Nombre: Alberto Robles Hernández
DNI: 76065648W
Grupo: B2
Fecha: 09/03/20

Algorítmica 2019/20

Practica 1:

Análisis de la Eficiencia de Algoritmos

Guion:

1º-.Calculo empírico:

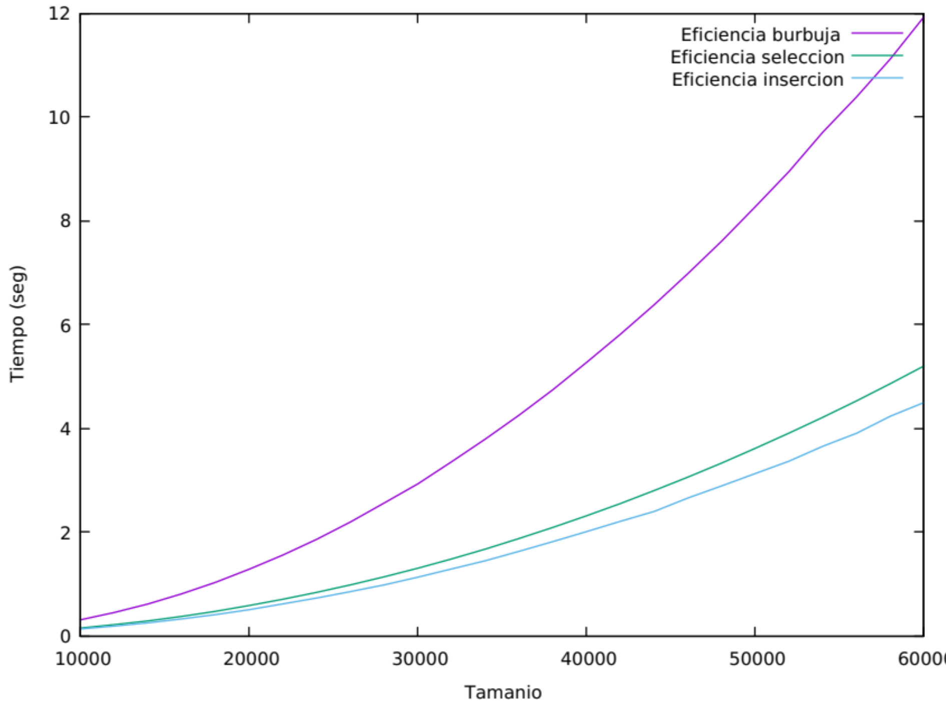
Vamos a obtener la eficiencia empírica de 6 algoritmos de ordenación, (Inserción, Selección, Burbuja, quicksort, mergesort, heapsort), de el algoritmo floyd y del algoritmo que resuelve el problema de las pirámides de hanoi.

Para empezar he separado los distintos algoritmos según su eficiencia teórica, ya que no podemos tratar de observar empíricamente la eficiencia asintótica de por ejemplo el algoritmo de quicksort y el de hanoi con las mismas entradas, ya que no seria computacionalmente viable. Por lo que al separarlos según su eficiencia teórica se nos quedan tal que:

1.1-. $O(n^2)$

Para los algoritmos de ordenación O(n²) he utilizado un tamaño de vector que comience en 10000 y que vaya incrementando su valor de 2000 en 2000 hasta llegar a 60000. Esto lo he repetido 10 veces y después hecho la media. Tras hacer esto la tabla y la grafica con los distintos tamaños y algoritmos nos quedaría:

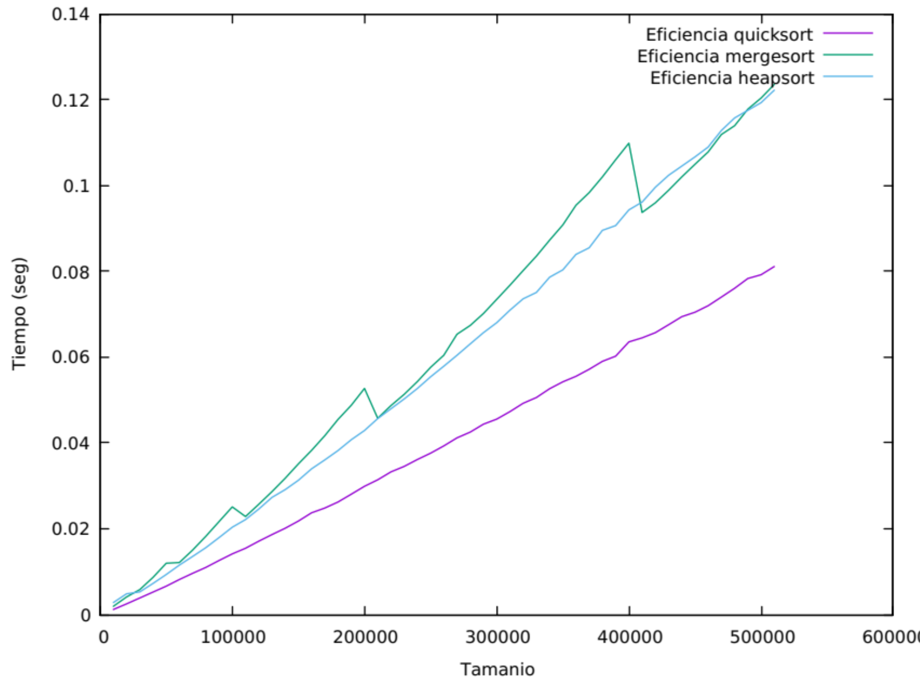
Tamaño	Burbuja	Selección	Inserción
10000	0,302993	0,144426	0,125332
12000	0,442299	0,209097	0,180031
14000	0,608327	0,28301	0,24494
16000	0,802108	0,369378	0,319463
18000	1,02372	0,467559	0,404406
20000	1,2777	0,577237	0,497208
22000	1,55023	0,698466	0,609755
24000	1,85269	0,83116	0,722747
26000	2,18826	0,975502	0,847598
28000	2,55445	1,13122	0,976089
30000	2,92562	1,29881	1,12646
32000	3,34929	1,47768	1,2828
34000	3,78682	1,66796	1,44313
36000	4,24787	1,87019	1,62453
38000	4,73862	2,08412	1,80929
40000	5,2641	2,30903	2,0042
42000	5,8097	2,54569	2,20253
44000	6,37523	2,79386	2,3912
46000	6,9766	3,05392	2,65096
48000	7,60233	3,32507	2,88274
50000	8,26628	3,6081	3,1227
52000	8,94536	3,90247	3,36166
54000	9,70451	4,20811	3,65115
56000	10,3849	4,52582	3,89966
58000	11,1164	4,85558	4,22728
60000	11,9301	5,19887	4,49366



*1.2-. O(n*log(n))*

Para los algoritmos de ordenación O(n*log(n)) he utilizado un tamaño de vector que comience en 10000 y que vaya incrementando su valor de 20000 en 20000 hasta llegar a 510000. De la misma forma que antes lo he ejecutado 10 veces y después hecho la media y ha quedado:

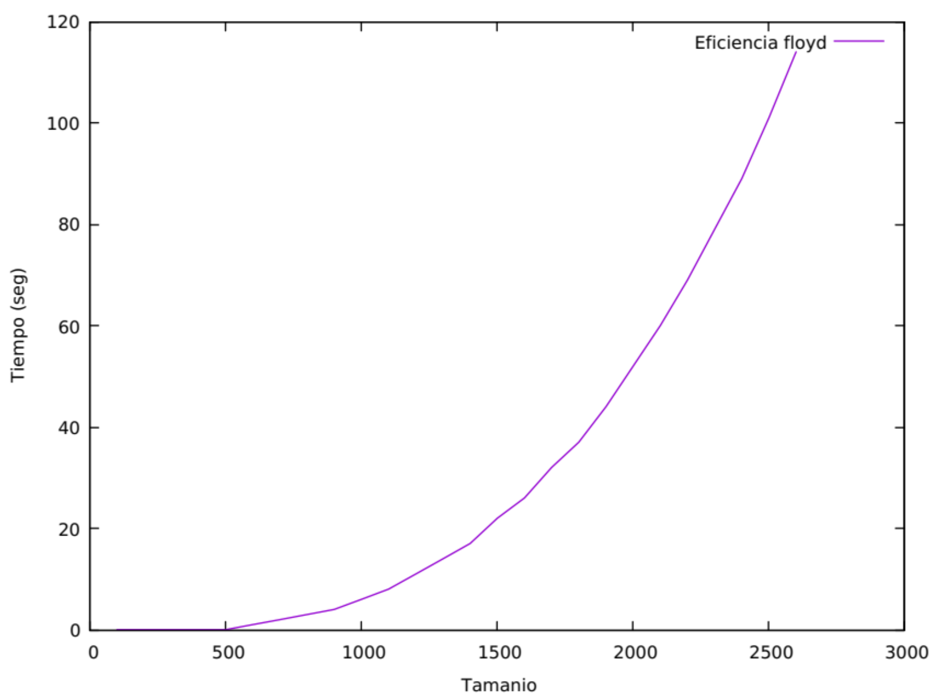
Tamaño	Heapsort	Mergesort	Quicksort
10000	0,002796	0,00189987	0,001167
30000	0,005273	0,005809	0,003846
50000	0,009323	0,011952	0,006618
70000	0,013545	0,014998	0,009605
90000	0,01794	0,021663	0,012581
110000	0,022144	0,02285	0,015449
130000	0,027288	0,02864	0,018639
150000	0,031291	0,035116	0,021805
170000	0,03603	0,041659	0,02483
190000	0,040723	0,04875	0,028028
210000	0,045683	0,045737	0,031406
230000	0,050234	0,051321	0,034526
250000	0,0554	0,057638	0,037612
270000	0,060426	0,065314	0,041156
290000	0,065701	0,070164	0,044354
310000	0,070921	0,076728	0,047273
330000	0,075056	0,083516	0,050582
350000	0,080336	0,090817	0,054247
370000	0,085456	0,098332	0,057155
390000	0,090645	0,105997	0,060227
410000	0,096117	0,093697	0,06446
430000	0,102366	0,098858	0,067531
450000	0,106668	0,104927	0,070456
470000	0,112792	0,111857	0,073978
490000	0,117503	0,117776	0,078307
510000	0,122218	0,123539	0,081083



1.3-. O(n³)

Para el algoritmo de Floyd, con una eficiencia teórica $O(n^3)$ he utilizado una entrada que comience en 100 y que vaya incrementando su valor de 100 en 100 hasta llegar a 2600. Tras ejecutarlo 10 veces y hacer la media nos queda:

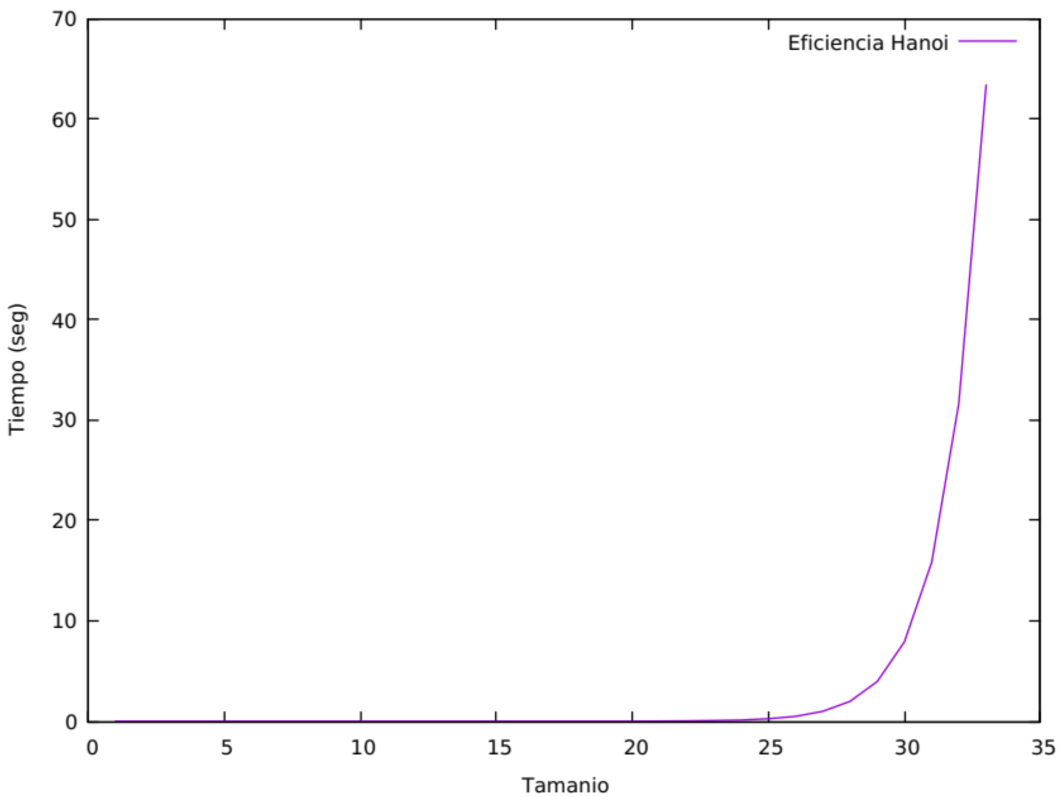
Tamaño	Floyd
100	0,009597
200	0,052907
300	0,176909
400	0,41661
500	0,818768
600	1,40149
700	2,22141
800	3,33339
900	4,76016
1000	6,55763
1100	8,74859
1200	11,3253
1300	14,3672
1400	17,9396
1500	22,0407
1600	26,7268
1700	32,037
1800	37,9883
1900	44,7065
2000	52,1548
2100	60,228
2200	69,2075
2300	79,0112
2400	89,718
2500	101,336
2600	114,011



1.4-. O(2^n)

Para el algoritmo que resuelve el problema de las pirámides de hanoi voy a utilizar una entrada muy pequeña en comparación con los otros algoritmos anteriormente observados, (debido al rápido crecimiento que tiene la función 2^n), empieza en 1 una pieza y incrementamos el numero de piezas de 1 en 1 hasta llegar a 33.

Tamaño	Hanoi
1	2,00E-06
2	2,00E-06
3	2,00E-06
4	3,00E-06
5	3,00E-06
6	3,00E-06
7	4,00E-06
8	5,00E-06
9	6,00E-06
10	9,00E-06
11	2,00E-05
12	3,20E-05
13	6,30E-05
14	0,000123
15	0,000244
16	0,000485
17	0,000971
18	0,001933
19	0,003902
20	0,007816
21	0,015508
22	0,030994
23	0,062063
24	0,124215
25	0,247866
26	0,494867
27	0,990299
28	1,97943
29	3,95856
30	7,91544
31	15,8299
32	31,6607
33	63,3761

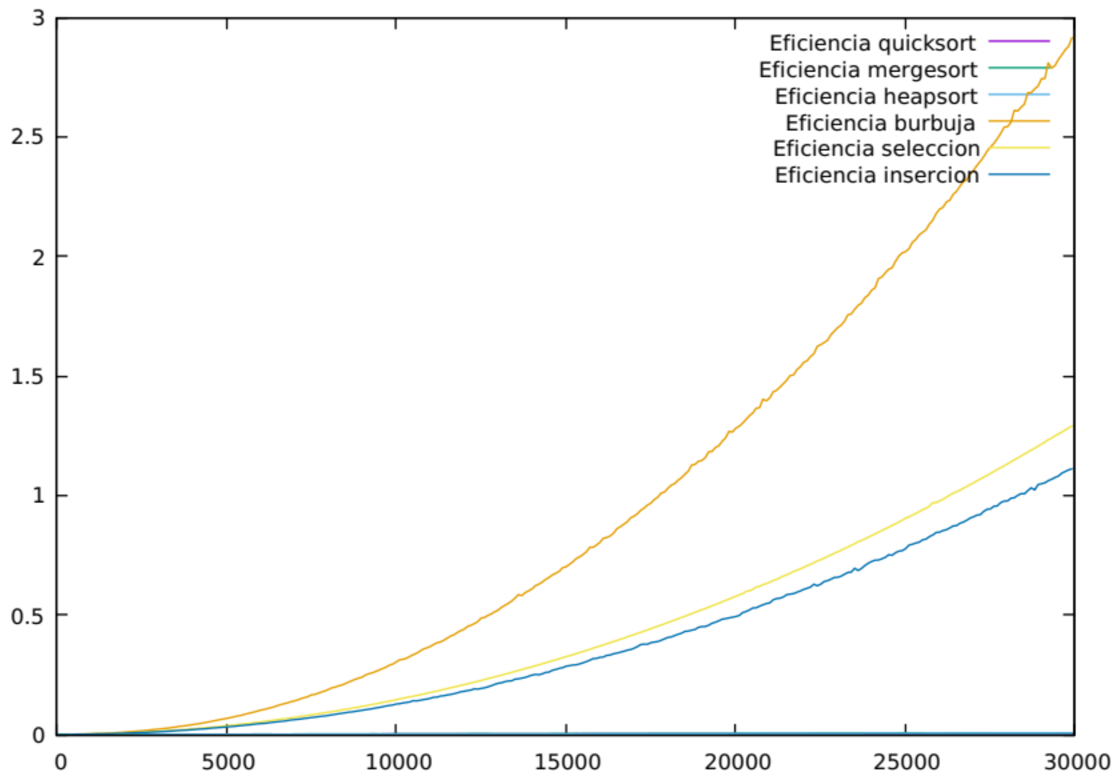


<u>2º-.Comparación de algoritmos de ordenación:</u>

Como podemos observar hay una clara diferencia entre dos conjuntos de algoritmos de ordenación, los algoritmos $O(n^2)${Burbuja, Inserción, Selección} y los algoritmos O(n*log(n)) {Quicksort, Heapsort, Mergesort}.
Esto implica que al tener vectores muy grandes, el tiempo que tardan en ordenar dicho vector los algoritmos $O(n^2)$ va a ser mucho mayor que lo que tarden los O(n*log(n)); Para ponernos en perspectiva de la diferencia entre ambos conjuntos he obtenido el tiempo de ejecución de ordenar un vector de 200000 casillas.

	Burbuja	Selección	Inserción	Quicksort	Heapsort	Mergesort
Segundos	133,451	57.7356	49.9891	0.029696	0.043619	0.052706

Como podemos observar la diferencia de tiempos es considerable cuando el tamaño del vector es grande, o este tiende al infinito. Y tal como muestra la grafica continuación esta diferencia de tiempos se incrementa conforme crece el tamaño del vector.



3º-.Eficiencia híbrida:

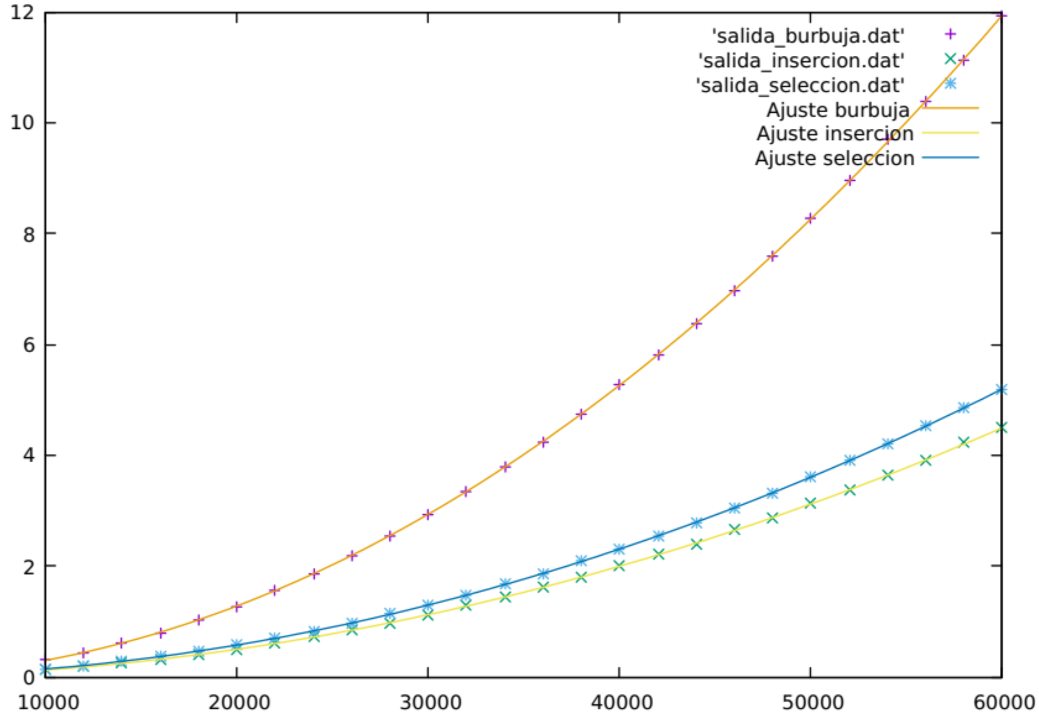
Eficiencia híbrida de la burbuja	Final set of parameters =========================== a0 = 3.36521e-09 a1 = -2.75025e-06 a2 = -0.0130417 r = 0.9818 f(x)=3.36521e-09*x²-2.75025e-06*x-0.0130417
Eficiencia híbrida de la inserción	Final set of parameters =========================== a0 = 1.24984e-09 a1 = -7.23036e-08 a2 = 0.00185414 r = 0.9821 f(x)=1.24984e-09*x²-7.23036e-08*x-0.00185414

Eficiencia híbrida de la selección	Final set of parameters =========================== a0 = 1.44532e-09 a1 = -1.36815e-07 a2 = 0.00186103 r = 0.9822 f(x)=1.44532e-09*x^2-1.36815e-07*x-0.00186103
Eficiencia híbrida del quicksort	Final set of parameters =========================== a0 = 1.21093e-08 r = 0.9997 f(x)=1.21093e-08 *x*log(x)
Eficiencia híbrida del heapsort	Final set of parameters =========================== a0 = 1.8138e-08 r = 0.9991 f(x)=1.8138e-08 *x*log(x)
Eficiencia híbrida del mergesort	Final set of parameters =========================== a0 = 1.89671e-08 r = 0.9931 f(x)=1.89671e-08*x*log(x)
Eficiencia híbrida de Floyd	Final set of parameters =========================== a0 = 6.35546e-09 a1 = 4.36785e-07 a2 = -0.000280022 a3 = 0.0399928 r = 0.9214 f(x)=6.35546e-09*x^3+4.36785e-07*x^2-0.000280022*x+0.0399928
Eficiencia híbrida de Hanoi	Final set of parameters =========================== a0 = 7.37636e-09 r = 0.9107

	f(x)=7.37636e-09*2^x

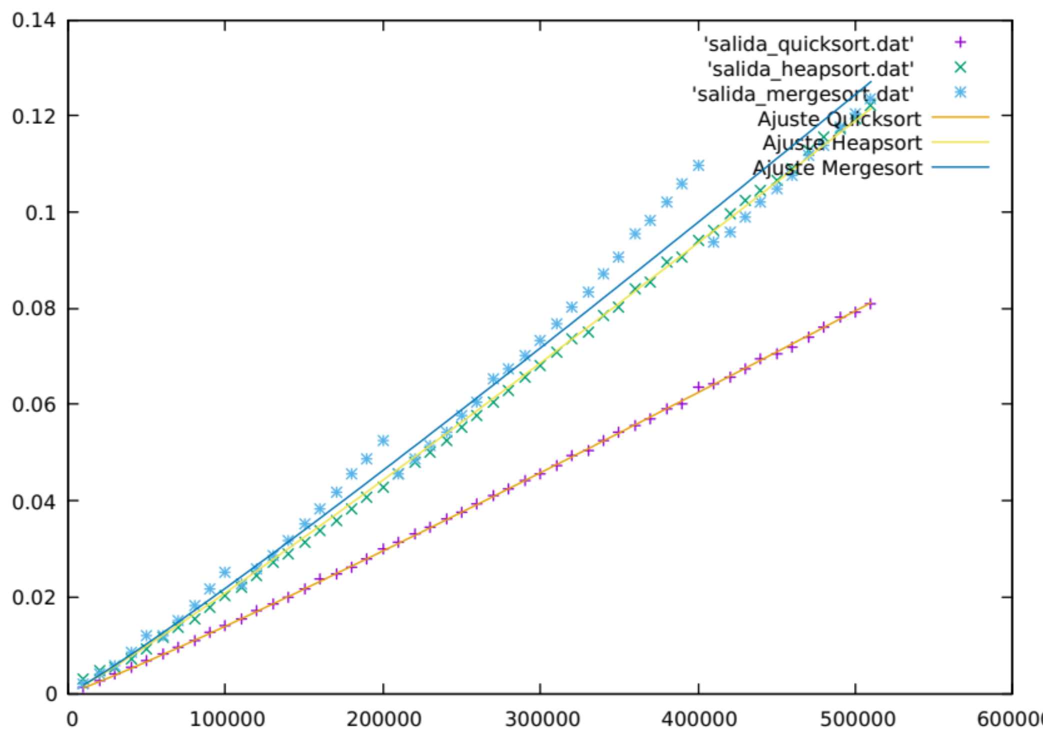
3.1.1-.Eficiencia híbrida de algoritmos de eficiencia teórica O(n²)

Ajuste: x²*a0+x*a1+a2



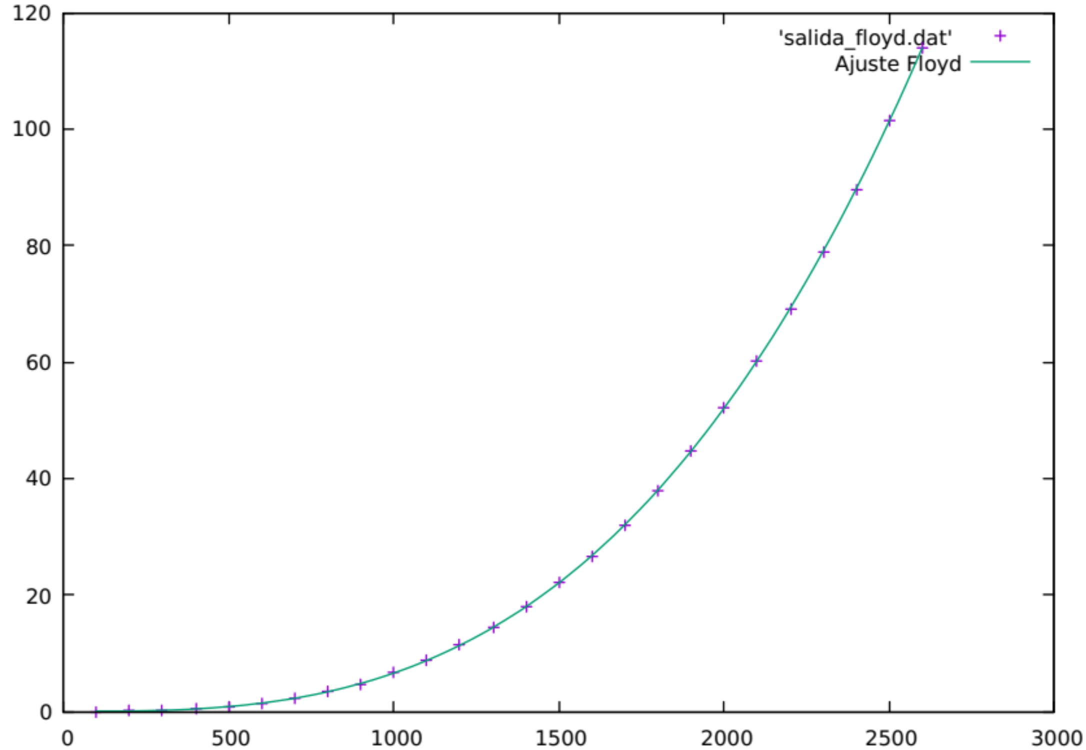
*3.1.2-.Eficiencia híbrida de algoritmos de eficiencia teórica O(n*log(n))*

Ajuste: a0 * x*log(x)



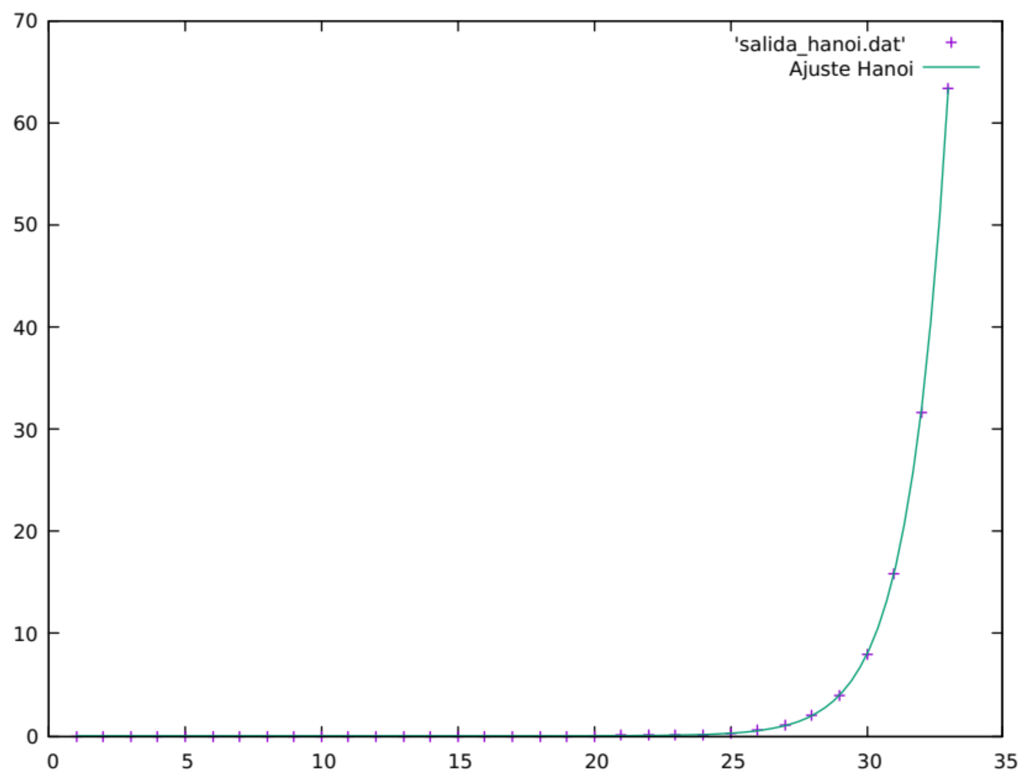
3.1.3-.Eficiencia híbrida de eficiencia teórica O(n³)→Floyd

Ajuste: $x^3a0 + x^2a1 + xa2 + a3$



3.1.4-.Eficiencia híbrida de algoritmo de eficiencia teórica O(2^n)→Hanoi

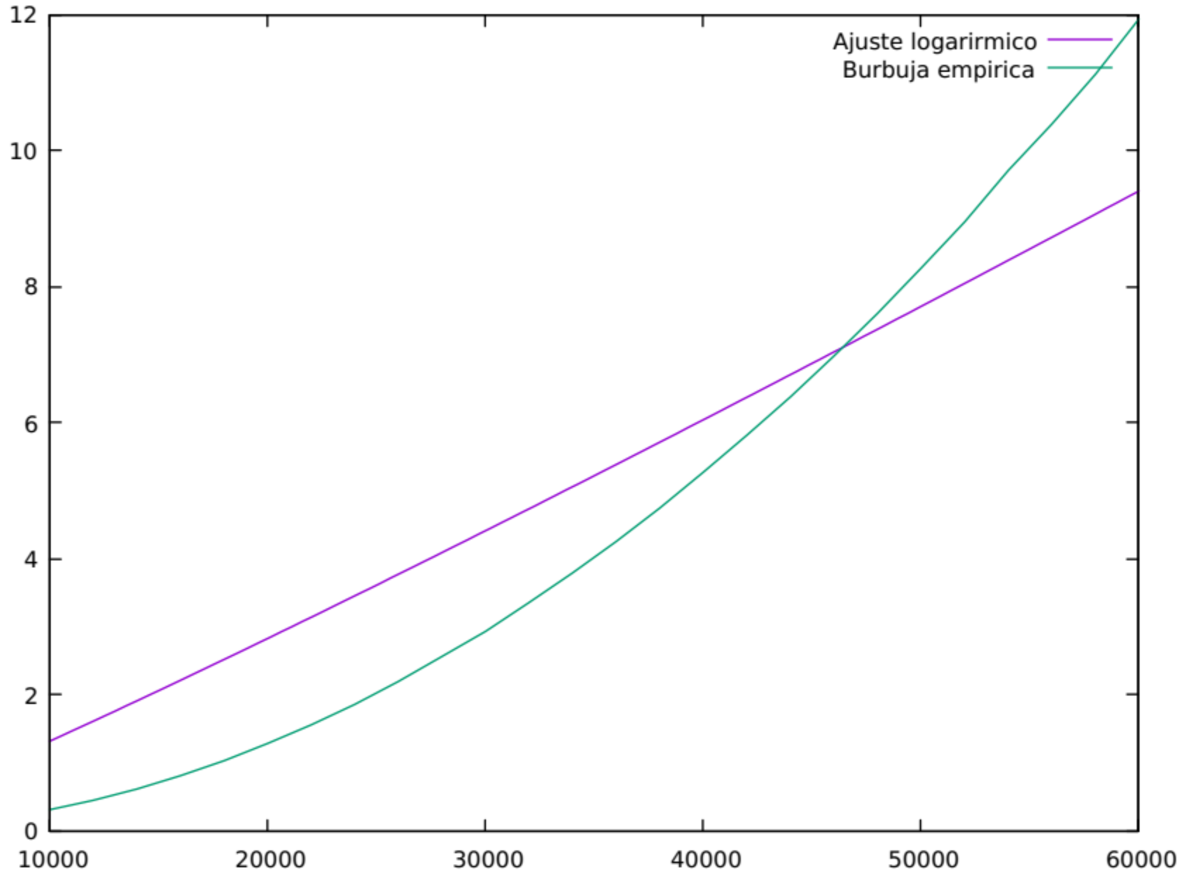
Ajuste: a0 * 2^x



*3.2-.Algoritmo de eficiencia teórica $O(n^2)$ ajustado a a0*n*log(n)*

Al tratar de calcular la eficiencia híbrida del algoritmo de la burbuja ajustando los datos obtenidos de sus ejecuciones a a0*n*log(n) el resultado es que la eficiencia híbrida sería:
f(x) = 1.42424e-05*x*log(x)

La cual al pintarla en una grafica junto con la grafica de la eficiencia empírica de la burbuja nos quedaría:



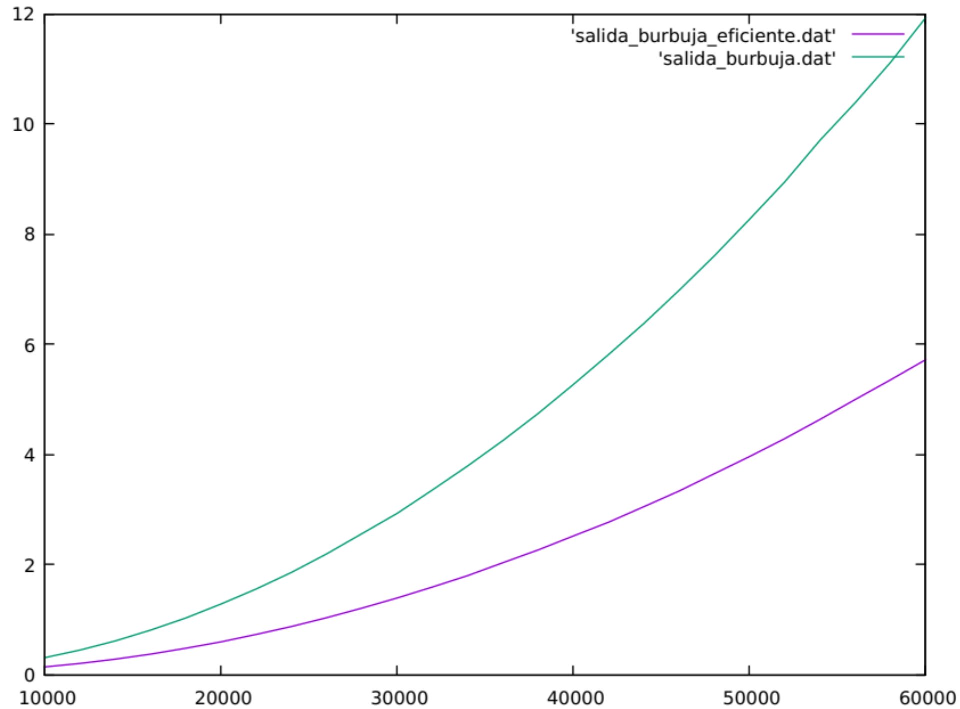
Como podemos observar no se acerca nada a la eficiencia real de dicho algoritmo por lo que seria un ajusto muy malo, no nos sirve.

4º-.Otros aspectos que influyen en la eficiencia:

Ahora vamos a observar como afecta la optimización con la que compilemos distintos programas a la eficiencia empírica de los mismos.
Voy a coger un algoritmo de cada conjunto de eficiencia teórica y a comprarlo con sigo mismo, uno compilado sin optimización y otro con optimización -O3.
Y como podemos observar la diferencia entre usar -O3 y no usarlo es unicamente un factor multiplicativo, lo que quiere decir que por mucho que mejoremos la maquina en la que ejecutamos los algoritmos o la optimización con la que compilamos no vamos a conseguir por ejemplo que cuando $n \rightarrow$ infinito un algoritmo de ordenación $O(n^2)$ sea mas rápido que uno $O(n\log(n))$.

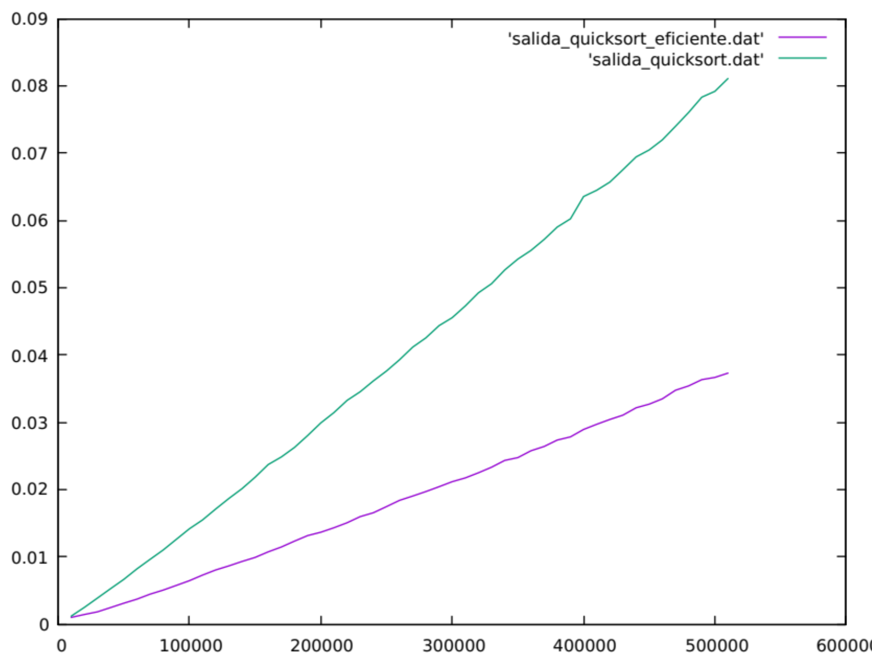
Burbuja vs Burbuja -O3

Tamaño	Burbuja	Burbuja Eficiente
10000	0,302993	0,135131
12000	0,442299	0,19732
14000	0,608327	0,275031
16000	0,802108	0,365846
18000	1,02372	0,473449
20000	1,2777	0,591317
22000	1,55023	0,724628
24000	1,85269	0,869212
26000	2,18826	1,02873
28000	2,55445	1,20296
30000	2,92562	1,38779
32000	3,34929	1,58558
34000	3,78682	1,79312
36000	4,24787	2,02683
38000	4,73862	2,25879
40000	5,2641	2,51276
42000	5,8097	2,76566
44000	6,37523	3,04755
46000	6,9766	3,33025
48000	7,60233	3,64407
50000	8,26628	3,95542
52000	8,94536	4,2817
54000	9,70451	4,63166
56000	10,3849	4,99395
58000	11,1164	5,35136
60000	11,9301	5,71766



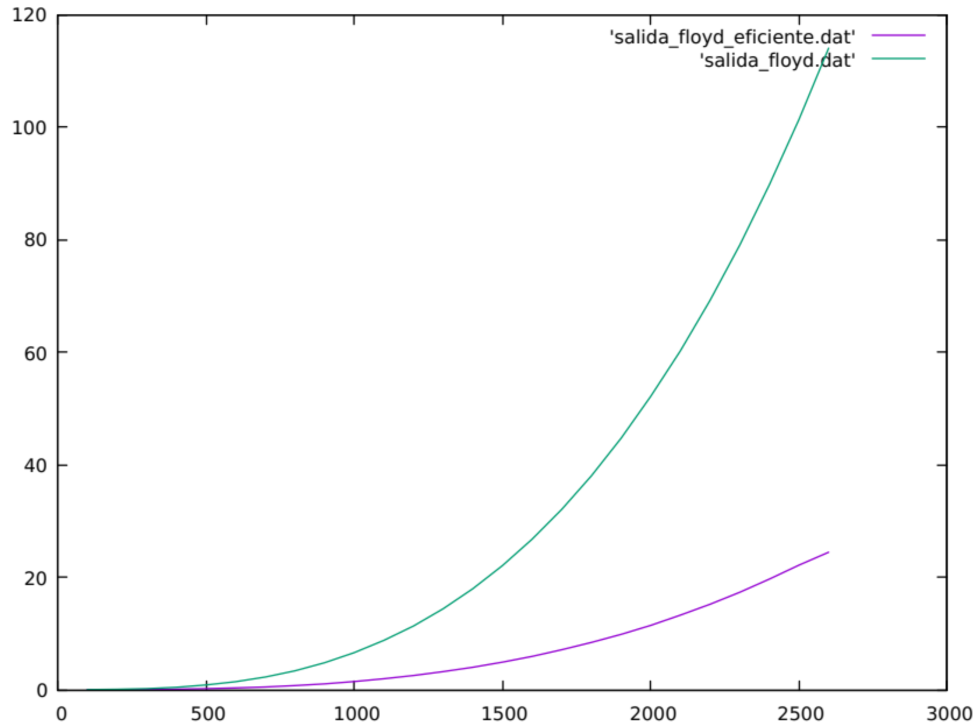
Quicksort vs Quicksort -O3

Tamaño	Quicksort	Quicksort Eficiente
10000	0,001167	0,000971
30000	0,003846	0,0018
50000	0,006618	0,003076
70000	0,009605	0,004436
90000	0,012581	0,005732
110000	0,015449	0,007266
130000	0,018639	0,008634
150000	0,021805	0,009902
170000	0,02483	0,011445
190000	0,028028	0,013142
210000	0,031406	0,01429
230000	0,034526	0,015948
250000	0,037612	0,017453
270000	0,041156	0,019016
290000	0,044354	0,020407
310000	0,047273	0,021705
330000	0,050582	0,023298
350000	0,054247	0,024723
370000	0,057155	0,026392
390000	0,060227	0,027815
410000	0,06446	0,029673
430000	0,067531	0,031043
450000	0,070456	0,03268
470000	0,073978	0,034743
490000	0,078307	0,036314
510000	0,081083	0,037299



Floyd vs Floyd -O3

Tam	Floyd	Floyd Eficiente
100	0,009597	0,002473
200	0,052907	0,010689
300	0,176909	0,036223
400	0,41661	0,085853
500	0,818768	0,167954
600	1,40149	0,290707
700	2,22141	0,463672
800	3,33339	0,7051
900	4,76016	1,02257
1000	6,55763	1,43656
1100	8,74859	1,93599
1200	11,3253	2,52129
1300	14,3672	3,19996
1400	17,9396	3,97942
1500	22,0407	4,88166
1600	26,7268	5,91793
1700	32,037	7,083
1800	37,9883	8,3783
1900	44,7065	9,81367
2000	52,1548	11,4304
2100	60,228	13,2394
2200	69,2075	15,1625
2300	79,0112	17,2919
2400	89,718	19,6509
2500	101,336	22,1313
2600	114,011	24,4128



Hanoi vs Hanoi -O3

Tamaño	Hanoi	Hanoi eficiente
1	3,00E-06	3,00E-06
2	3,00E-06	3,00E-06
3	2,00E-06	2,00E-06
4	2,00E-06	1,00E-06
5	2,00E-06	2,00E-06
6	3,00E-06	2,00E-06
7	3,00E-06	3,00E-06
8	3,00E-06	3,00E-06
9	6,00E-06	4,00E-06
10	9,00E-06	5,00E-06
11	2,00E-05	9,00E-06
12	3,20E-05	1,50E-05
13	6,30E-05	2,80E-05
14	0,000123	5,50E-05
15	0,000244	0,000108
16	0,000485	0,000219
17	0,000971	0,000428
18	0,001933	0,000872
19	0,003902	0,001744
20	0,007816	0,003393
21	0,015508	0,006801
22	0,030994	0,013646
23	0,062063	0,026938
24	0,124215	0,054283
25	0,247866	0,107692
26	0,494867	0,216029
27	0,990299	0,429027
28	1,97943	0,85898
29	3,95856	1,72521
30	7,91544	3,45827
31	15,8299	6,91705
32	31,6607	13,808
33	63,3761	27,6278