Algorítmica: Práctica 1

Andrés Herrera Poyatos, Antonio Rafael Moya Martín-Castaño, Iván Sevillano García, Juan Luis Suárez Díaz

22 de marzo de 2015

Contents

Organización de la práctica

Se adjunta el directorio comprimido **Code** con todos los datos obtenidos. La información se organiza como sigue:

- Los códigos **.cpp** de los distintos algoritmos están disponibles en la carpeta **src**.
- En la carpeta **sh** se encuentran scripts auxiliares, cada uno especializado en la toma de datos de uno o varios algoritmos concretos.
- En la carpeta **plot**, de la misma forma, se encuentran scripts especializados en la elaboración de las distintas gráficas.
- El script de bash **ejecuciones.sh** se encarga de obtener todos los datos y gráficas para todos los algoritmos llamando a los scripts mencionados anteriormente.
- En las carpetas **Datos*Autor*** se almacenan los archivos .dat generados por cada uno de los autores, en sus respectivos PCs. Los ficheros contienen, para cada algoritmo, varias parejas *[tamaño tiempo]* correspondientes a distintas ejecuciones del programa con distintos tamaños y sus respectivos resultados. Han sido generados con **ejecuciones.sh** y son utilizados a lo largo del trabajo.
- De forma análoga, están disponibles los directorios **Tablas*Autor*** e **Imagenes*Autor*** que contienen tablas en formato **.md** con los resultados y las gráficas del comportamiento de los algoritmos generadas por *gnuplot*, respectivamente.

Cada ejercicio tiene su apartado en el pdf con su corresponiente enunciado y solución.

Ejercicio 1: Cálculo de la eficiencia empírica

Enunciado:

Calcule la eficiencia empírica de los algoritmos pedidos. Defina adecuadamente los tamaños de entrada para que se generen al menos 25 datos. Incluya en la memoria tablas diferentes para los algoritmos de distinto orden de eficiencia (una con los algoritmos $O(n^2)$, otra con los $O(n \log n)$, otra con $O(n^3)$ y otra con $O((\frac{1+\sqrt{5}}{2})^n)$.

A continuación se proporcionan las tablas, una para cada clase de algoritmos:

Tabla con los algoritmos cuadráticos

Tamaño del Vector	Burbuja	Seleccion	Insercion
1000	0.005971	0.003397	0.001321
2000	0.018136	0.009589	0.007588
3000	0.024143	0.014704	0.020282
4000	0.043267	0.025817	0.023064
5000	0.067684	0.037817	0.034221
6000	0.099499	0.055028	0.047872
7000	0.137072	0.073739	0.064517
8000	0.181558	0.092111	0.082905

Tamaño del Vector	Burbuja	Seleccion	Insercion
9000	0.232648	0.118624	0.103043
10000	0.290489	0.14394	0.124546
11000	0.354349	0.178614	0.151216
12000	0.433737	0.20678	0.178228
13000	0.519202	0.239558	0.209278
14000	0.59308	0.273397	0.248141
15000	0.689312	0.314147	0.276967
16000	0.789129	0.356495	0.317291
17000	0.890449	0.402106	0.358508
18000	1.01538	0.450575	0.397242
19000	1.1313	0.50472	0.435913
20000	1.26128	0.55525	0.483853
21000	1.39441	0.611367	0.541654
22000	1.55788	0.670662	0.600085
23000	1.68169	0.732809	0.644882
24000	1.84769	0.800821	0.70273
25000	1.9893	0.864984	0.762199

Tabla con los algoritmos cúbicos

Nodos del Grafo	Floyd
32	0.000596
64	0.004593
96	0.01017
128	0.017141
160	0.035407
192	0.054113
224	0.083649
256	0.116013
288	0.153556
320	0.217792
352	0.280357
384	0.362685
416	0.460287
448	0.581175
480	0.703839

Nodos del Grafo	Floyd
512	0.852424
544	1.02124
576	1.25977
608	1.44669
640	1.68365
672	1.93344
704	2.23303
736	2.54158
768	2.89293
800	3.25971

Tabla con el algoritmo de Fibonacci ($O((\frac{1+\sqrt{5}}{2})^n)$)

Índice	Fibonacci
15	1.3e-05
16	2e-05
17	2.6e-05
18	4.4e-05
19	5e-05
20	0.000114
21	8.6e-05
22	0.000154
23	0.000582
24	0.00097
25	0.001314
26	0.002554
27	0.002394
28	0.003356
29	0.004289
30	0.007083
31	0.011583
32	0.017354
33	0.029313
34	0.047371
35	0.073093
36	0.127835

Índice	Fibonacci
37	0.190808
38	0.308124
39	0.498824
40	0.849934

Tabla con el algoritmo de Hanoi ($O(2^n)$)

Num. Discos	Hanoi
5	1e-06
6	3e-06
7	3e-06
8	6e-06
9	9e-06
10	1.3e-05
11	4.9e-05
12	7.6e-05
13	0.00015
14	0.00019
15	0.000393
16	0.000851
17	0.002302
18	0.003382
19	0.009191
20	0.019015
21	0.024593
22	0.041194
23	0.065421
24	0.127555
25	0.246427
26	0.483075
27	0.96832
28	1.9249
29	3.83247
30	7.63996

Tabla con los algoritmos n log n

Tamaño del Vector	Mergesort	Quicksort	Heapsort
40000	0.015087	0.006235	0.008042
80000	0.02682	0.014736	0.018251
120000	0.037756	0.02246	0.027588
160000	0.041266	0.025439	0.043
200000	0.059359	0.032775	0.048956
240000	0.057706	0.041055	0.067071
280000	0.065938	0.045861	0.065239
320000	0.082393	0.053183	0.072391
360000	0.093771	0.057395	0.095929
400000	0.107337	0.063843	0.102829
440000	0.102685	0.071064	0.104917
480000	0.122825	0.076521	0.111892
520000	0.136037	0.082585	0.120963
560000	0.141045	0.087434	0.132638
600000	0.150005	0.093448	0.143338
640000	0.1658	0.100634	0.156171
680000	0.181068	0.109131	0.165101
720000	0.211107	0.115456	0.178005
760000	0.205422	0.121493	0.188856
800000	0.226734	0.129283	0.193322
840000	0.200972	0.136155	0.207626
880000	0.211482	0.141553	0.223124
920000	0.23571	0.148845	0.229493
960000	0.240497	0.155352	0.247564
1000000	0.244299	0.178312	0.267759

Tabla con los algoritmos de ordenación

Finalmente, mostramos una tabla con la comparativa de todos los algoritmos de ordenación, tanto cuadráticos como $n \log n$. Podemos apreciar que para tamaños relativamente pequeños (25.000) ya existen notables diferencias:

Tamaño del Vector	Burbuja	Seleccion	Insercion	Mergesort	Quicksort	Heapsort
1000	0.005971	0.003397	0.001321	0.000359	0.000195	0.000114
2000	0.018136	0.009589	0.007588	0.000756	0.000269	0.000639
3000	0.024143	0.014704	0.020282	0.00074	0.000671	0.001008

Tamaño del Vector	Burbuja	Seleccion	Insercion	Mergesort	Quicksort	Heapsort
4000	0.043267	0.025817	0.023064	0.000748	0.00067	0.001327
5000	0.067684	0.037817	0.034221	0.002255	0.000567	0.000822
6000	0.099499	0.055028	0.047872	0.001549	0.001661	0.001141
7000	0.137072	0.073739	0.064517	0.003041	0.001953	0.001359
8000	0.181558	0.092111	0.082905	0.003166	0.002456	0.001679
9000	0.232648	0.118624	0.103043	0.004058	0.001649	0.002775
10000	0.290489	0.14394	0.124546	0.003803	0.001971	0.002436
11000	0.354349	0.178614	0.151216	0.004144	0.002048	0.003717
12000	0.433737	0.20678	0.178228	0.0041	0.003616	0.004062
13000	0.519202	0.239558	0.209278	0.005279	0.003037	0.004538
14000	0.59308	0.273397	0.248141	0.006677	0.002399	0.003738
15000	0.689312	0.314147	0.276967	0.006024	0.002247	0.006105
16000	0.789129	0.356495	0.317291	0.007461	0.002721	0.003677
17000	0.890449	0.402106	0.358508	0.006324	0.002745	0.00402
18000	1.01538	0.450575	0.397242	0.008806	0.005765	0.00688
19000	1.1313	0.50472	0.435913	0.008887	0.004034	0.003986
20000	1.26128	0.55525	0.483853	0.007868	0.003407	0.007215
21000	1.39441	0.611367	0.541654	0.00544	0.00359	0.007316
22000	1.55788	0.670662	0.600085	0.006238	0.00374	0.008641
23000	1.68169	0.732809	0.644882	0.010339	0.006076	0.008602
24000	1.84769	0.800821	0.70273	0.010512	0.006921	0.009229
25000	1.9893	0.864984	0.762199	0.009398	0.003104	0.006293

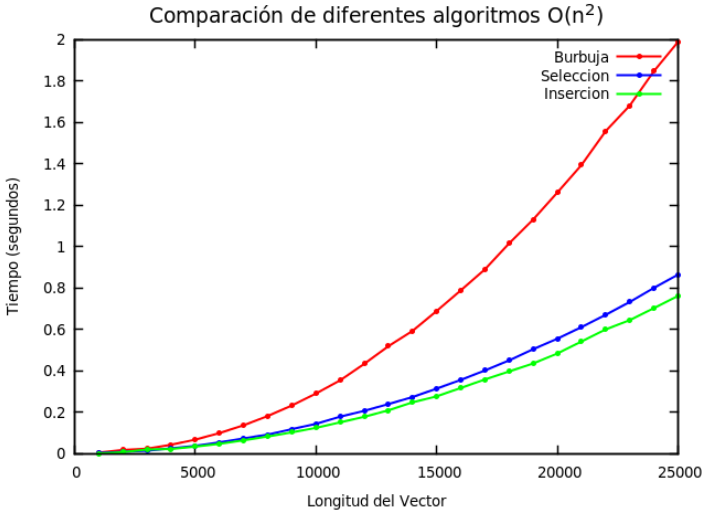
Ejercicio 2: Elaboración de gráficas

Enunciado:

Con cada una de las tablas anteriores genere un gráfico comparando los tiempos de los algoritmos. Indique claramente el significado de cada serie. Para los algoritmos que realizan la misma tarea (los de ordenación), incluya también una gráfica con todos ellos, para poder apreciar las diferencias de rendimiento de algoritmos con diferente orden de eficiencia.

No se debe pasar por alto que los datos utilizados en los algoritmos de ordenación y el de Floyd se generan de manera aleatoria. Esto influye sobre todo al comparar los algoritmos de ordenación entre sí pues es bien conocido que ciertos algoritmos operan bien cuando los datos están casi ordenados (insección, burbuja) y otros no (quicksort). Existen otros algoritmos que el tiempo dedicado es casi independiente de los datos de entrada (selección, mergesort, heapsort). Sin embargo, el error que se pueda cometer por este hecho es pequeño ya que es muy raro obtener datos casi ordenados de forma aleatoria.

Gráfica comparativa de los algoritmos cuadráticos.



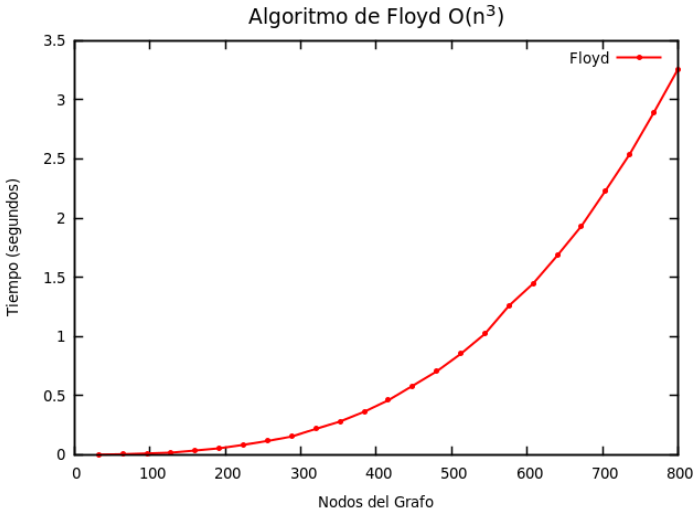
Cabe destacar, que dentro de los algoritmos cuadráticos también existen ciertas diferencias; a continuación, vamos a razonar su existencia. Como se puede observar, el algoritmo de la burbuja emplea un mayor tiempo que los otros dos. Le sigue el algoritmo de selección y de cerca el de inserción. La razón es la siguiente:

- Comparaciones e intercambios del algoritmo de selección: $\frac{n(n-1)}{2}$ y n respectivamente.
- Comparaciones e intercambios del algoritmo de burbuja: $\frac{n(n-1)}{2}$ y un intercambio por cada par de índices $i < j$ tales que $v[i] > v[j]$.
- Comparaciones e intercambios del algoritmo de inserción: $\frac{n(n-1)}{4}$ en promedio y n desplazamientos de los datos (uno por iteración) luego $\frac{n(n-1)}{4}$ asignaciones en promedio.

El número de intercambios del algoritmo burbuja suele ser mucho mayor en promedio que en el caso del algoritmo de selección, que siempre es n. Los intercambios son una operación relativamente costosa, lo que produce el mayor tiempo dedicado.

Veamos ahora, por qué el de insercción es el mejor de los tres. El algoritmo de insercción, en su peor caso (vector totalmente invertido) utiliza el mismo número de comparaciones que el de selección y en el caso promedio la mitad. Esto produce un ahorro importante de tiempo que prevalece en el total del algoritmo (los desplazamientos no son tan costosos). Al ser mejor que el de selección ya es mejor automáticamente que el algoritmo de burbuja.

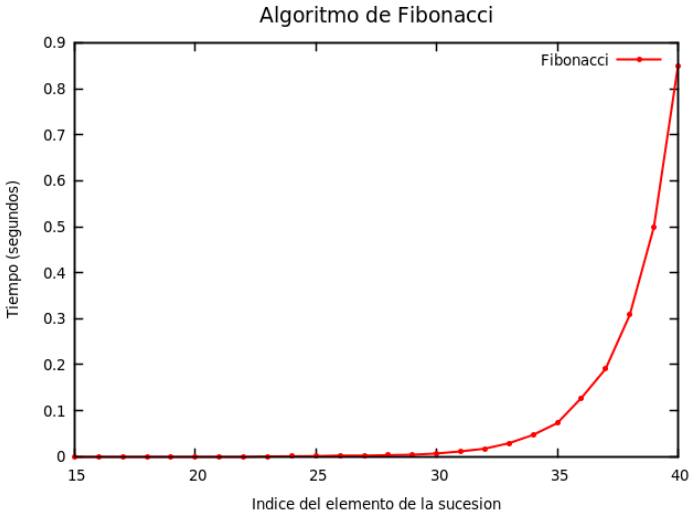
Gráfica del algoritmo cúbico (Floyd)



Contrastando con las gráficas anteriores, podemos ver que existe una gran diferencia entre los algoritmos cuadráticos y el algoritmo de Floyd, que es cúbico (éste último emplea un tiempo mucho mayor).

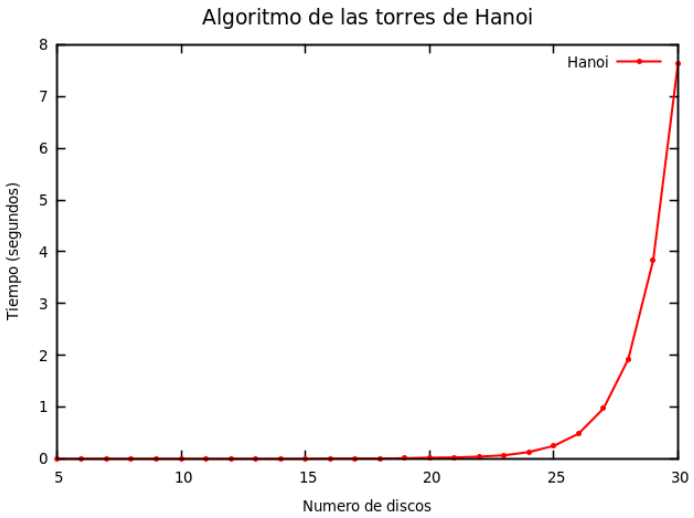
Como nota, el algoritmo de Dijkistra resuelve el mismo problema para grafos con pesos no negativos y consigue una eficiencia de $O(n^2 \log n)$, por lo que es mucho más práctico dada la diferencia que existe entre ambos órdenes de eficiencia.

Gráfica del algoritmo de Fibonacci ($O((\frac{1+\sqrt{5}}{2})^n)$)



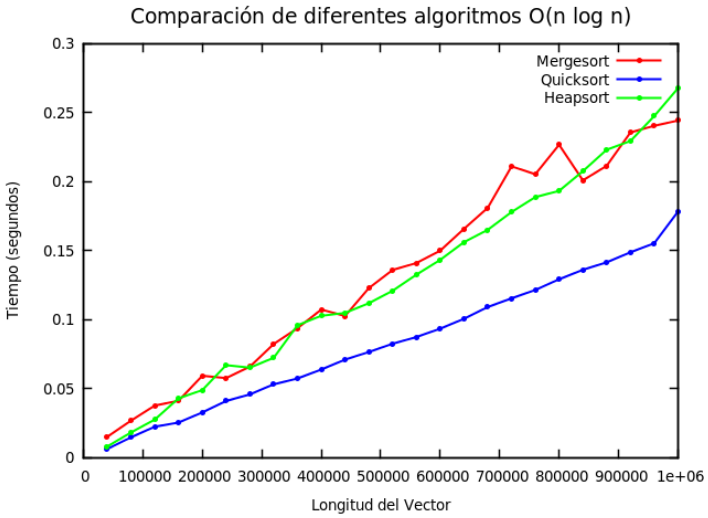
Tanto en éste algoritmo como en el de Hanoi podemos ver la naturaleza exponencial de la curva. ¡A partir de un n pequeño el tiempo ya es de segundos! Por ello se han utilizado datos pequeños en las ejecuciones. Como curiosidad, para $n = 50$ ya no terminaba en un tiempo razonable.

Gráfica del algoritmo de Hanoi ($O(2^n)$)



La naturaleza exponencial de este algoritmo es más drástica, tardando más de 7 segundos solo para $n = 30$. ¡Para $n = 40$ tardaría dos días!

Gráfica de los algoritmos $O(n \log n)$



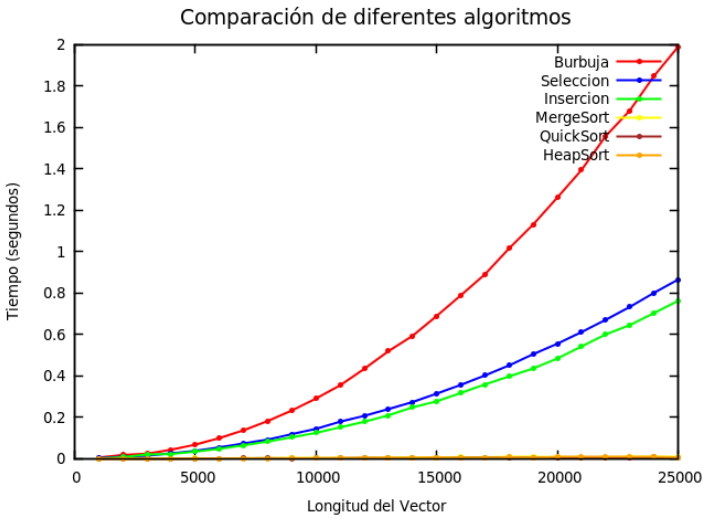
El algoritmo quicksort es claramente más rápido. Este algoritmo es de naturaleza aleatoria, así que el hecho de que los vectores se generen aleatoriamente influye en su correcto funcionamiento. Nótese que si los vectores estuviesen casi ordenados quicksort sería cuadrático. Sin embargo, esto es muy difícil que pase con vectores aleatorios.

El peor funcionamiento de mergesort se produce ya que utiliza espacio extra en la operación de *merge*, que suele ser además bastante más lenta que el particionamiento del quicksort.

En el caso del heapsort, este algoritmo en primer lugar crea el heap y luego lo ordena extrayendo el mínimo cada vez. El tener que realizar ambas operaciones hace que su constante sea casi el doble que la de quicksort que empieza a ordenar desde el primer momento.

Mergesort y heapsort operan con similar velocidad. Sin embargo, es preferible el último puesto que no utiliza memoria exta, es *in-place*.

Gráfica comparativa con todos los algoritmos de ordenación.



Por último y a modo de conclusión, podemos observar esta gráfica en la que quedan comparados los algoritmos cuadráticos y los de orden $n\log(n)$. No se compara con el resto de algoritmo por la abismal diferencia existente, hecho mostrado anteriormente. Esta diferencia es aún mayor en el caso de los exponenciales que son inviables en la práctica.

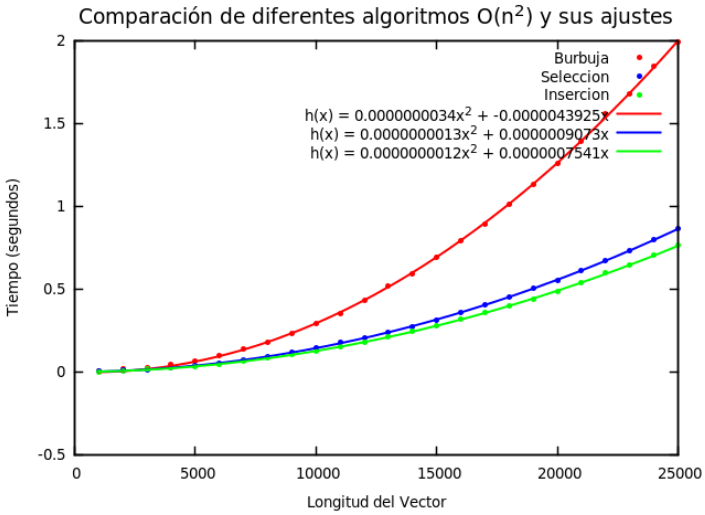
En cuánto a lo que vemos en esta gráfica, podemos observar la gran diferencia entre los algoritmos quicksor, mergesort y heapsort y los cuadráticos. Podemos ver también que dentro de este conjunto de algoritmos, como hemos explicado anteriormente, el peor de todos es el de la burbuja.

Ejercicio 3: Eficiencia híbrida.

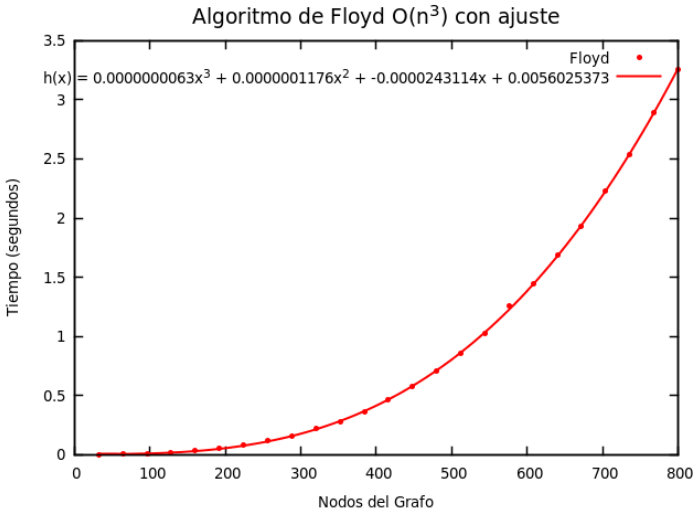
Enunciado:

Calcule la eficiencia híbrida de los algoritmos. Pruebe también con otros ajustes que no se correspondan con la eficiencia teórica y compruebe la variación en la calidad del ajuste.

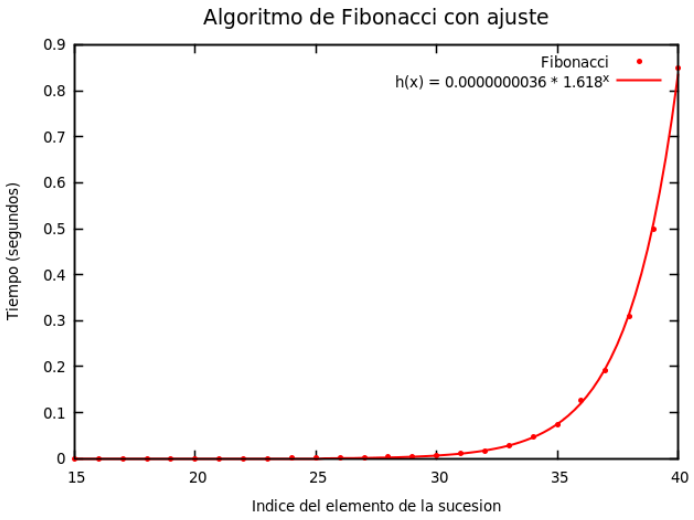
Ajustes de los algoritmos de ordenación cuadráticos:



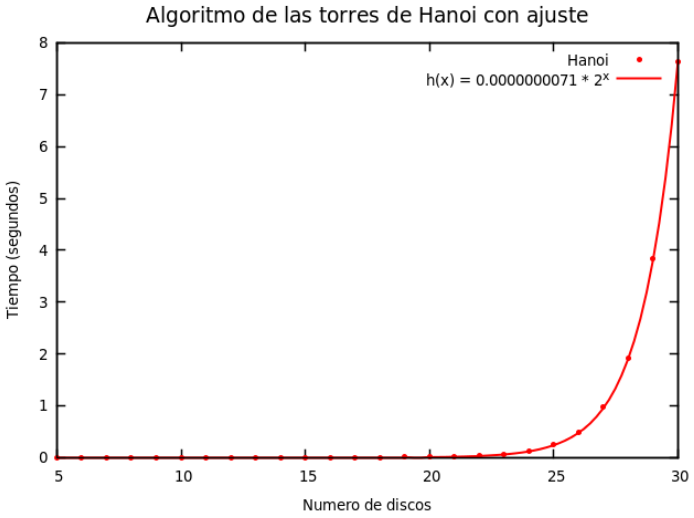
Ajuste del algoritmo de Floyd:



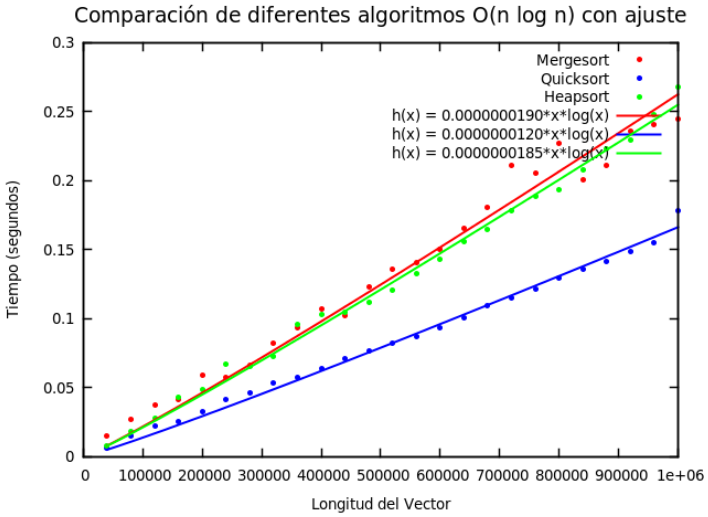
Ajuste del algoritmo de Fibonacci:



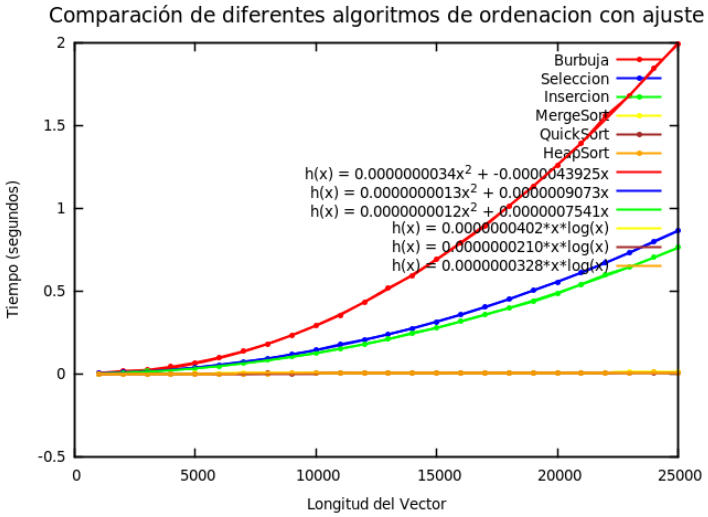
Ajuste del algoritmo de las torres de Hanoi:



Ajuste de los algoritmos de ordenación $O(n \log n)$



Comparativa de ajustes de todos los algoritmos de ordenación



Ejercicio 4: estudio de la eficiencia empírica en función de parámetros externos.

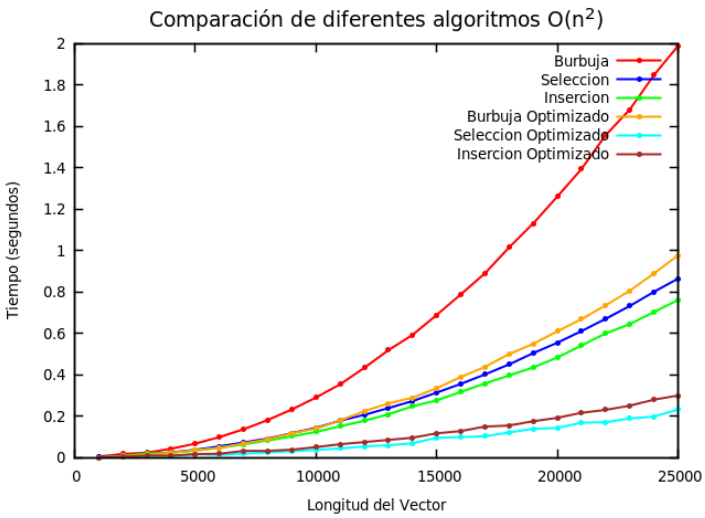
Enunciado:

Otro aspecto interesante a analizar mediante este tipo de estudio es la variación de la eficiencia empírica en función de parámetros externos tales como: las opciones de compilación utilizadas, el ordenador donde se realizan las pruebas, el sistema operativo, etc. Sugiera algún estudio de este tipo, consulte con el profesor de prácticas y llévelo a acabo.

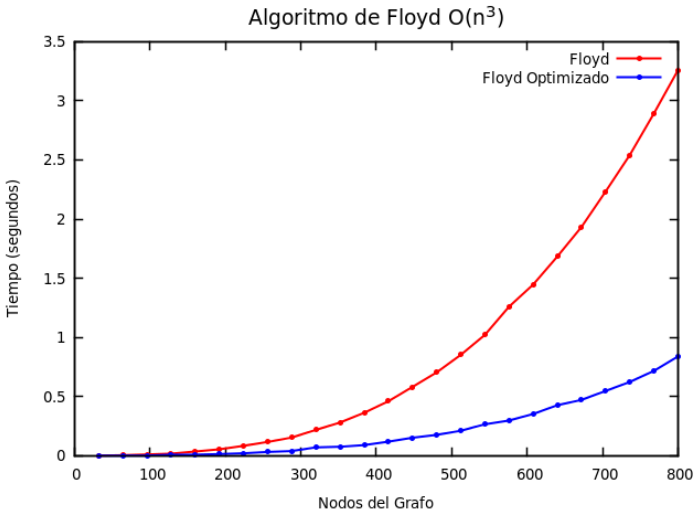
Comparación de ejecuciones con y sin optimización

A continuación se muestran las comparaciones de los algoritmos dados para distintos niveles de optimización: sin optimización, y con la optimización *O2* proporcionada por el compilador de *g++*.

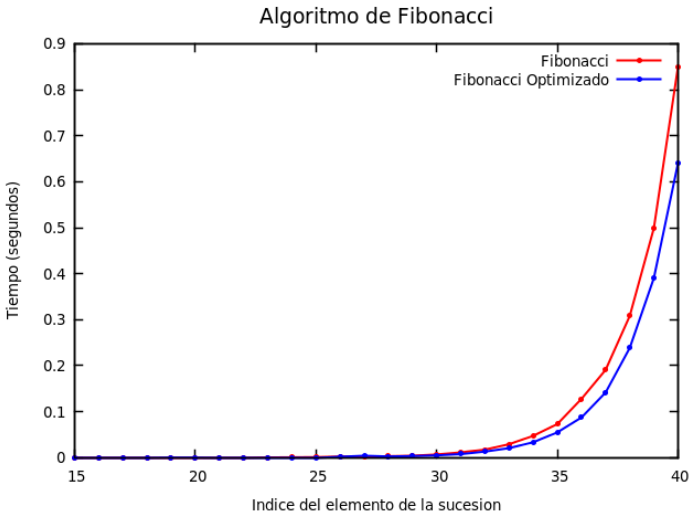
Optimización de los algoritmos cuadráticos



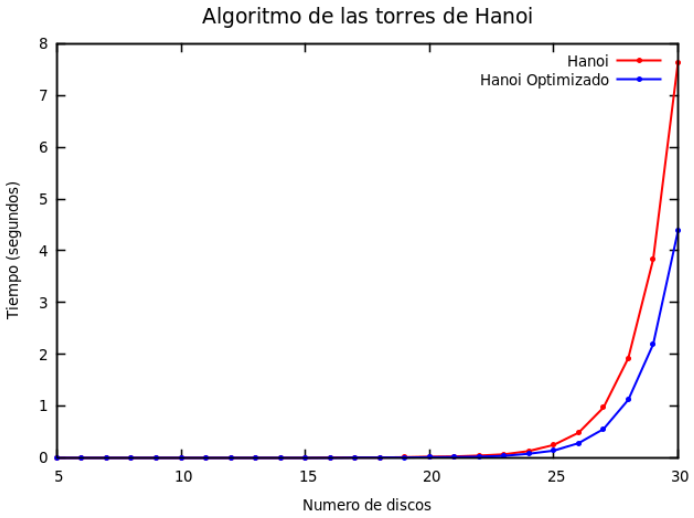
Optimización del algoritmo de Floyd:



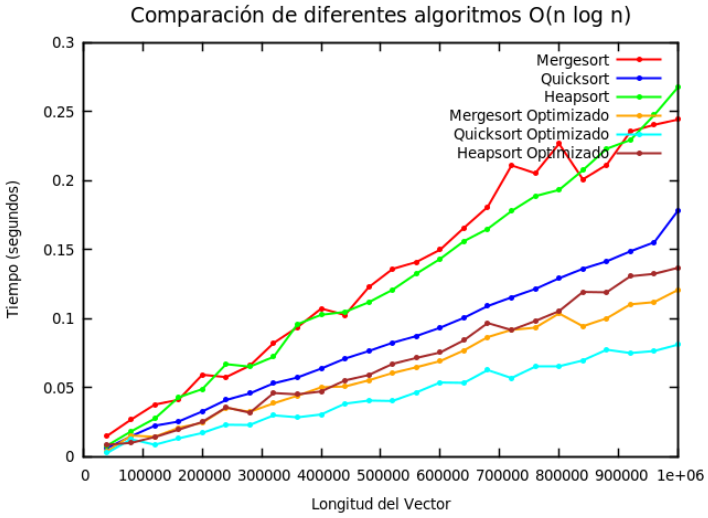
Optimización del algoritmo de Fibonacci:



Optimización del algoritmo de las torres de Hanoi:



Optimización de los algoritmos de ordenación $O(n \log n)$



Comparación de ejecuciones entre los componentes del grupo.

En ésta última sección, vamos a comparar las ejecuciones realizadas en los 4 ordenadores de los componentes del grupo, y veamos las diferencias. Para ello, dentro de cada apartado, vamos a realizar una tabla de tres filas, teniendo cada una los datos para cierto n fijado de antemano.

Características de los ordenadores

Se proporcionan las características del ordenador de cada integrante del grupo.

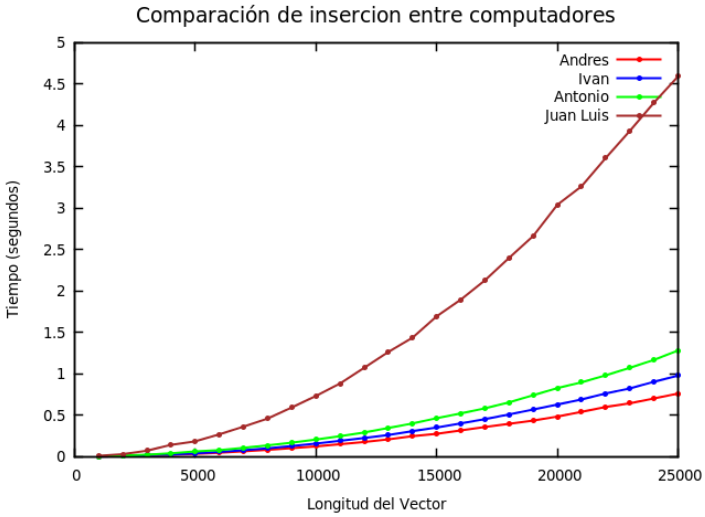
- **Andrés :** Toshiba - 8 GB de RAM - Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
- **Iván :** HP - 4 GB de RAM - Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz

- **Antonio :** Acer - 4 GB de RAM - Intel(R) Core(TM) i5 CPU M 450 @ 2.40GHz
- **Juan Luis :** Olidata - 1 GB de RAM - Intel(R) Atom(TM) CPU N450 @ 1.66GHz

Comparación: algoritmos cuadráticos

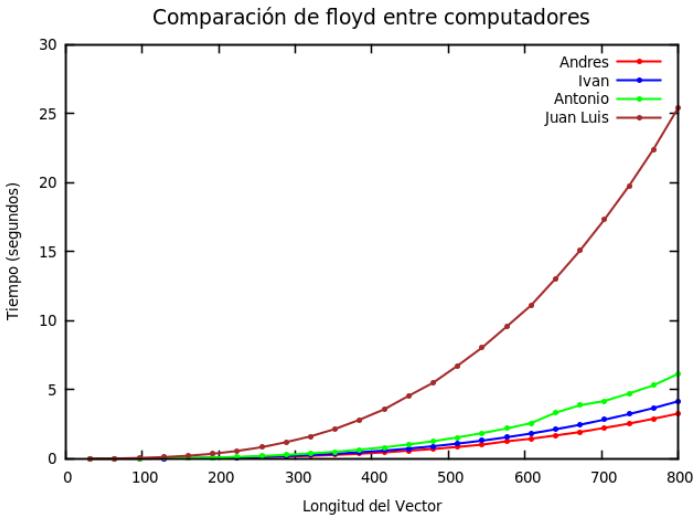
En el caso de los cuadráticos, cogemos el método de inserción.

Tamaño del Vector	Andrés	Antonio	Iván	Juan Luis
1000	0.001321	0.003619	0.001556	0.010846
13000	0.209278	0.34684	0.262103	1.26515
25000	0.762199	1.28187	0.978268	4.59284



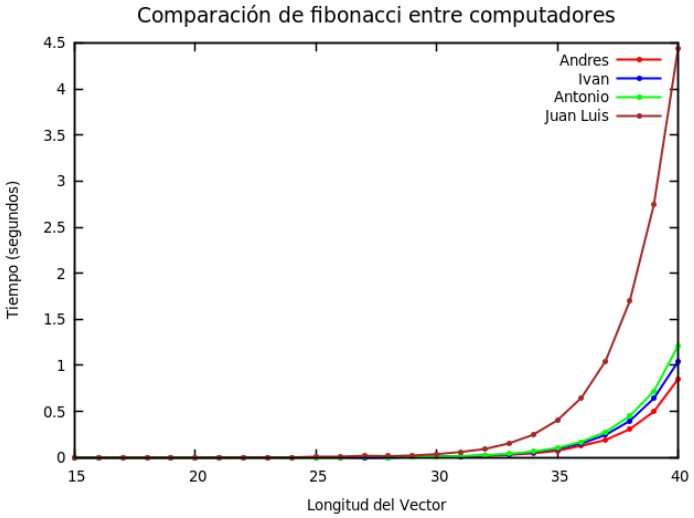
Comparación: algoritmos cúbicos

Nodos del grafo	Andrés	Antonio	Iván	Juan Luis
32	0.000596	0.000961	0.000298	0.001617
416	0.460287	0.825125	0.586223	3.57139
800	3.25971	6.14008	4.15313	25.4081



Comparación: algoritmo de Fibonacci

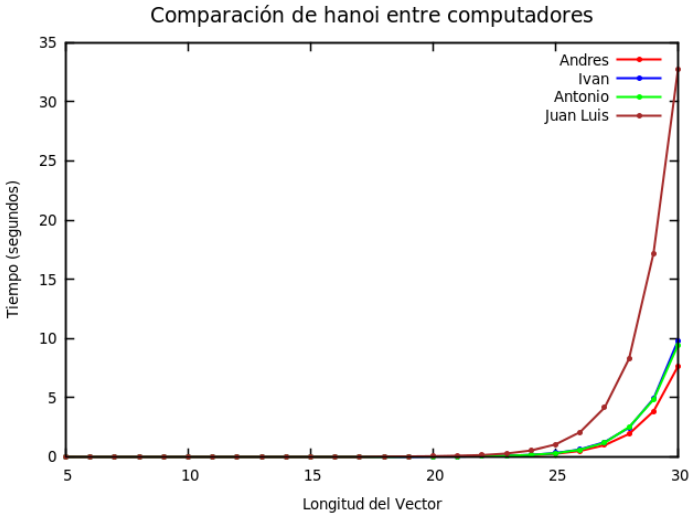
Índice	Andrés	Antonio	Iván	Juan Luis
15	1.3e-05	1.5e-05	7e-06	3.2e-05
27	0.002394	0.004638	0.003859	0.020256
40	0.849934	1.21011	1.03951	4.43443



Comparación: algoritmo de las Torres de Hanoi

Número de discos	Andrés	Antonio	Iván	Juan Luis
5	1e-06	1e-06	1e-06	4e-06
17	0.002302	0.002542	0.00182	0.005348

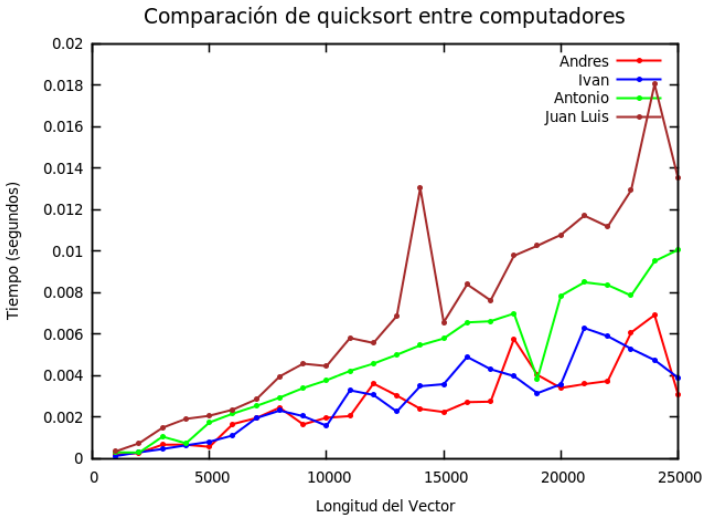
Número de discos	Andrés	Antonio	Iván	Juan Luis
30	7.63996	9.44513	9.81995	32.7569



Comparación: algoritmos n logn

El algoritmo que comparamos es el quicksort.

Tamaño del Vector	Andrés	Antonio	Iván	Juan Luis
40000	0.006235	0.012287	0.007532	0.03168
520000	0.082585	0.118127	0.103365	0.272054
1000000	0.178312	0.229693	0.204994	0.583711



Comentarios

Tras ver los resultados de las distintas tablas, observamos que las prestaciones del ordenador de Andrés son superiores a los demás, y que la máquina utilizada por Juan Luis es muy lenta, que era lo que se esperaba previamente al experimento. Aun así, pese a la mejora de las prestaciones del ordenador de Andres, es claro que si el algorítmo es cúbico o exponencial, el tiempo de espera puede ser eterno para un n lo suficientemente grande, como es el caso de las torres de hanoi. Igualmente, un algorítmo muy rápido como el quicksort al compararlo en varias máquinas vemos que tampoco difiere mucho en cuanto a tiempos y que es mucho más importante este hecho que las prestaciones.