	**1300**	132.095	0,003453	0,000267
14	0,00028	**1400**	166.996	0,003883	0,000298
15	0,000466	**1500**	265.484	0,004416	0,000305
16	0,000918	**1600**	26.768	0,005737	0,000307
17	0,001738	**1700**	362.819	0,005749	0,000459
18	0,003407	**1800**	360.257	0,00644	0,000505
19	0,006631	**1900**	420.754	0,007085	0,000473
20	0,013291	**2000**	491.758	0,008531	0,000429
21	0,021551	**2100**	575.416	0,008719	0,000585
22	0,034348	**2200**	711.056	0,009605	0,000413
23	0,058882	**2300**	745.454	0,011107	0,000598
24	0,115738	**2400**	954.837	0,011434	0,000613
25	0,229335	**2500**	104.571	0,012559	0,000463
26	0,459994	**2600**	103.162	0,0141	0,000457
27	0,987156	**2700**	122.192	0,015121	0,000508
28	1,87696	**2800**	146.007	0,015996	0,000748

29	3,62779	**2900**	156.621	0,01745	0,000683
30	7,35357	**3000**	172,58	0,018322	0,000533
31	14,5693	**3100**	185.113	0,020647	0,000558
32	29,1016	**3200**	202.439	0,021766	0,000524
33	58,2624	**3300**	223.837	0,022521	0,000748
34	116,03	**3400**	254.822	0,024245	0,000528
35	231,611	**3500**	261.728	0,026378	0,000524
36	455,499	**3600**	307,33	0,027872	0,000537
37	908,245	**3700**	321.716	0,029146	0,000738
		3800	341.295	0,030883	0,00058
		3900	372.199	0,032407	0,000597

Se puede apreciar que la diferencia de las distintas eficiencias aumenta cuanto mayor es el tamaño de entrada para cada algoritmo. Por ejemplo, en el algoritmo de *Burbuja* tenemos que para un tamaño de 3900 necesitará un tiempo de 0,032407 segundos pero en cambio en el de *Hanoi*, para un dato de 37 tarda un tiempo de 908,245; es decir del orden de aproximadamente 28.026.198 veces más. En los siguientes ejercicios compararemos de manera más exhaustiva

Ordenados de **mayor a menor** tiempo de ejecución quedarían de la siguiente manera: *Hanoi, Floyd, Burbuja* y *Quicksort.*

Ejercicio 2. Generar un gráfico comparando los tiempos de los algoritmos.

Con el programa **GNUPLOT** representaremos los anteriores valores en los correspondientes gráficos. El eje de la **X representará el número de entradas** y el **eje de la Y el tiempo en segundos**. Veamos los resultados:

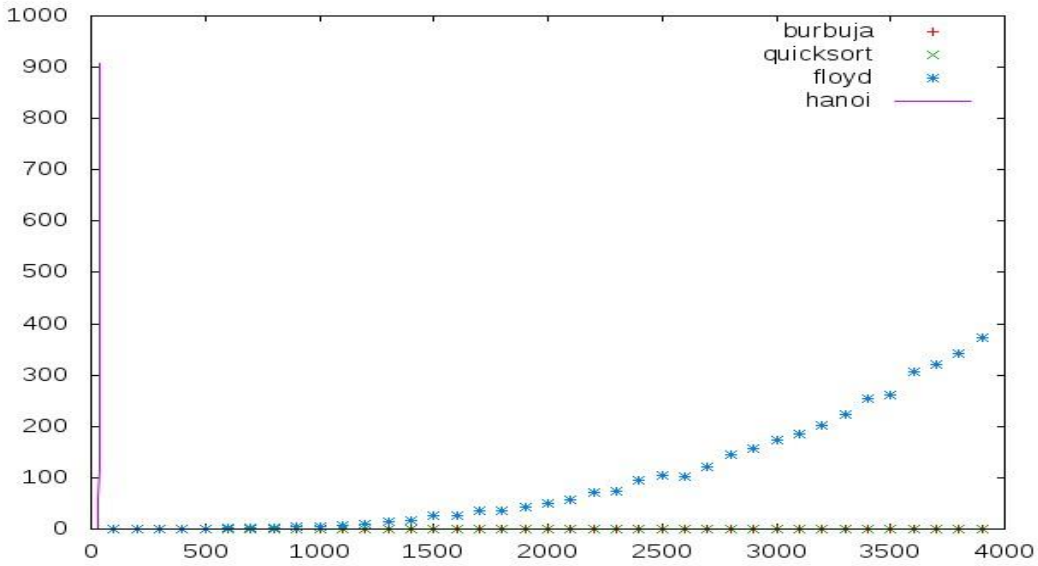


Imagen 1. Comparación de tiempos de los diferentes algoritmos (burbuja, quicksort, Floyd y hanoi).

Claramente se aprecia que ***Hanoi* es el algoritmo más lento** en ejecución como hemos comentado antes.

No obstante, veamos de manera empírica que sucederá con los **otros 6 algoritmos de ordenación**:

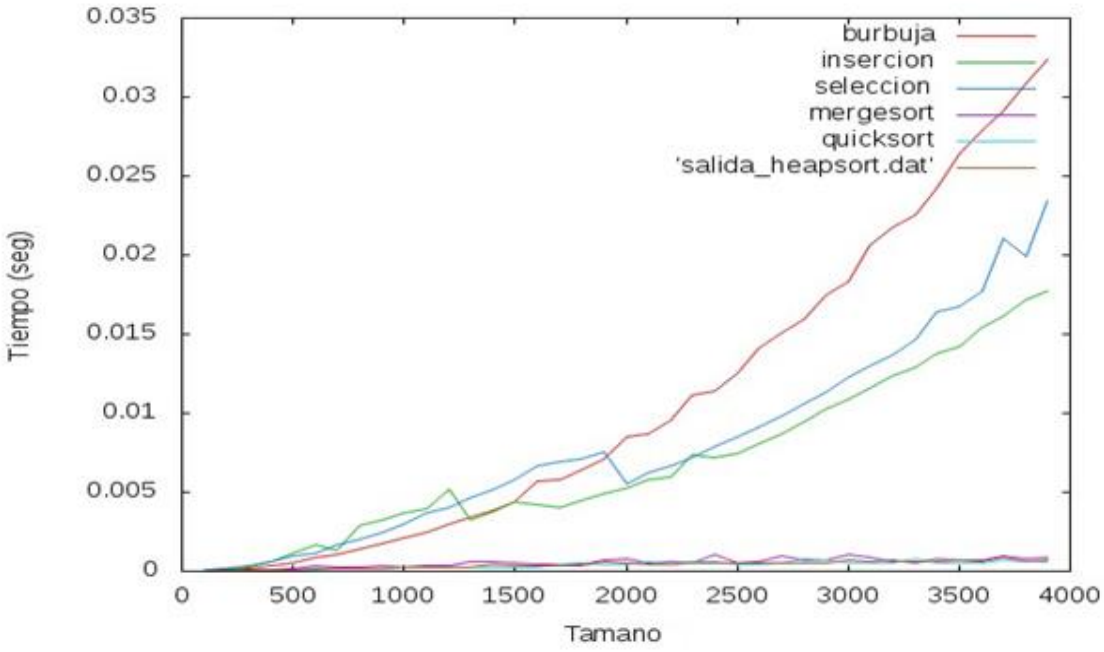


Imagen 2. Comparación de tiempos de todos los algoritmos.

Se aprecia claramente la diferencia entre el orden **O(nlogn) (heapsort, mergesort, quicksort)** y el **orden O(n^2) (burbuja, inserción y selección)**. En el siguiente apartado definiremos la eficiencia de cada algoritmo.

Ejercicio 3. Calcular eficiencia híbrida (nube de puntos).

Algoritmo de Floyd

A partir de los valores de tiempo obtenidos anteriormente de manera empírica y sabiendo que su cálculo teórico es de eficiencia de **O(n³)** podemos crear un enfoque híbrido. De esta manera, su función de regresión debería de ser:

$$f(n) = a0*x^3+a1*x^2+a2*x+a3$$

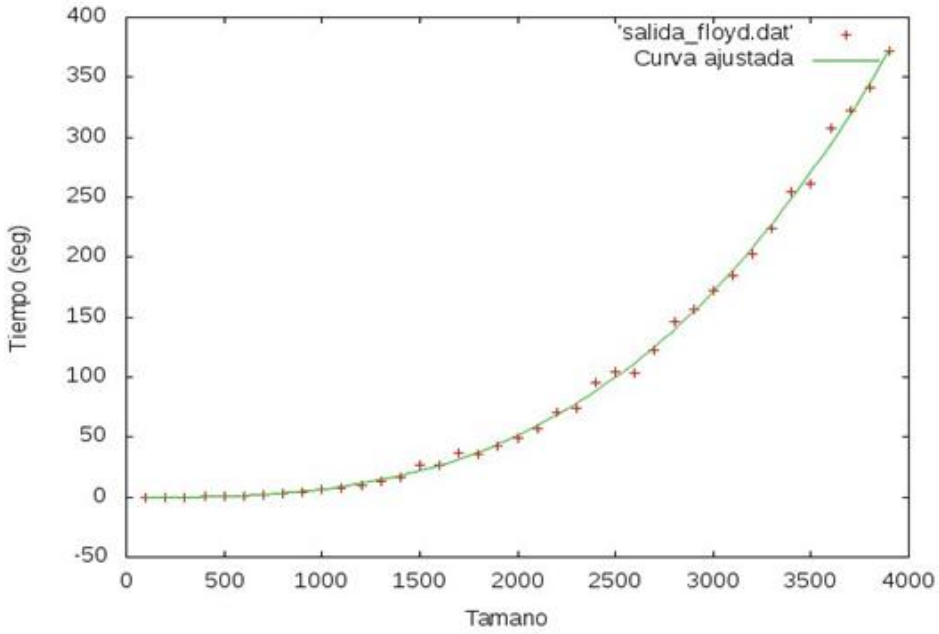


Imagen 3. Ajuste cuadrático del algoritmo de Floyd.

Si quisieramos ajustar el algoritmo de floyd con otro tipo de gráfica, por ejemplo una lineal, obtendríamos el siguiente ajuste:

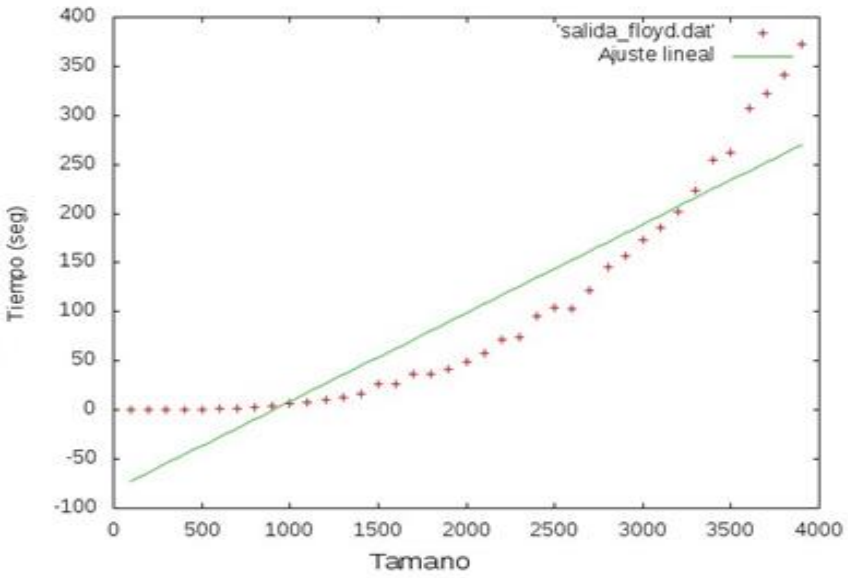


Imagen 4. Ajuste lineal del algoritmo de floyd.

Como podemos observar el ajuste lineal no es un buen ajuste. Para conseguir un ajuste preciso debemos hacer un ajuste de regresión que defina bien la función representada por los datos obtenidos empíricamente.

Algoritmo de Hanoi

A partir de los valores de tiempo obtenidos anteriormente de manera empírica y sabiendo que su cálculo teórico es de eficiencia de **$O(2^n)$** su función de regresión en este caso debería de ser:

$$f(n) = 2^{a0*n+a1}$$

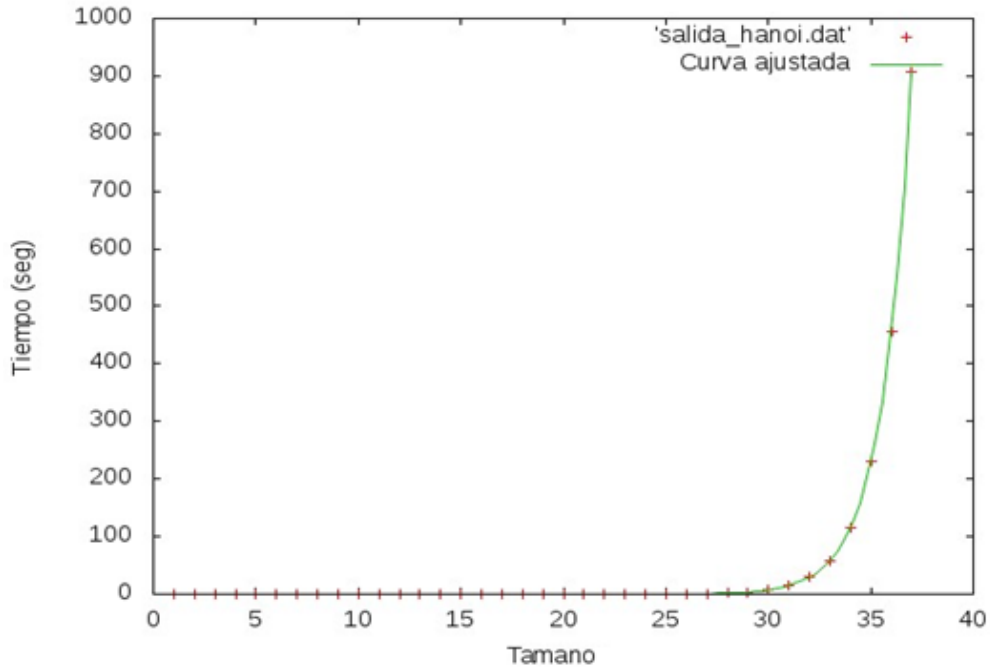


Imagen 5. Ajuste del algoritmo de Hanoi.

Algoritmos Heapsort, Quicksort y Mergesort

Al igual que antes, calculamos su función híbrida de orden **O(nlogn)**:

$$f(n) = a0*n*log(a1*n) + a2$$

Concretamente en las gráficas que hemos obtenido se muestran como una dispersión de puntos. Por lo que tenemos un ajuste algo peor con dichos datos. No obstante, **para conseguir una regresión más fiable necesitaríamos obtener muchísimos más puntos** y ver que hay en común para realizar el ajuste de regresión. Las gráficas obtenidas son las siguientes:

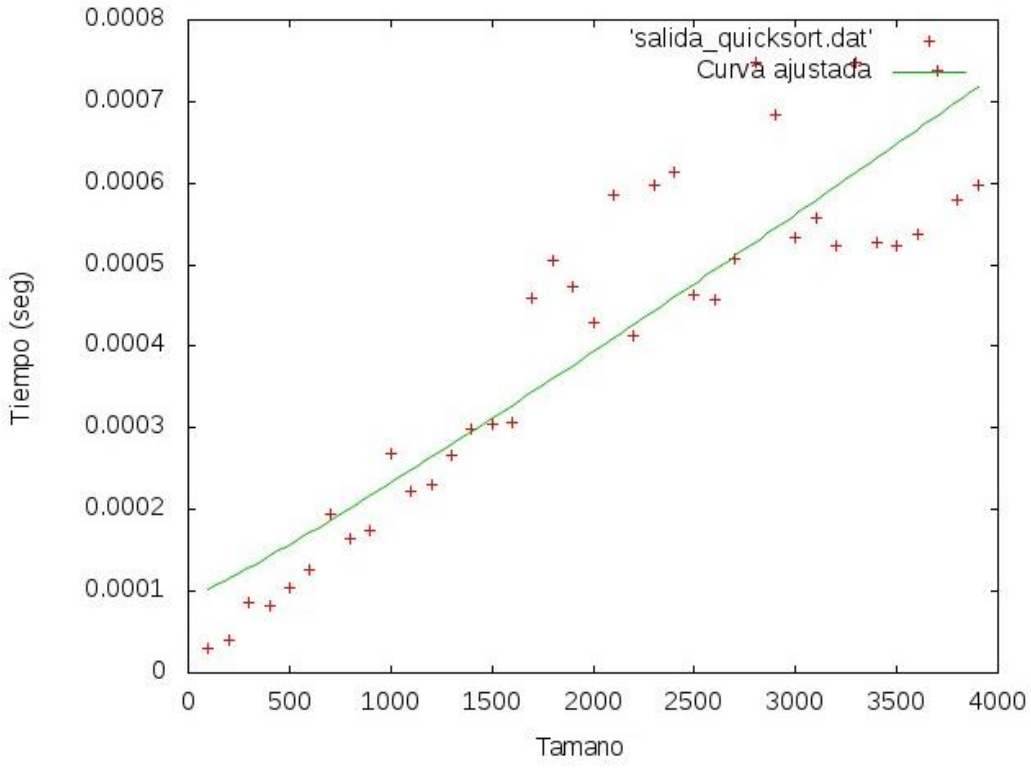


Imagen 6. Ajuste del algoritmo de QuickSort.

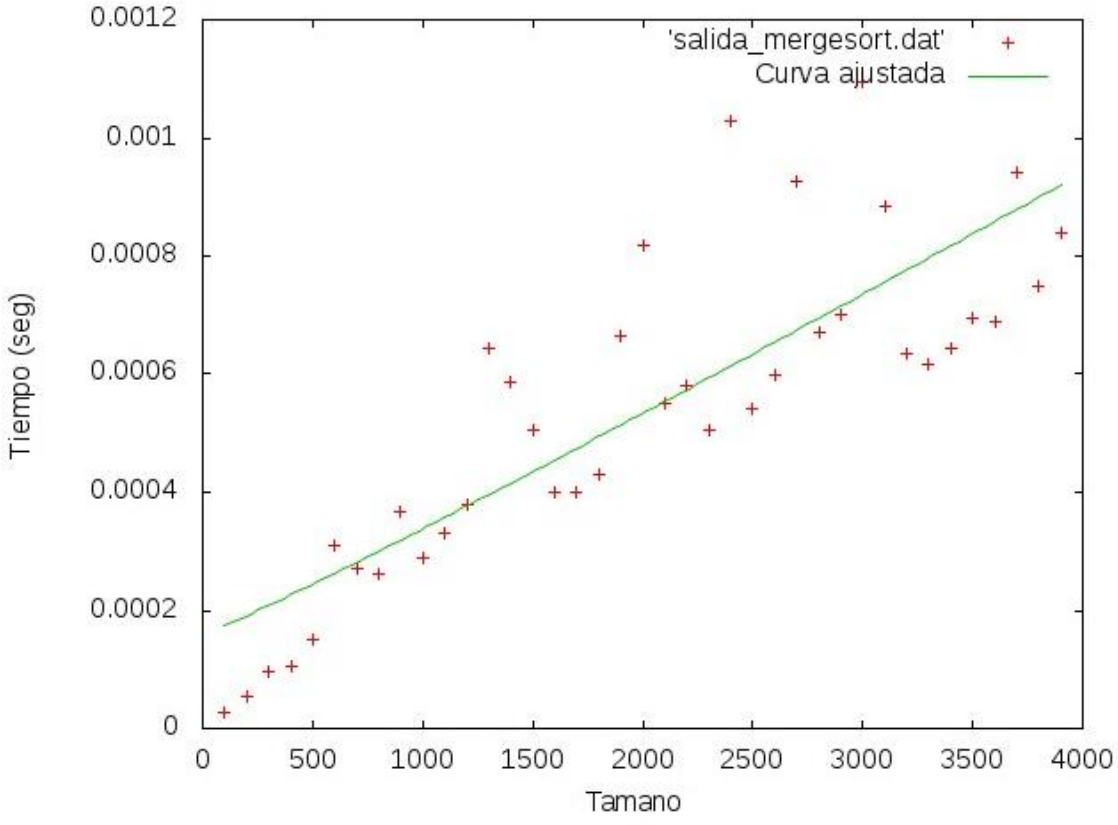


Imagen 7. Ajuste del algoritmo de MergeSort.

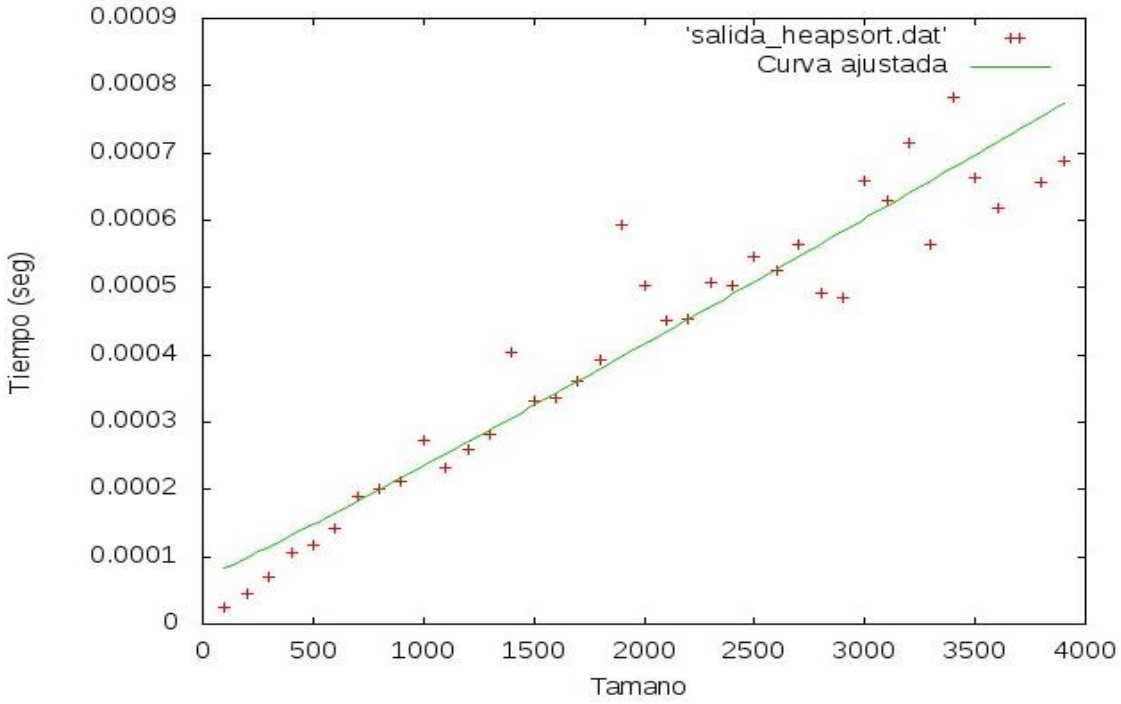


Imagen 8. Ajuste del algoritmo de HeapSort.

Algoritmo de inserción, burbuja y selección

Al igual que antes, calculamos su función híbrida cuadrática de orden **O(n²)**:

$$f(n) = a0*x^2+a1*x+a2$$

En estas gráficas se muestran algo menos de dispersión y las gráficas consiguen ajustarse mejor a los datos empíricos.

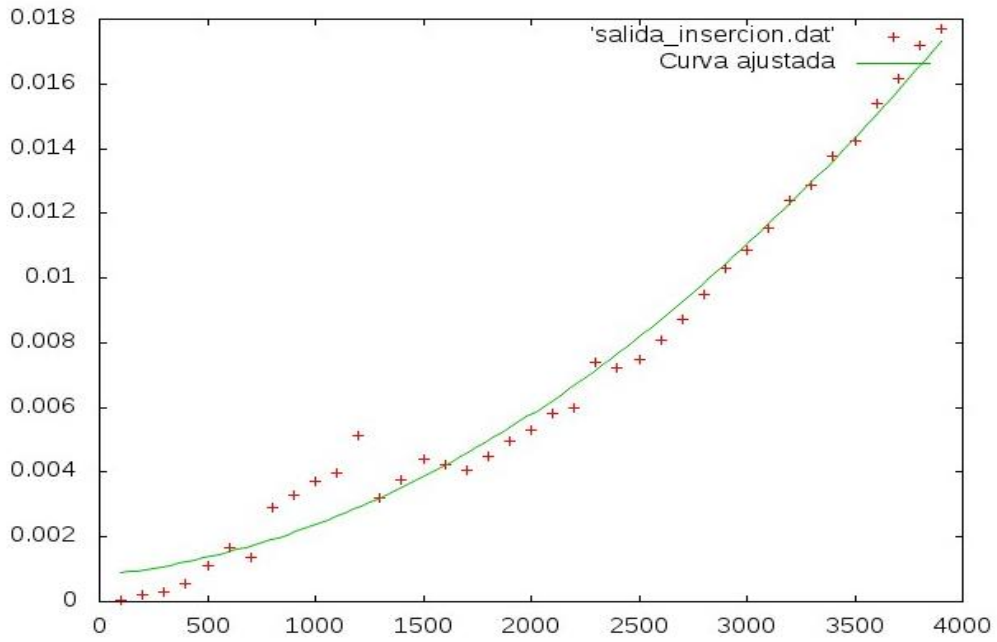


Imagen 9. Ajuste cuadrático del algoritmo de inserción.

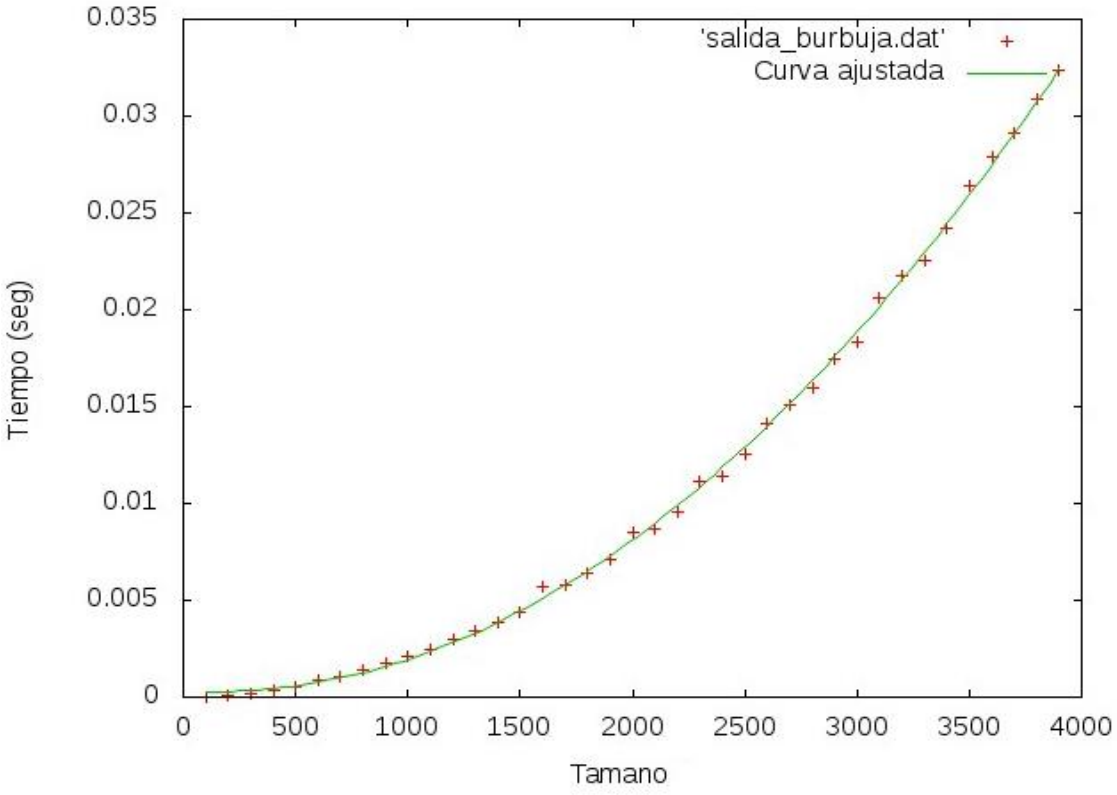


Imagen 10. Ajuste cuadrático del algoritmo de burbuja.

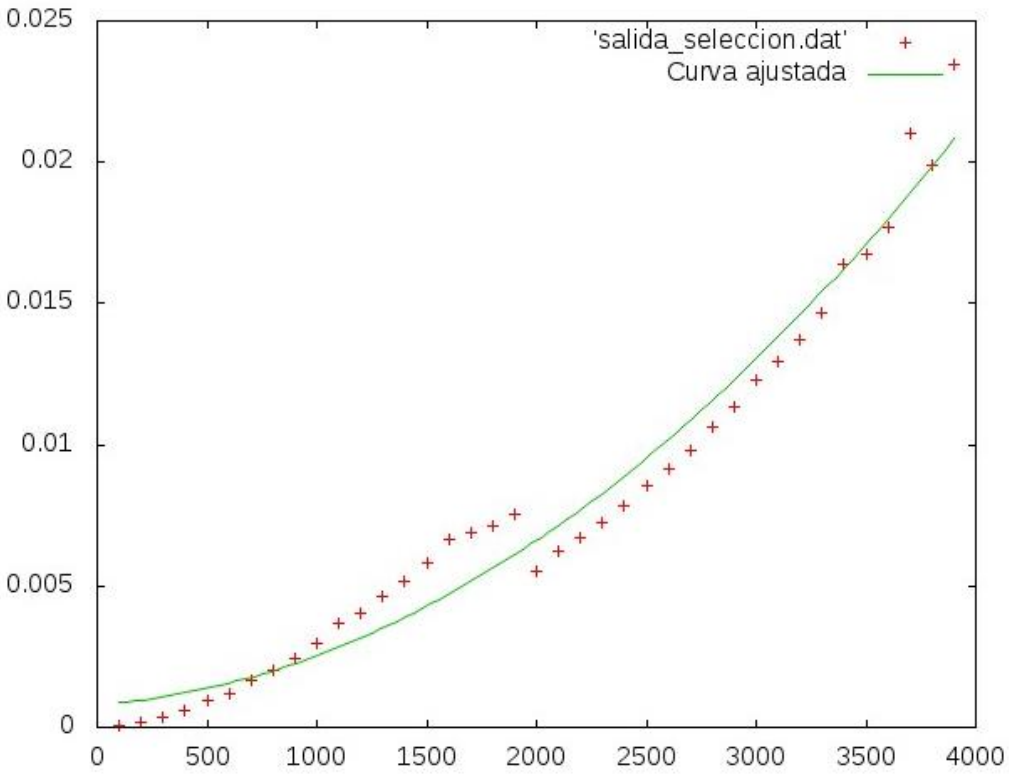


Imagen 11. Ajuste cuadrático del algoritmo de selección.

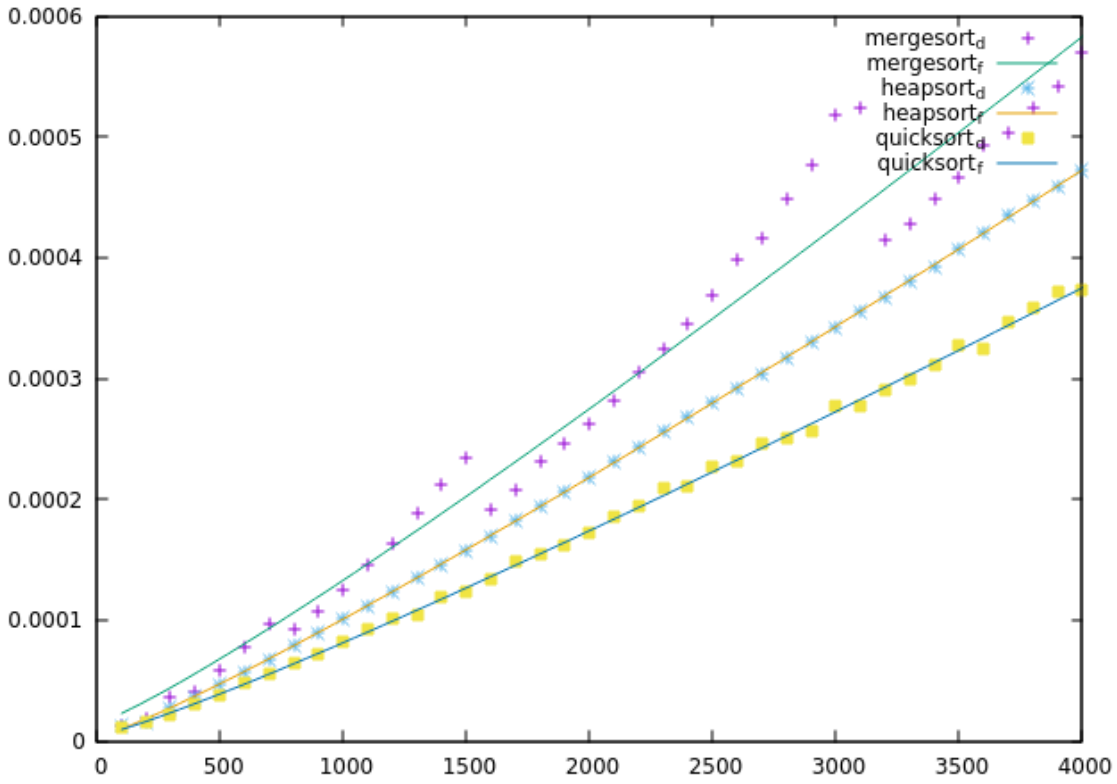
Ejercicio 3.1. Calcular eficiencia híbrida (mediana de 100 ejecuciones).

Anteriormente realizamos los ajustes sobre los algoritmos habiendo realizado **una sola ejecución** de estos.

Ahora para mejorar ese ajuste y evitar tiempos erróneos (debido a uso compartido del procesador, administración del sistema operativo, etc. durante la ejecución) hemos ejecutado **100 veces** cada uno de los **algoritmos de ordenación** (HeapSort, QuickSort, MergeSort) para posteriormente realizar la **mediana** de esas 100 ejecuciones y obtener un valor mucho más fiable.

Iteraciones	QuickSort	MergeSort	HeapSort
100	0.000011	0.000013	0.000012
200	0.000015	0.000019	0.000016
300	0.000022	0.000036	0.000027
400	0.00003	0.000041	0.000036
500	0.000038	0.000059	0.000047
600	0.000048	0.000078	0.000057
700	0.000056	0.000097	0.000068
800	0.000064	0.000092	0.000079
900	0.000072	0.000107	0.00009
1000	0.000082	0.000125	0.000101
1100	0.000092	0.000145	0.000112
1200	0.000101	0.000164	0.000124
1300	0.000105	0.000189	0.000135
1400	0.000119	0.000212	0.000146
1500	0.000123	0.000234	0.000157
1600	0.000134	0.000191	0.00017
1700	0.000148	0.000208	0.000182
1800	0.000155	0.000231	0.0001945
1900	0.000162	0.000246	0.000206
2000	0.000172	0.000263	0.000218
2100	0.000186	0.000282	0.000232
2200	0.000195	0.000305	0.000243
2300	0.000209	0.000324	0.000257
2400	0.000211	0.000345	0.000269
2500	0.000227	0.000369	0.00028
2600	0.000231	0.000398	0.000292
2700	0.000246	0.000416	0.000304
2800	0.00025	0.000448	0.000317
2900	0.000257	0.000477	0.00033
3000	0.000277	0.0005175	0.000343
3100	0.000277	0.0005245	0.000355
3200	0.00029	0.000414	0.000368
3300	0.000299	0.000428	0.000381
3400	0.000311	0.000448	0.000393
3500	0.000327	0.000467	0.000407
3600	0.000325	0.000493	0.00042
3700	0.000346	0.000504	0.000435
3800	0.000359	0.0005245	0.000447
3900	0.0003715	0.000542	0.000459

La gráfica que obtenemos al realizar este ajuste es mucho mejor ya que la mediana del resultado de las 100 ejecuciones se acerca mucho más al valor de la función.



Ejercicio 4. Variación de eficiencia empírica en función de la optimización.

En este caso hemos compilado **dos veces el mismo algoritmo con diferentes tipos de optimización**. En el gráfico *burbuja*, se ha compilado sin optimización y en *burbuja_o2,* hemos usado una optimización de -O2. Esta optimización lo que hace disminuir el número de instrucciones en ensamblador. Se nota claramente que sigue siendo de orden cuadrático pero los tiempos de ejecución para el mismo número de entradas han disminuido. Veamos a continuación los valores:

Burbuja		Burbuja -O2	
Iteraciones	Tiempo (s)	Iteraciones	Tiempo (s)
100	3.9e-05	**100**	3.5e-05
200	0.000106	**200**	0.000112
300	0.000199	**300**	0.000233
400	0.000348	**400**	0.000391
500	0.000535	**500**	0.000628
600	0.000866	**600**	0.000901
700	0.001058	**700**	0.000945
800	0.001424	**800**	0.001488
900	0.001744	**900**	0.001608
1000	0.002075	**1000**	0.001924
1100	0.002498	**1100**	0.002108
1200	0.002979	**1200**	0.002547
1300	0.003453	**1300**	0.003196
1400	0.003883	**1400**	0.003348
1500	0.004416	**1500**	0.003706
1600	0.005737	**1600**	0.004196
1700	0.005749	**1700**	0.004421
1800	0.00644	**1800**	0.004172
1900	0.007085	**1900**	0.004479
2000	0.008531	**2000**	0.005229
2100	0.008719	**2100**	0.005156
2200	0.009605	**2200**	0.004847
2300	0.011107	**2300**	0.00446
2400	0.011434	**2400**	0.004787
2500	0.012559	**2500**	0.005245
2600	0.0141	**2600**	0.009791
2700	0.015121	**2700**	0.006417
2800	0.015996	**2800**	0.006618
2900	0.01745	**2900**	0.007271
3000	0.018322	**3000**	0.007784
3100	0.020647	**3100**	0.008387

3200	0.021766	**3200**	0.009128
3300	0.022521	**3300**	0.009741
3400	0.024245	**3400**	0.010392
3500	0.026378	**3500**	0.011234
3600	0.027872	**3600**	0.011868
3700	0.029146	**3700**	0.012516
3800	0.030883	**3800**	0.013415
3900	0.032407	**3900**	0.014172

Podemos observar que para un número de entradas de **3900 el algoritmo con compilación –O2 es 2,3 veces** más rápido empíricamente que el algoritmo burbuja con compilación estándar. Veamos a continuación sus gráficas:

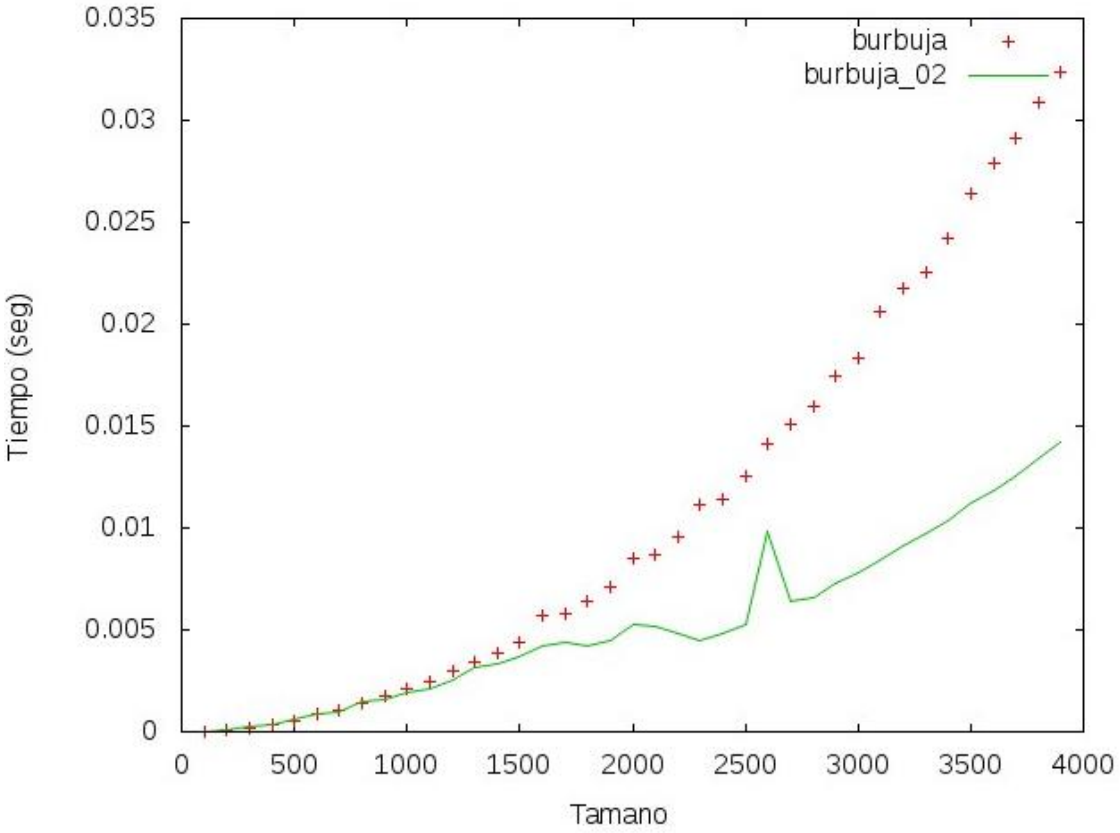


Imagen 12. Comparación de algoritmos con distintas opciones de compilación.