Práctica #1
Algorímtica

Valentino Lugli

Marzo 2020

Contenido

1 Cálculo empírico de la eficiencia

Los algoritmos que se han seleccionado para la realización de ésta práctica son: Burbuja, Quicksort, Floyd y Hanoi.

A continuación se presentan las tablas obtenidas de medir el tiempo de ejecución de cada algoritmo con diferentes números de entradas.

Algoritmo Burbuja $\in O(n^2)$

Nro. Elementos	Tiempo (s)
1000	0.0032496
2000	0.0111561
3000	0.0319347
4000	0.0468937
5000	0.0760799
6000	0.113508
7000	0.160327
8000	0.214343
9000	0.278918
10000	0.350058
11000	0.427935
12000	0.516449
13000	0.614826
14000	0.72116
15000	0.831686
16000	0.953205
17000	1.08103
18000	1.22873
19000	1.37308
20000	1.53009
21000	1.68375
22000	1.85655
23000	2.03307
24000	2.23305
25000	2.4166

Algoritmo Quicksort $\in O(n \cdot \log(n))$

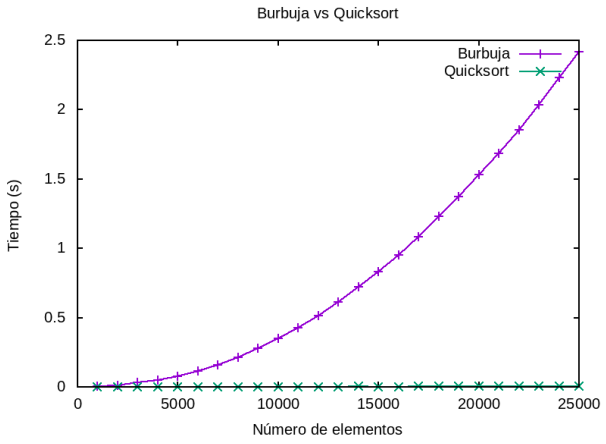
Nro. Elementos	Tiempo (s)
1000	0.0002648
2000	0.0004402
3000	0.0005153
4000	0.0005654
5000	0.0006764
6000	0.0008489
7000	0.0009703
8000	0.0011378
9000	0.0015307
10000	0.0014679
11000	0.0016161
12000	0.0017974
13000	0.0027016
14000	0.003028
15000	0.0025306
16000	0.0027122
17000	0.0029093
18000	0.0030888
19000	0.0035398
20000	0.0037948
21000	0.0041787
22000	0.0048082
23000	0.0049847
24000	0.005166
25000	0.0049595

Algoritmo Floyd $\in O(n^3)$	
Nro. Nodos	Tiempo (s)
100	0.008967
200	0.065512
300	0.224666
400	0.477144
500	0.943344
600	1.62377
700	2.56918
800	3.80467
900	5.46955
1000	7.55041
1100	10.0186
1200	13.0281
1300	16.5942
1400	20.6334
1500	25.3295
1600	30.7256
1700	36.7912
1800	43.6263
1900	51.3192
2000	59.7982
2100	68.9669
2200	79.2746
2300	90.5534
2400	102.626
2500	115.808

Algoritmo Hanoi $\in O(2^n)$	
Nro. Discos	Tiempo (s)
6	3e-06
7	3e-06
8	7e-06
9	7e-06
10	1e-05
11	2e-05
12	3.6e-05
13	7.1e-05
14	0.000139
15	0.000275
16	0.000548
17	0.001674
18	0.00288
19	0.006262
20	0.012519
21	0.024698
22	0.048053
23	0.089055
24	0.161288
25	0.317206
26	0.578443
27	1.11731
28	2.25221
29	4.47941
30	8.91821

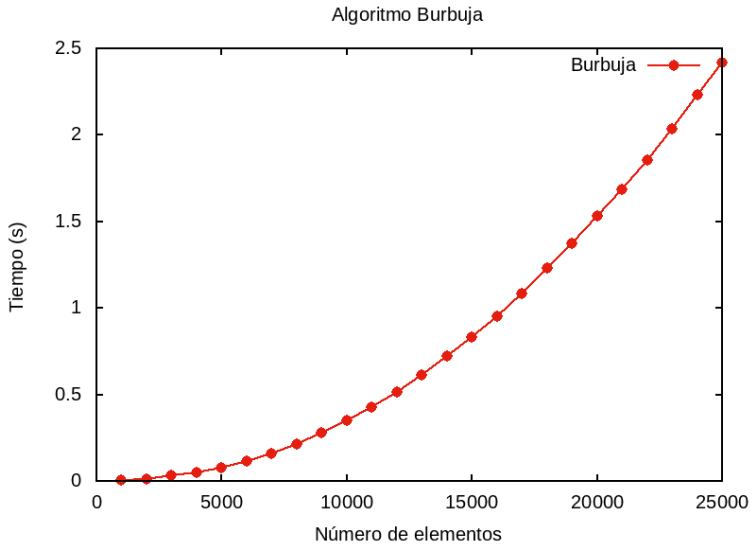
2 Graficando los tiempos

2.1 Gráfica comparativa entre algoritmos Burbuja y Quicksort



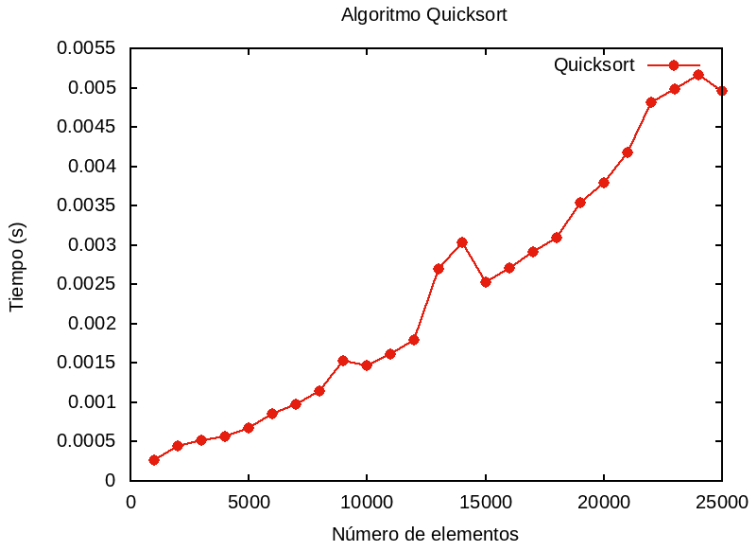
Como es de esperarse, el algoritmo Burbuja tarda mucho más que Quicksort para ordenar una cantidad idéntica de datos, en esta escala apenas se logra apreciar el crecimiento que Quicksort, que más bien parece una línea recta comparado con Burbuja. Cabe destacar que para estos algoritmos, el código se modificó para que se realizaran varias pruebas con el mismo tamaño de entrada para obtener un valor más representativo, ya que el tiempo de ejecución puede ser dependiente de que tan bien ordenados se encuentran los elementos.

2.2 Gráfica del algoritmo Burbuja



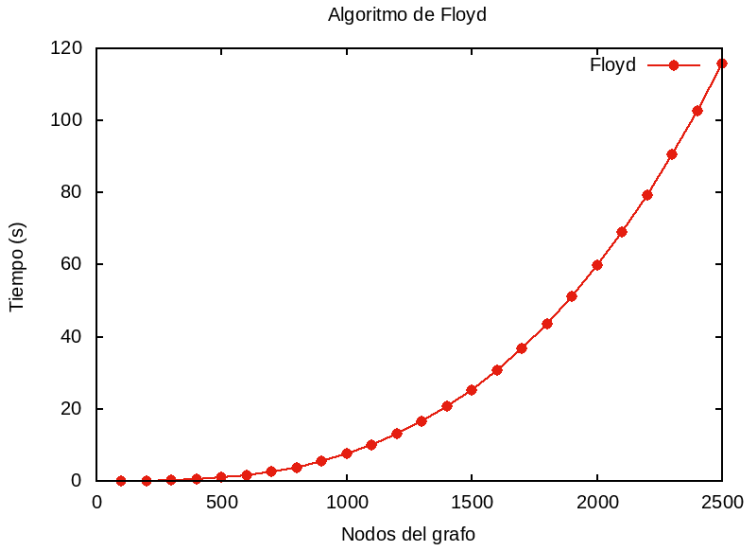
Aislando el gráfico de Burbuja, se aprecia en su totalidad la característica cuadrática del tiempo que toma en ejecutar valores cada vez más grandes. No ayuda que es el algoritmo cuadrático que más intercambios que realiza en una lista ordenada aleatoriamente, esto a su vez empeora más el rendimiento.

2.3 Gráfica del algoritmo Quicksort



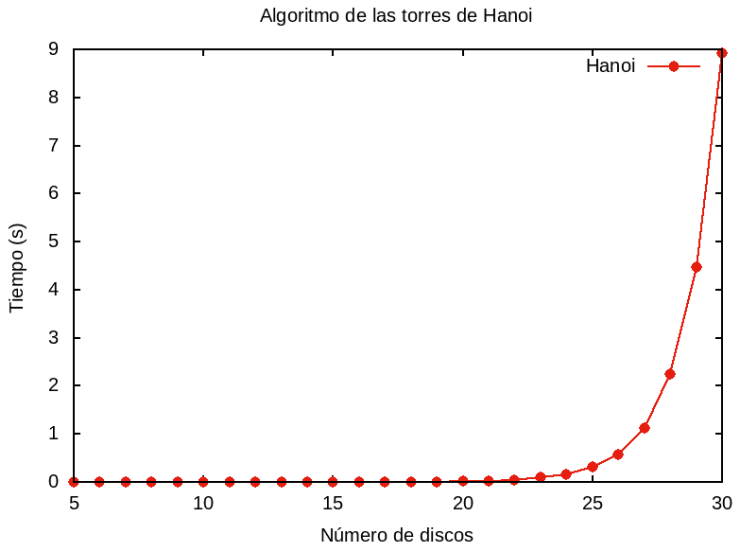
Ya aislado de Burbuja, se puede apreciar mejor el crecimiento de Quicksort a medida que aumentan los elementos, nótese que inclusive para ordenar 25000 elementos no ha tomado ni la mitad de un segundo.

2.4 Gráfica del algoritmo de Floyd



La naturaleza cúbica de este algoritmo de búsqueda de caminos en grafos destaca más al tomar en cuenta que se tuvo que reducir diez veces el tamaño de la entrada y aún así, el último caso con 2500 nodos ha tomado aproximadamente dos minutos. Comparemos eso con Burbuja, que siendo de orden cuadrático, para la misma cantidad de datos toma menos de la mitad de un segundo en ordenarlos.

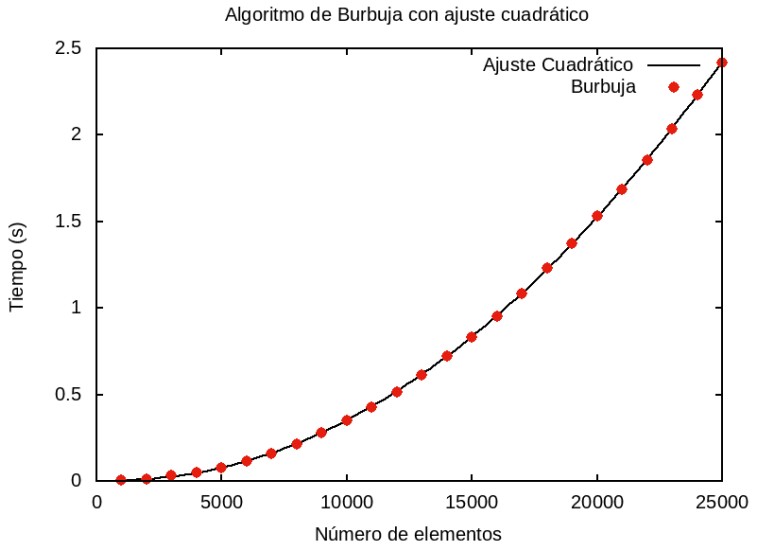
2.5 Gráfica del algoritmo de las Torres de Hanoi



La diferencia en tiempo de ejecución de un algoritmo exponencial con el resto de los estudiados es abismal, para tan solo 30 discos el algoritmo ya toma 10 segundos en, es importante y al mismo tiempo algo impresionante es que doblando la entrada a 60 discos, el algoritmo tomaría aproximadamente tres siglos en finalizar, basado en los cálculos realizados en el siguiente apartado.

3 Calculando la eficiencia híbrida

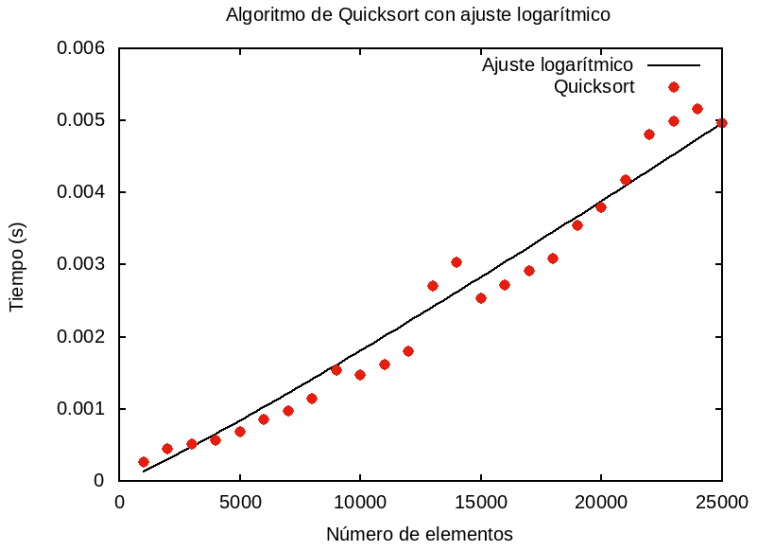
3.1 Gráfica del ajuste del algoritmo Burbuja




```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 4.16085e-09      +/- 1.62e-11     (0.3893%)
a1              = -7.52417e-06     +/- 4.339e-07    (5.767%)
a2              = 0.00977364       +/- 0.002448     (25.05%)
```

Como era de esperarse, la función cuadrática se ajusta bastante bien a la curva generada por el algoritmo de Burbuja.

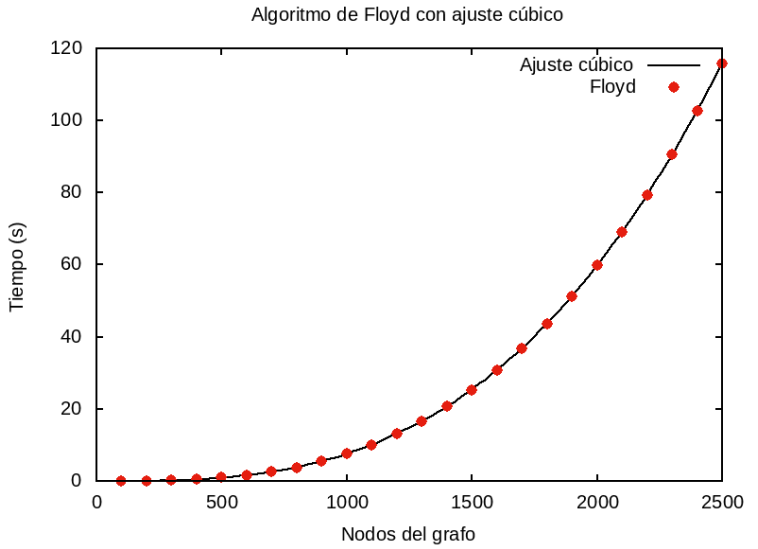
3.2 Gráfica del ajuste del algoritmo Quicksort




```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 1.95921e-08      +/- 3.987e-10    (2.035%)
```

Aunque la gráfica del algoritmo posee ciertas fluctuaciones, el ajuste logarítmico la logra representar bastante bien la gráfica generada por Quicksort.

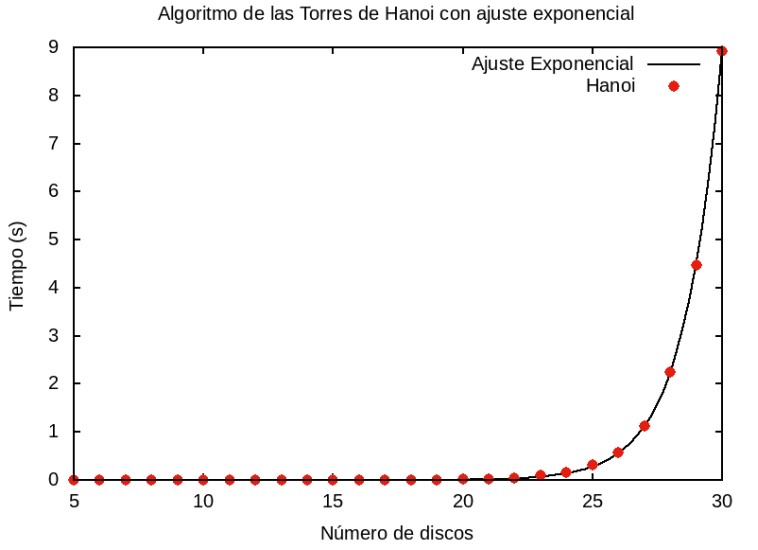
3.3 Gráfica del ajuste del algoritmo de Floyd




```
Final set of parameters          Asymptotic Standard Error
=======================          ==========================
a0              = 7.09657e-09    +/- 3.271e-11    (0.4609%)
a1              = 1.08449e-06    +/- 1.292e-07    (11.92%)
a2              = -0.000763825   +/- 0.0001461    (19.13%)
a3              = 0.126457       +/- 0.04473      (35.37%)
```

El algoritmo se ajusta de buena manera a la función cúbica, de nuevo llevando a la luz lo rápido que los tiempos de ejecución aumentan a medida que aumenta el tamaño de la entrada.

3.4 Gráfica del ajuste del algoritmo de las Torres de Hanoi

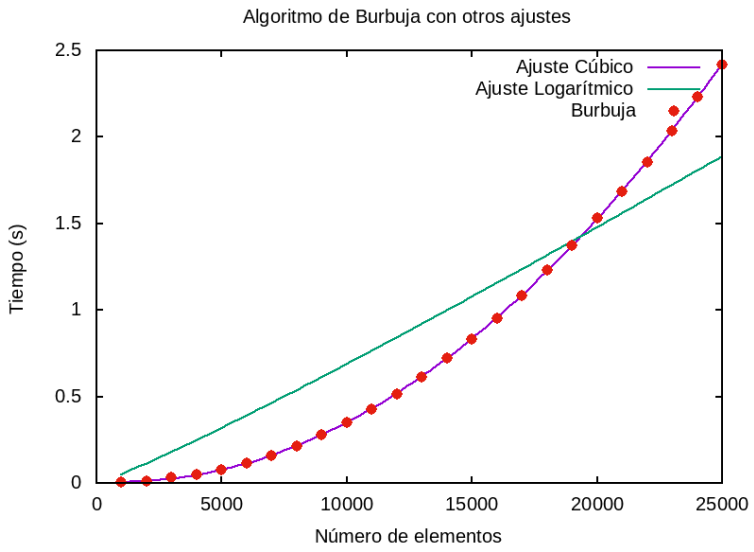



```
Final set of parameters          Asymptotic Standard Error
=======================          ==========================
a0              = 8.31915e-09    +/- 9.793e-12    (0.1177%)
```

Nuevamente, el algoritmo logra ajustarse muy bien al ajuste exponencial, esto también logra mostrar que aún con una constante oculta tan pequeña, el tiempo de ejecución se dispara, recordemos que tomarían 3 siglos para que este algoritmo finalizara de mover 60 discos.

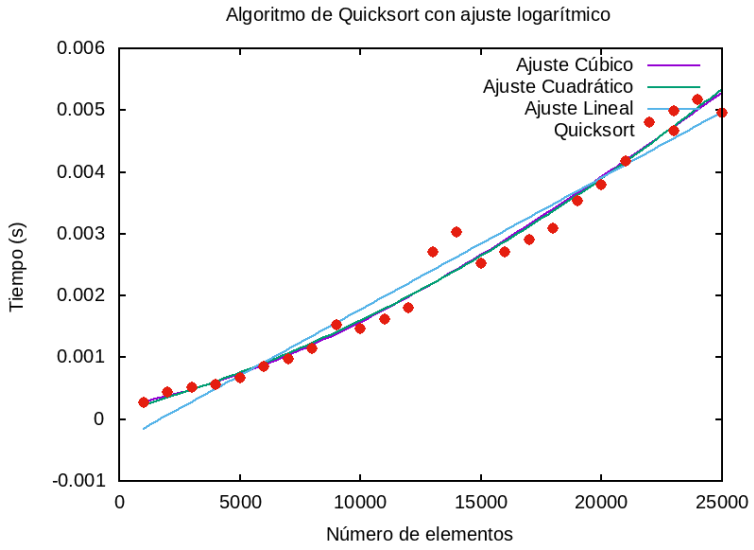
3.5 Probando otros ajustes

3.5.1 Algoritmo Burbuja



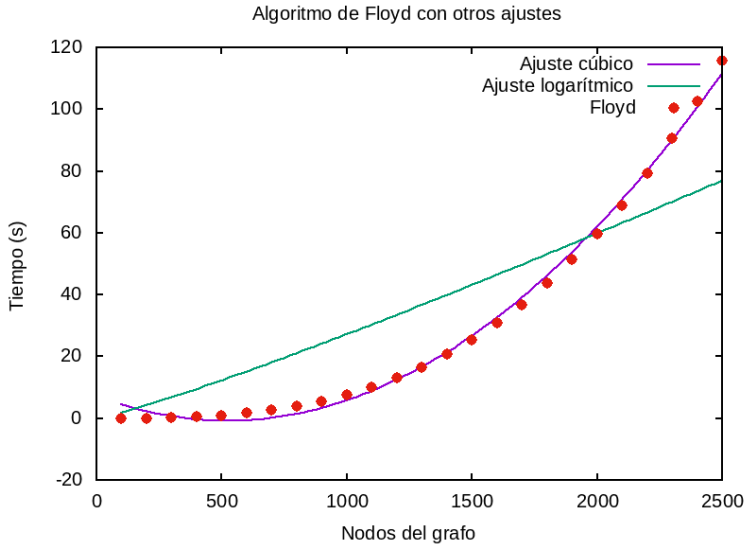
Como era de esperarse, un ajuste logarítimico no puede ajustarse a uno cuadrático, pero un cúbico sí puede hacerlo relativamente bien, se ajusta a los datos de Burbuja sin mucha diferencia de un ajuste cuadrático.

3.5.2 Algoritmo Quicksort



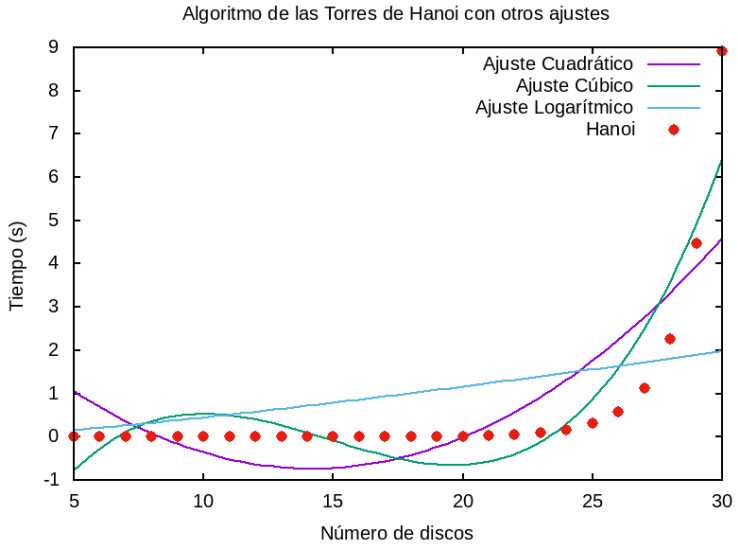
Los ajustes cúbico y cuadrático intentan justarse pero se puede notar levemente las características polinómicas de las funciones, ya que se curvan a los datos. Como nota interesante, una función lineal se ajusta bastante bien a los datos.

3.5.3 Algoritmo de Floyd



En este ejemplo es interesante notar como el ajuste cuadrático logra una ligera semejanza a los datos.

3.5.4 Algoritmo de las Torres de Hanoi

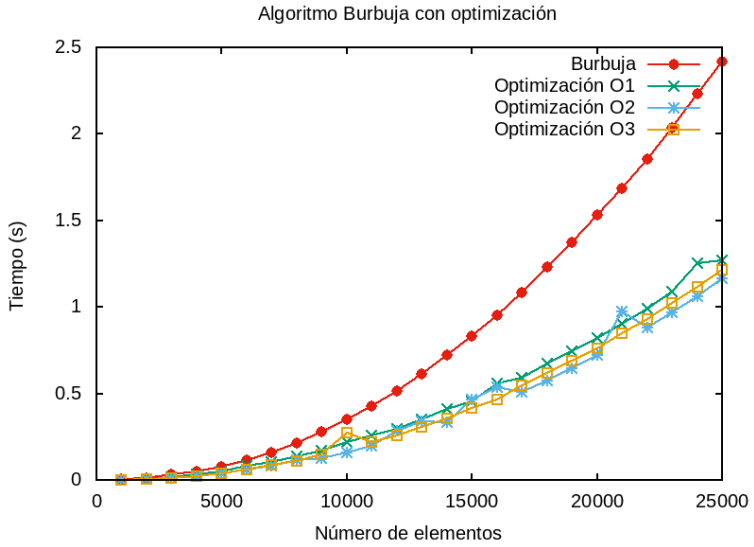


¡No hay nada que pueda compararse con el crecimiento astronómico del algoritmo exponencial de Hanoi! Se puede ver como las funciones polinómicas pueden ligeramente asemejarse al crecimiento pero al final no pueden crecer lo suficiente como para ajustarse a los datos. Cabe notar que un ajuste exponencial en los otros algoritmos arrojaba una variable indefinida.

4 Otras pruebas y detalles

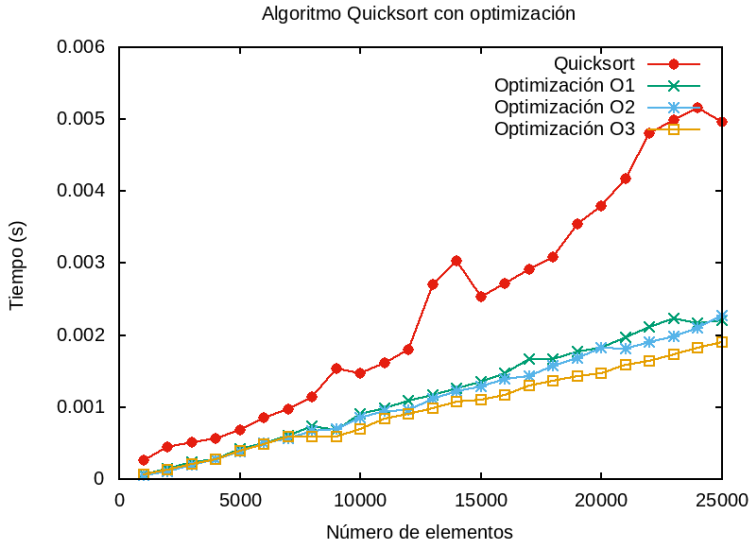
4.1 Comparación de los algoritmos con diferentes niveles de optimización

4.1.1 Optimizando el Algoritmo Burbuja



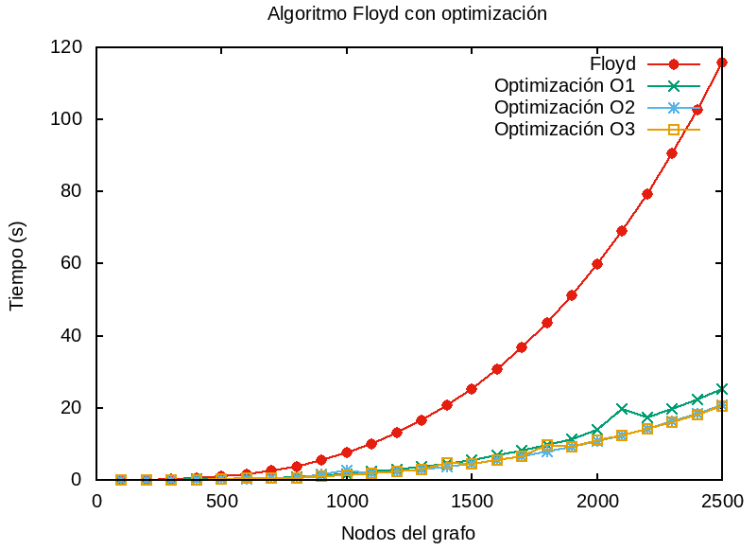
Resulta curioso como los niveles de optimización no difieren mucho entre si, y curiosamente, el nivel O2 resulta ligeramente mejor que la optimización más extrema de O3. Esto sin embargo no evita que la forma que describen los tiempos optimizados sean igualmente cuadráticos, pero con variables ocultas de menor valor.

4.1.2 Optimizando el Algoritmo Quicksort



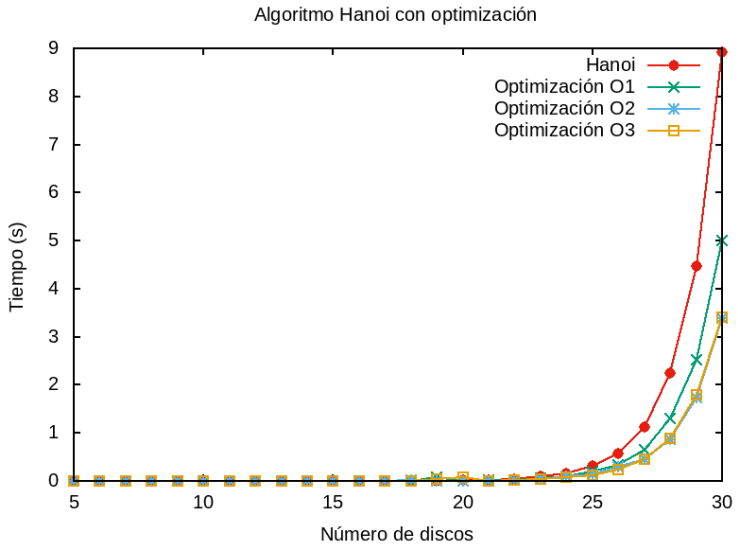
Nuevamente, se ve una mejora ligera en los tiempos de ejecución y en este caso se nota que cada nivel de optimización ligeramente mejora sobre el anterior, y aunque se sigue describiendo un algoritmo de forma logarítmica.

4.1.3 Optimizando el Algoritmo de Floyd



De todos los algoritmos evaluados, este es el que ha tenido las mejoras más significativas, posiblemente porque la manera que está construido permite al compilador agilizar y compactar el código objeto resultante, esto como ya se ha visto, no evita que sea un algoritmo de orden cúbico.

4.1.4 Optimizando el Algoritmo de Hanoi



Finalmente, llegamos al algoritmo de las torres de Hanoi, si bien las versiones optimizadas toman menos tiempo, la verdad es que la diferencia es bastante menor al respecto del resto de los algoritmos, y claramente se puede ver como todos siguen teniendo forma exponencial.

4.2 Prestaciones del ordenador utilizado

Todas las pruebas plasmadas en esta memoria fueron ejecutadas en **Ubuntu 19.18 64 bits** ejecutándose en un ordenador portátil marca VIT P2412, el cual posee un CPU **Intel i3 4000M @ 2.40 GHz**, con **16 GB** de memoria RAM.

5 Conclusiones

Lo que se puede concluir luego de la realización de este trabajo es que tan sumamente importante es la eficiencia de un algoritmo y cómo el diseño de un algoritmo puede realizar cambios tan extremos como los vistos, por ejemplo, entre Burbuja y Quicksort: ambos desempeñan la misma tarea pero uno es masivamente más rápido, o bien, lo rápido que una entrada de un tamaño relativamente pequeño para el algoritmo de Floyd o Hanoi tarda demasiado como para ser evaluada. Esto es algo más fundamental que el hecho de poseer una máquina más rápida o de optimizar la compilación, que igualmente ayudan a reducir el tiempo por medio de las constantes ocultas pero, el buen diseño de un algoritmo produce mejoras fundamentales, aquellas en que el orden se reduce a uno menor y por ende, se puede lograr obtener un algoritmo mucho más eficiente siempre que las constantes ocultas no sean excesivamente grandes.