UNIVERSIDAD DE GRANADA

Práctica 1: Eficiencia

Análisis de eficiencia de algoritmos con diferentes órdenes de eficiencia

José María Gómez García
Fernando Lojano Mayaguari
Valentino Lugli
Carlos Mulero Haro

Universidad de Granada
Granada
Marzo 2020

Índice

1 Introducción

En esta entrega se nos pide analizar la eficiencia de una serie de algoritmos: Bubblesort, Insertion Sort, Selection Sort, Mergesort, Quicksort, Heapsort, Floyd y Hanoi.

Primero vamos a comentar la eficiencia empírica de los algoritmos, comparando también los resultados de los algoritmos que presentan la misma eficiencia teórica.

A continuación analizaremos la eficiencia híbrida de cada algoritmo, valorando si el ajuste que nos proporciona gnuplot es bueno o malo.

Además realizaremos un análisis de como varían los tiempos de ejecución de los algoritmos anteriormente mencionados en función de la opción de compilación elegida (-O0, -O1, -O2, -O3, ninguna opción).

Finalmente mostraremos las gráficas que resultan de tomar los tiempos de ejecución de un mismo algoritmo en diferentes máquinas para así valorar la diferencia en tiempo de ejecición en función de las características de los diferentes ordenadores.

Un dato a tener en cuenta es que el ordenador en el cual se ha realizado el análisis de los algoritmos (Cálculo empírico y Eficiencia Híbrida) tiene las siguientes especificaciones:

- Procesador Intel Core i5-1035G1
- Memoria RAM de 8 GB
- Sistema Operativo Windows 10 con consola de Linux para Windows (Cywin)

2 Cálculo Empírico

2.1 O(n^2)

2.1.1 Bubblesort

Bubblesort es un algoritmo con eficiencia O(n^2). Esto se debe a que realiza un recorrido completo del vector, comparando cada elemento con su adyacente posterior, intercambiando sus posiciones si están en el orden equivocado y repitiendo este proceso hasta que no sean necesarios más intercambios.

Tabla de datos

A continuación se va a presentar la tabla con los datos obtenidos al realizar el estudio de Bubblesort. El rango de los mismos es 4000-100000 con un valor de 4000 en cada incremento a medida que va aumentando el tamaño de los datos:

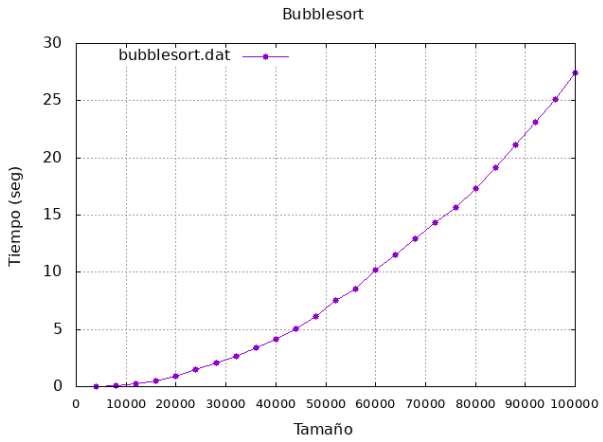
Tamaño	Tiempo (seg)
4000	0.0256
8000	0.10372
12000	0.265
16000	0.515
20000	0.921
24000	1.453
28000	2.046
32000	2.671
36000	3.421
40000	4.156
44000	5.078
48000	6.171
52000	7.578

Tamaño	Tiempo (seg)
56000	8.531
60000	10.202
64000	11.546
68000	12.937
72000	14.312
76000	15.656
80000	17.343
84000	19.171
88000	21.171
92000	23.093
96000	25.14
100000	27.39

En ella podemos apreciar como a medida que aumentamos el tamaño, el tiempo aumenta drásticamente, en este caso, podemos ver como en la última entrada con un tamaño de 100000 obtenemos un tiempo de 27.39 segundos.

Representación de datos

A continuación vamos a presentar todos los resultados del algoritmo Bubblesort en una misma gráfica.



En esta gráfica podemos apreciar que, aunque existen ciertas imperfecciones, la traza de los datos se asemeja bastante a los de una función cuadrática, algo bastante lógico teniendo en cuenta que el orden de eficiencia de este algoritmo es cuadrático.

2.1.2 Insertion Sort

El algoritmo Insertion Sort se trata de otro caso de orden de eficiencia O(n^2). El comportamiento de este consiste en realizar un recorrido por el vector dejando una parte de esta ordenada y en cada iteración inserta el elemento que se está analizando en la posición correcta de la parte ordenada. Este proceso se repetirá hasta llegar al final del vector quedando este completamente ordenado.

Tabla de datos

Como entrada de datos para este programa se han utilizado también 25 entradas con un tamaño dentro del rango 4000-100000.

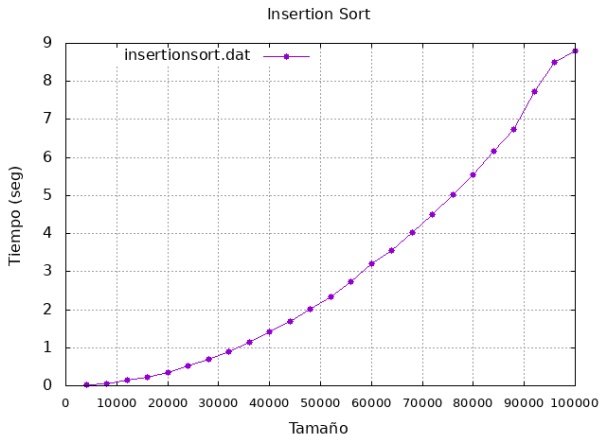
Tamaño	Tiempo (seg)
4000	0.01372
8000	0.05624
12000	0.14
16000	0.235
20000	0.359
24000	0.515
28000	0.687
32000	0.906
36000	1.14
40000	1.406
44000	1.703
48000	2.015
52000	2.344

Tamaño	Tiempo (seg)
56000	2.734
60000	3.203
64000	3.546
68000	4.031
72000	4.5
76000	5.015
80000	5.546
84000	6.156
88000	6.734
92000	7.734
96000	8.5
100000	8.796

Al igual que en el caso anterior, podemos ver como el tiempo aumenta con el tamaño de datos. No obstante, observamos que el tiempo en la última entrada se ha reducido drásticamente pues con el tamaño de 100000 obtenemos un tiempo de 8.796 segundos.

Representación de datos

Con estos datos obtenemos una gráfica de la siguiente forma.



Teniendo en cuenta su funcionamiento y el hecho de que se trata de un algoritmo de orden cuadrático, la trayectoria de su curva es parecida a la de una función cuadrada.

2.1.3 Selection Sort

Selection Sort es un algoritmo de ordenamiento que requiere O(n^2) operaciones para ordenar una lista de n elementos. Su comportamiento consiste en localizar el menor elemento de un vector intercambiándolo por el primero y buscar el menor elemento siguiente en el vector e intercambiarlo en la posición actual analizada. Este proceso se repetirá sucesivamente con el resto de las posiciones del vector.

Tabla de datos

Al igual que en los casos anteriores, usaremos entradas de datos con un tamaño delntro del rango 4000-100000. Obtenemos como resultado:

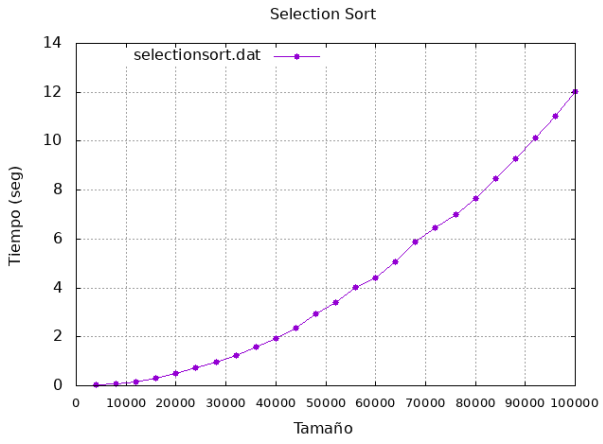
Tamaño	Tiempo (seg)
4000	0.01936
8000	0.07812
12000	0.172
16000	0.312
20000	0.5
24000	0.718
28000	0.953
32000	1.25
36000	1.578
40000	1.953
44000	2.359
48000	2.937
52000	3.39

Tamaño	Tiempo (seg)
56000	4.015
60000	4.39
64000	5.062
68000	5.89
72000	6.468
76000	7.015
80000	7.656
84000	8.453
88000	9.281
92000	10.14
96000	11.031
100000	12.015

Al analizar esta tabla en comparación con el resto, podemos ver que se trata de una especie de termino medio, ya que, si bien con entradas pequeñas el tiempo es similar al del algoritmo insertion sort, en entradas mas grandes el tiempo se hace considerablemente mayor que insertion sort, pero definitivamente tiene un tiempo menor que bubblesort.

Representación de datos

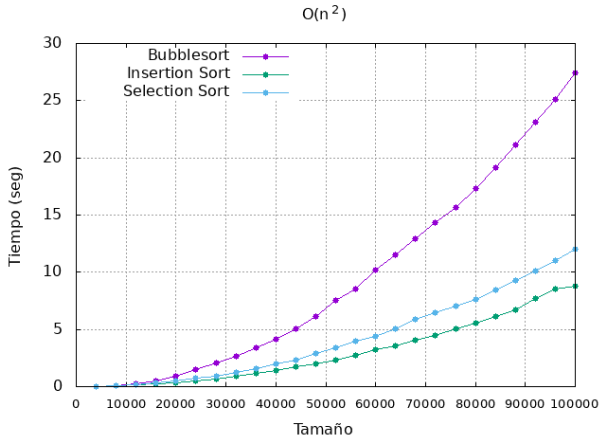
Usando el programa de ejecución de terminal gnuplot, obtenemos la siguiente gráfica:



Podemos observar en la gráfica, teniendo en cuenta su orden cuadrático, que la curva generada es nuevamente similar a una función cuadrada.

2.1.4 Comparación entre algoritmos O(n^2)

Una vez analizados los algoritmos por separados, vamos a representar sus datos en una sola gráfica y analizar así los tres a la vez.



En esta gráfica podemos ver como para tamaños pequeños, los tres algoritmos tienen un tiempo parecido. No obstante, a medida que vamos aumentando el tamaño, se va notando una diferencia considerable en el algoritmo bubblesort, pues su tiempo va creciendo de una forma bastante notable, llegando a un punto en el que se encuentra muy lejos de insertion sort y selection sort.
En cuanto a insertion sort, apreciamos como, en comparación con los otros dos, es el más eficiente, pues su rango de tiempo se mantiene por debajo del rango de los demás, dejando a selection sort como término medio en cuanto a tiempo entre estos 3 algoritmos.

2.2 O(nlogn)

2.2.1 Mergesort

Mergesort es un algoritmo de ordenamiento basado en la técnica divide y vencerás con orden de eficiencia O($nlogn$).Es un algoritmo de ordenamiento por mezcla que consiste en dividir el vector por la mitad y a la vez estas dos nuevas partes dividirlas por sus respectivas mitades. Este proceso se repite hasta que las partes divididas estén compuestas por un sólo elemento. Una vez llegado a este punto, los elementos individuales se vuelven a juntar en pares de elementos, esta vez ordenados. Volveremos a mezclar todas las partes (ordenadas) de la misma forma en la que se dividieron en la primera fase hasta llegar a la primera división. En la última división, todas las partes(ordenadas) mezcladas se reestructurarán hasta obtener el vector ordenado.

Tabla de datos

Para analizar el comportamiento de este algoritmo se van a utilizar entradas de tamaño dentro del rango 2000000-50000000 con incrementos de dos millones en cada iteración.

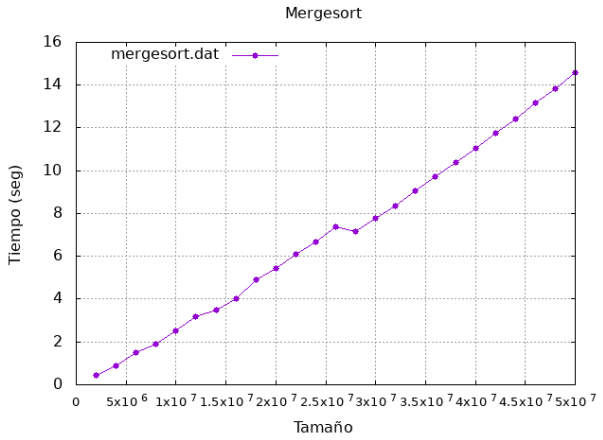
Tamaño	Tiempo (seg)
2000000	0.437
4000000	0.906
6000000	1.499
8000000	1.906
10000000	2.499
12000000	3.204
14000000	3.485
16000000	4.031
18000000	4.906
20000000	5.454
22000000	6.078
24000000	6.656
26000000	7.375

Tamaño	Tiempo (seg)
28000000	7.157
30000000	7.765
32000000	8.344
34000000	9.079
36000000	9.719
38000000	10.406
40000000	11.047
42000000	11.749
44000000	12.406
46000000	13.187
48000000	13.827
50000000	14.594

Como podemos ver aunque el resultado del tiempo en su última ejecución es de más de 14 segundos, el tiempo de este algoritmo es mucho mejor que cualquiera de los tres anteriores usados, pues en este análisis comenzamos con una entrada de tamaño 2000000 y un resultado en tiempo de menos de medio segundo.
Contrastando los tiempos obtenidos en los algoritmos anteriores, en los cuales la entrada máxima es de 1000000, se aprecia que la diferencia es notable pues el mejor tiempo supera los 8 segundos(obtenido en insertion sort).

Representación de datos

Una vez presentados los datos, los dibujaremos en una gráfica



Podemos observar como en la gráfica, aunque existen ciertas imperfecciones, la trayectoria de los datos es similar a la de una función casi lineal. Esto es lógico, ya que el orden de eficiencia de Mergesort es casi lineal.

2.2.2 Quicksort

Quicksort se trata de otro algoritmo de eficiencia O($n \log n$).
Basado en la técnica Divide y vencerás, el funcionamiento de este algoritmo consiste en elegir un elemento del vector (pivote) a partir del cual, se van dejando en un lado todos los valores menores que el pivote y en el otro los mayores. Repetiremos esta acción recursivamente para todos los subconjuntos que contengan mas de un elemento. Al terminar, obtendremos un vector ordenado. Este es de hecho, el método más rápido y eficiente de los métodos de ordenación.

Tabla de datos

Al igual que para mergesort, en este caso también se utilizarán datos dentro del rango 2000000-50000000.

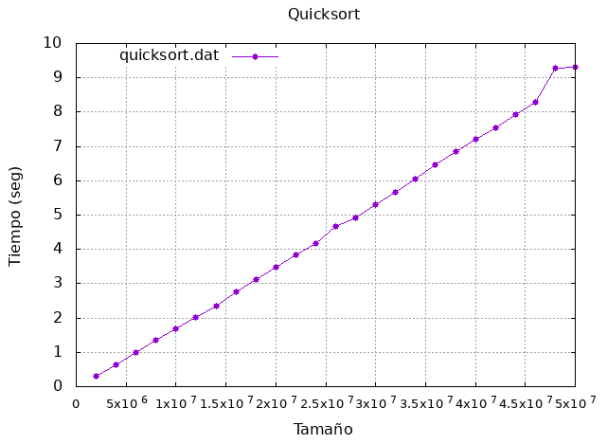
Tamaño	Tiempo (seg)
2000000	0.313
4000000	0.641
6000000	0.985
8000000	1.344
10000000	1.672
12000000	2.015
14000000	2.359
16000000	2.75
18000000	3.125
20000000	3.468
22000000	3.829
24000000	4.172
26000000	4.656

Tamaño	Tiempo (seg)
28000000	4.907
30000000	5.297
32000000	5.656
34000000	6.062
36000000	6.453
38000000	6.844
40000000	7.203
42000000	7.547
44000000	7.922
46000000	8.296
48000000	9.281
50000000	9.297

Observamos que en este análisis los tiempos conseguidos son considerablemente mejor que en el caso de mergesort. De hecho, podemos asumir que los tiempos que conseguimos serán mucho mejores que todos los casos que analizaremos en este apartado (Orden O($nlogn$)), pues como se ha mencionado antes, quicksort se trata del algoritmo más rápido para ordenar valores.

Representación de datos

Representaremos la tabla en una gráfica. De esta forma:



Podemos ver que la trayectoria de estos datos también es casi lineal, aunque presenta ciertas imperfecciones, la más notable se observa al final de la misma.

2.2.3 Heapsort

Este es el último algortimo con orden O($nlogn$) que se va a analizar en este estudio. Heapsort basa su funcionamiento en la propiedad de los montículos, que se trata de un tipo de árbol binario que es completo, es decir que cada nivel debe estar lleno, con excepcion del último, que puede no estarlo. El proceso del algoritmo consiste en construir un montículo en orden contrario al de ordenación deseado. De esta forma, en la raiz del montículo nos queda el mayor o el menor elemento del vector (dependiendo del orden deseado). La raíz pasa a formar parte del vector ordenado, después, el último elemento del montículo pasa a ser el primero. Una vez llegado a este punto, la nueva raíz se intercambia con el elemento mayor/menor de cada nivel, de esta forma, la raíz vuelve a ser el el mayor/menor de todos los elementos del vector. Se vuelve a añadir la raíz al vector ordenado. Esto se repite hasta que nos queda el vector con todos los elementos completamente ordenados.

Tabla de datos

Con una entrada de datos con tamaños dentro del rango anteriormente usado para estudiar los algoritmos de orden O($nlogn$), obtenemos la siguiente tabla:

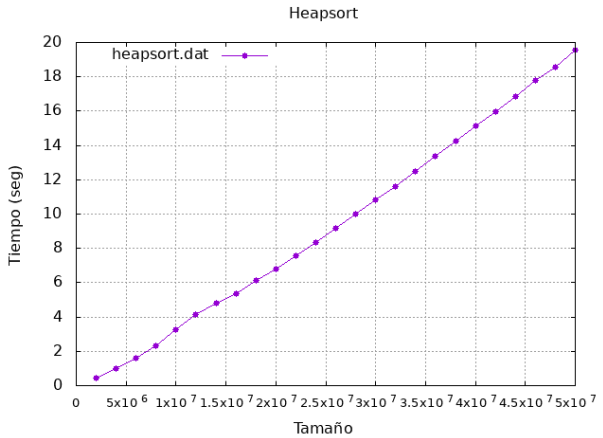
Tamaño	Tiempo (seg)
2000000	0.421
4000000	1
6000000	1.609
8000000	2.328
10000000	3.235
12000000	4.125
14000000	4.813
16000000	5.359
18000000	6.109
20000000	6.797
22000000	7.594
24000000	8.36
26000000	9.188

Tamaño	Tiempo (seg)
28000000	10.016
30000000	10.828
32000000	11.625
34000000	12.5
36000000	13.344
38000000	14.25
40000000	15.125
42000000	15.969
44000000	16.859
46000000	17.766
48000000	18.578
50000000	19.547

Comparando estos datos con los de los algoritmos anteriores, llegamos a la conclusión de que estamos ante el algoritmo con los peores resultados de tiempo. Esto se puede deber a que se trata de un algoritmo no recursivo y considerado como no estable.

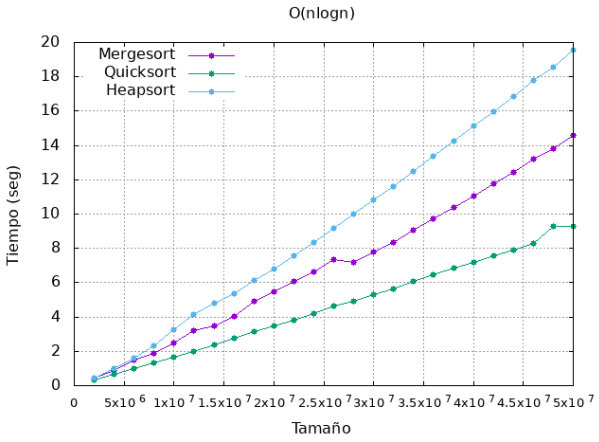
Representación de datos

Dada la tabla anterior, usando gnuplot obtenemos la siguiente gráfica:



2.2.4 Comparación entre algoritmos O($n \log n$)

Ya hemos estudiado de forma individual los algoritmos de ordenamiento de orden O($n \log n$). Ahora, como hicimos para los algoritmos cuadráticos, realizaremos una comparación de estos algoritmos representándolos en una gráfica para estudiar sus trazas. La gráfica obtenida es la siguiente:



Podemos ver que heapsort es el que mayor tiempo de todos obtiene, mientras que quicksort que como mencionamos antes es el mejor algoritmo de ordenacion interna, es el que menos tiempo obtiene. Esto nos deja a Mergesort como el termino medio de algoritmo de ordenación con orden casi lineal. Un dato a tener en cuenta es que, el peor algoritmo de estos tres (Heapsort) es mejor que todos los demás de orden cuadrático, pues los tiempos obtenidos en este análisis comienzan en 2000000 y no superan 1 segundo, mientras que en el análisis cuadrático el máximo valor de entrada es de 1000000 y el tiempo llega a superar los 8 segundos en el mejor algoritmo.

2.3 O(n^3) Floyd

El algoritmo de Floyd se trata de una algoritmo de análisis sobre grafos con orden de eficiencia O(n^3) que sirve para encontrar el camino mínimo en grafos dirigidos. Su objetivo principal es encontrar el camino entre cualquier par de vértices de un grafo.

Tabla de datos

Debido a que el orden de este algoritmo es O(n^3) podemos suponer que el tiempo de ejecución para entradas tan grandes como en las usadas para el estudio de algoritmos pertenecientes al orden O($nlogn$) será mucho mayor y tardará mucho tiempo en procesar todos esos datos.
Debido a esto, vamos a reducir el tamaño de forma que el rango de estudio será de 100-2500, con un incremento de 100 en cada iteración.

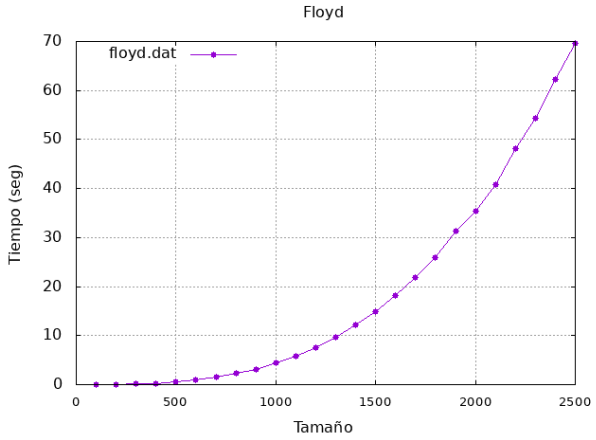
Tamaño	Tiempo (seg)
100	0
200	0.031
300	0.125
400	0.281
500	0.562
600	0.968
700	1.515
800	2.25
900	3.187
1000	4.36
1100	5.875
1200	7.61
1300	9.719

Tamaño	Tiempo (seg)
1400	12.156
1500	14.922
1600	18.266
1700	21.828
1800	25.828
1900	31.343
2000	35.375
2100	40.875
2200	48.172
2300	54.25
2400	62.25
2500	69.547

Apreciamos en estos resusltados que, efectivamente, en la última iteración el tiempo de ejecución es notablemente mayor que en cualquier otro caso estudiado anteriormente, tardando más de un minuto para un tamaño de entrada de datos de 2500.

Representación de datos

Al representar en una gráfica los datos de la tabla anterior, obtenemos:



Como podemos ver, la trayectoria de la traza formada por los datos entrada en el algorimtmo se asemeja bastante a la de una función cúbica, además de presentar muy pocas imperfecciones.

2.4 $O(2^n)$ Hanoi

El último orden de eficiencia que estudiaremos en este apartado es $O(2^n)$. Para este orden, analizaremos el algoritmo de las torres de Hanoi.
El funcionamiento de este algoritmo consiste en mover todos los discos de una varilla (considerando que hay únicamente 3 varillas) a otra, estando estos ordenados de forma que el disco de tamaño mas grande esté situado en la base y los discos estén ordenados.

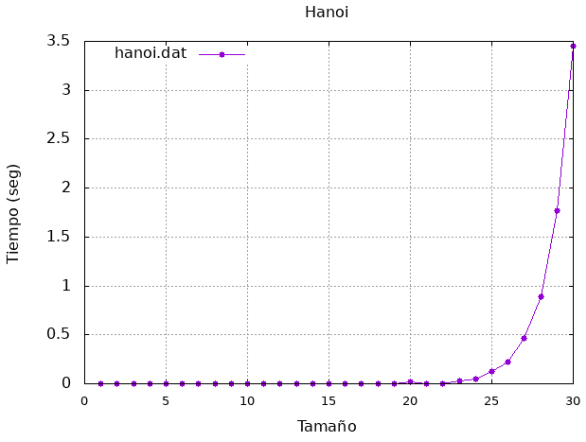
Tabla de datos

Se trata de un algoritmo exponencial, por tanto, podemos asumir que sus tiempos irán creciendo de forma gigantesca a medida que los tamaños vayan aumentando. Debido a esto, el rango de tamaño de datos irá de 1-30, aumentando únicamente un único valor durante cada iteración, obteniendo por tanto 30 datos.

Tamaño	Tiempo (seg)	Tamaño	Tiempo (seg)
1	0	16	0
2	0	17	0
3	0	18	0
4	0	19	0
5	0	20	0.015
6	0	21	0
7	0	22	0
8	0	23	0.031
9	0	24	0.046
10	0	25	0.125
11	0	26	0.218
12	0	27	0.468
13	0	28	0.89
14	0	29	1.765
15	0	30	3.453

Representación de datos

Al representar los datos en una gráfica:



Podemos ver que el tiempo es lineal con apenas diferencia de tiempo durante más de la mitad de las iteraciones. Sin embargo, a medida que van aumentando las mismas, el tiempo irá incrementando de forma exponencial hasta dispararse en la última iteración.

3 Eficiencia Híbrida

Para estudiar la eficiencia híbrida de cada caso, se han definido determinadas funciones de acuerdo al orden de eficiencia al que pertenece cada uno de los algortimos. Los ajustes entre estas funciones y los resultados obtenidos en la ejecución de cada algoritmo se realizarán con el programa de interfaz de línea de comandos `gnuplot`. Más específicamente se utilizarán los siguientes comandos: Para presentar los datos de una tabla en una ventana emergente:

```
plot 'datosObtenidos.dat' w lp
```

Para definir una función

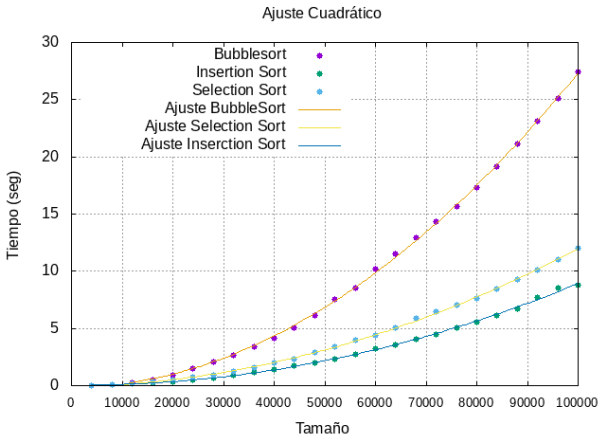
```
f(x)= a+b+c
```

Para realizar un ajuste con una función f(x) determinada

```
fit f(x) 'datosObtenidos.dat' via a,b,c
```

3.1 O(n^2)

Dada la forma que describieron los algoritmos de búsqueda Bubblesort, Insertion Sort y Selection Sort, lo natural es primero intentar ajustar una ecuación cuadrática a los datos.



$f(x) = ax^2 + bx + c$

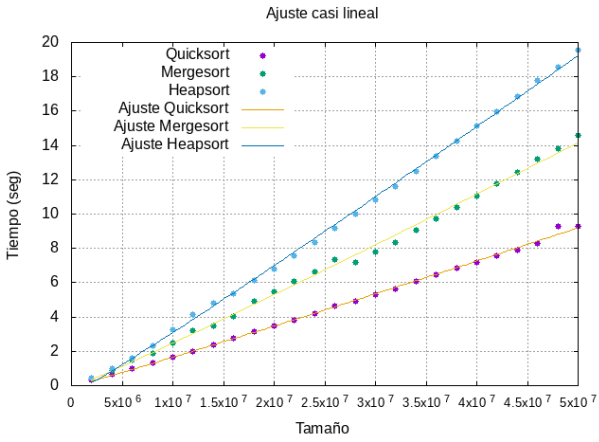
	Bubblesort		Insertion Sort		Selection Sort	
	Valor	Error	Valor	Error	Valor	Error
a	2.68917×10^{-9}	1.64%	9.34903×10^{-10}	2.825%	1.12415×10^{-9}	1.993%
b	6.45923×10^{-6}	73.15%	-4.93498×10^{-6}	57.34%	8.31886×10^{-6}	28.85%
c	-0.208944	51.04%	0.0713999	89.44%	-0.102149	53.02%

Como se puede apreciar, utilizando este ajuste se logran excelentes resultados, con errores bastante bajos para x^2, que es el término que nos interesa; esto nos demuestra que el cálculo de la eficiencia teórica y las intuiciones al ver las gráficas estaban en lo correcto: todos estos algoritmos son efectivamente cuadráticos, aunque claro, las constantes ocultas juegan un papel importante en la práctica.

Bubblesort resalta por ser el que más tarda, esto se ve reflejado en su constante oculta, que es la mayor del grupo; se debe principalmente a que, aparte de realizar múltiples recorridos a la lista de elementos, también realiza muchos intercambios de elementos, los cuales se traducen en mayores tiempos a nivel práctico, seguido viene Selection Sort, que realiza una cantidad menor de intercambios, y esto se refleja en su mejor eficiencia; finalmente Insertion Sort es es más eficiente del grupo, este realiza la mínima cantidad de intercambios, por lo tanto es el menos se ve afectado por el hardware de la máquina.

3.2 $\mathbf{O}(n\log(n))$

A continuación se presentan los restastes algoritmos de ordenación, estos describen casi una línea recta, ya que está ligeramente curvada y por lo tanto, es natural ajustar una función semilineal.



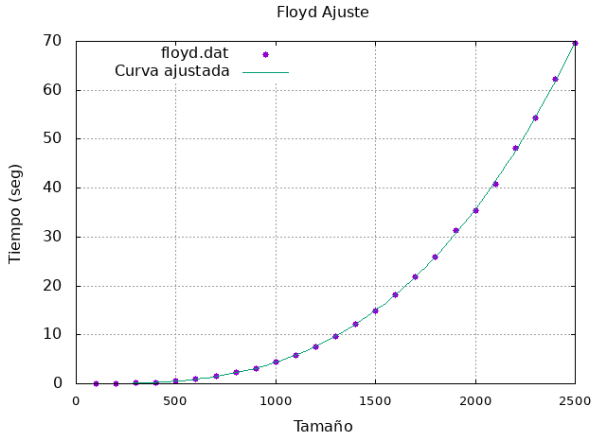
$$f(x) = ax\log(x) + b$$

	Quicksort		Mergesort		Heapsort	
	Valor	Error	Valor	Error	Valor	Error
a	1.04129×10^{-8}	0.8526%	1.61069×10^{-8}	1.203%	2.2266×10^{-8}	4.437%
b	-0.0259718	177%	-0.114632	87.52%	-0.491546	14.17%

Los datos demuestran que un ajuste casi lineal es excelente para estos algoritmos, las curvas que describen son extremadamente similares, pero como quedó plasmado en el apartado anterior, las variables ocultas juegan un rol muy importante a la hora de la práctica: Quicksort que posee un peor caso de $\mathrm{O}(n^2)$, paradójicamente es el más eficiente de todos, esto se debe en parte a que el peor caso de Quicksort ocurre cuando el pivote elegido es uno de los valores extremos (el mayor o menor de la lista), esto se mitiga implementando la búsqueda del valor mediano, que es posible en tiempo lineal, por otra parte, Quicksort no utiliza estructuras auxiliares para ordenar, a diferencia de Mergesort que necesita de una lista auxiliar y Heapsort hace uso del *heap*, estos requerimientos de espacio adicionales tienen un costo de ejecución que se traduce en que las constantes ocultas son mayores y por lo tanto, tardan más.

3.3 $\mathbf{O}(n^3)$

En vista de los datos, se ve una curva más pronunciada que la de los algoritmos cuadráticos, por lo tanto se realiza un ajuste cúbico.

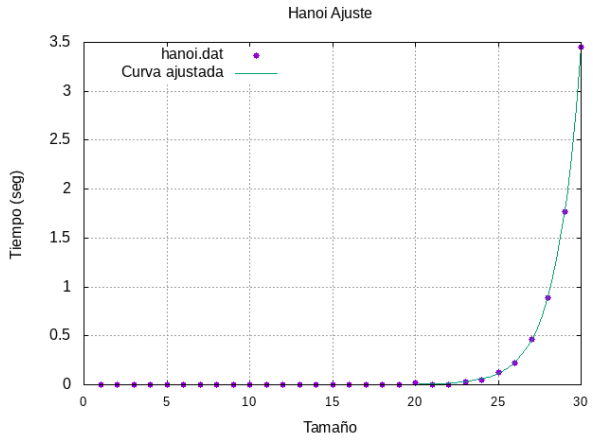


$f(x) = ax^3 + bx^2 + cx + d$		
	Floyd	
	Valor	Error
a	4.45958×10^{-9}	4.273%
b	1.29115×10^{-7}	583.1%
c	-0.000261991	324.9%
d	0.0653099	399.1%

Efectivamente, el ajuste que se obtiene es bastante bueno, el orden de Floyd es cúbico, el crecimiento que posee rebasa por mucho a Bubblesort que es un orden menor. Al tener en cuenta que el algoritmo de Floyd hace uso de tres bucles anidados para realizar la búsqueda de los caminos mínimos en un grafo, por lo que este resultado era de esperarse.

3.4 $\mathrm{O}(2^n)$

Por último, queda el ajuste del algoritmo de las torres de Hanoi, en vista de la curva que se genera, no queda más alternativa que probar un ajuste exponencial.



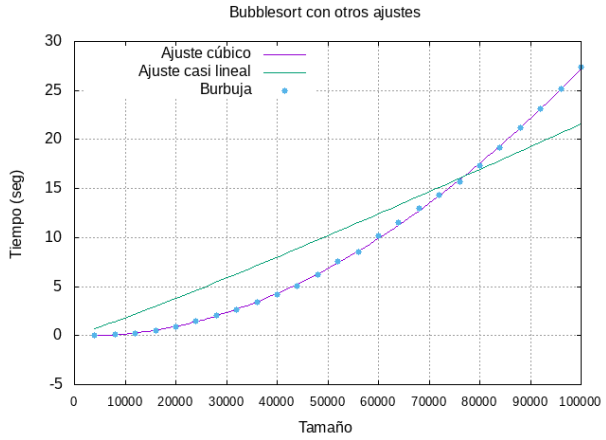
$f(x) = a2^{bx}$		
	Hanoi	
	Valor	Error
a	5.5483×10^{-9}	6.659%
b	0.9738	0.3293%

En efecto, el ajuste es muy bueno, esto deja sin lugar a dudas que efectivamente el algoritmo es exponencial por naturaleza, esto es debido a la manera en que funciona, para mover un disco n tal que $n \geq 2$, se necesita primero mover los otros $n-1$ discos de encima a la columna auxiliar, mover el disco actual a la columna final y luego apilar de nuevo los $n-1$ discos encima del disco n, la solución a esta recurrencia $T(n) = 2T(n-1) + 1$ es una ecuación exponencial.

3.5 Otros ajustes

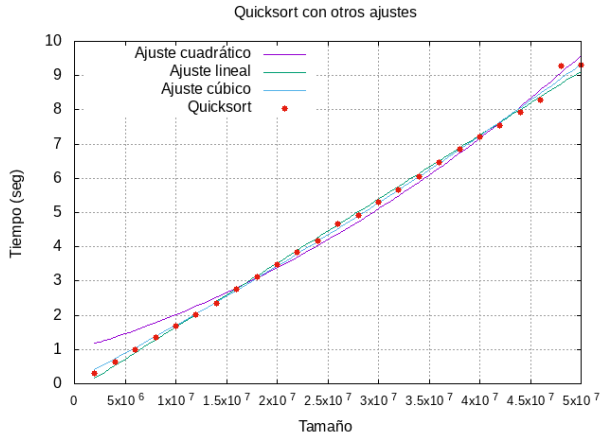
En este apartado se realizaron ajustes distintos a los que corresponderían a los algoritmos, se ha elegido un algoritmo de orden $\mathrm{O}(n^2)$ y $\mathrm{O}(n\log(n))$ para que represente ese orden, de modo que ilustren que lo sucedería en general para ese grupo.

3.5.1 $\mathbf{O}(n^2)$



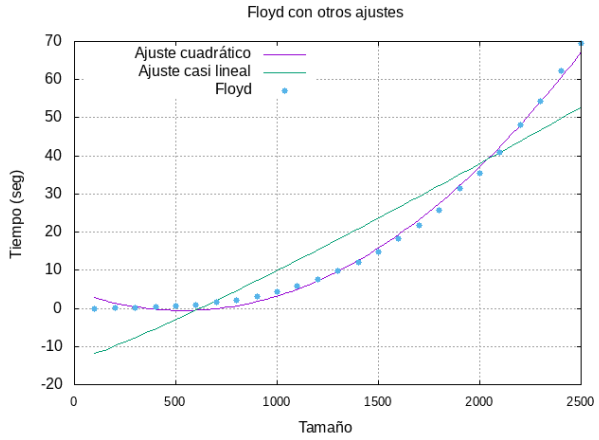
Para este orden tomamos al más conocido del grupo, el algoritmo Bubblesort. Se puede observar que no es posible para un ajuste semi lineal adaptarse a los datos, pero esto no es verdad para el ajuste cúbico, que se ajusta bastante bien a los datos, siendo casi indistinguible del ajuste cuadrático.

3.5.2 $\mathbf{O}(n\log(n))$



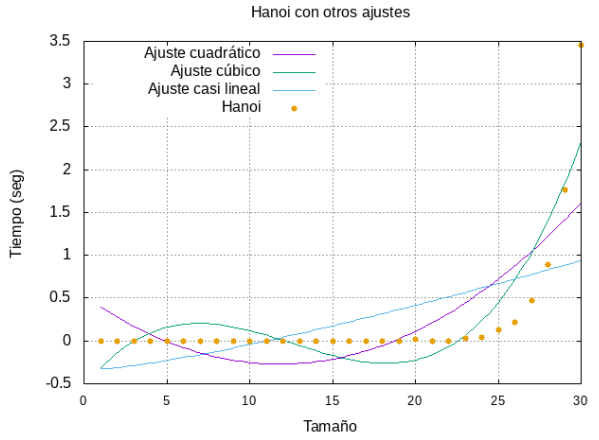
Para los algoritmos casi lineales, se toma Quicksort. Como se nota, los ajustes cuadrático y cúbico se asemejan a los datos pero por su naturaleza polinómica, se empiezan a curvar a los extremos de los datos, por otro lado, un ajuste lineal queda muy bien, por lo que se puede notar porque a estos algoritmos se les llama de tal forma, casi lineales.

3.5.3 $\mathbf{O}(n^3)$



Para el único ejemplar de este orden, podemos ver que, naturalmente no hay forma que un ajuste semi lineal logre tomar la forma de uno cúbico, aunque vale la pena notar que un ajuste cuadrático se asimila relativamente bien, pero siendo un grado menor no puede crecer lo suficientemente rápido.

3.5.4 $\mathbf{O}(2^x)$



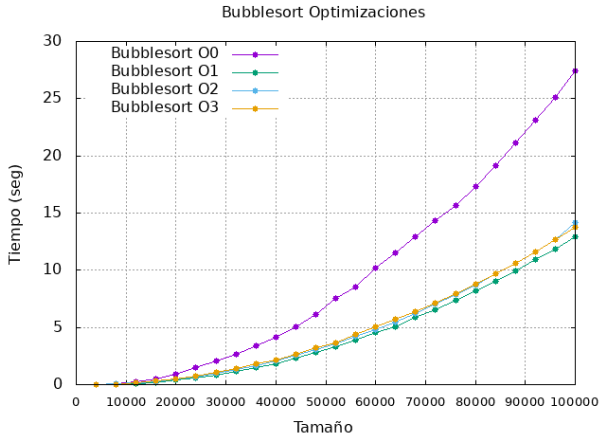
Hanoi prueba que no hay nada que se pueda comparar con un crecimiento tan extremo como lo es el exponencial, se puede ver como las funciones polinómicas, si bien por un momento logran ligeramente asimilarse, no pueden crecer lo suficientemente rápido como para ajustarse a los datos.

4 Comparación con Diferentes Optimizaciones de Compilación

En este apartado vamos a mostrar como puede afectar al tiempo de ejecución la optimización usada al compilar el algoritmo de ordenación. En todos los casos se ha usado el compilador g++ con todos los parámetros por defecto excepto su optimización (-O0, -O1, -O2, -O3).

4.1 $\mathbf{O}(n^2)$

4.1.1 Bubblesort

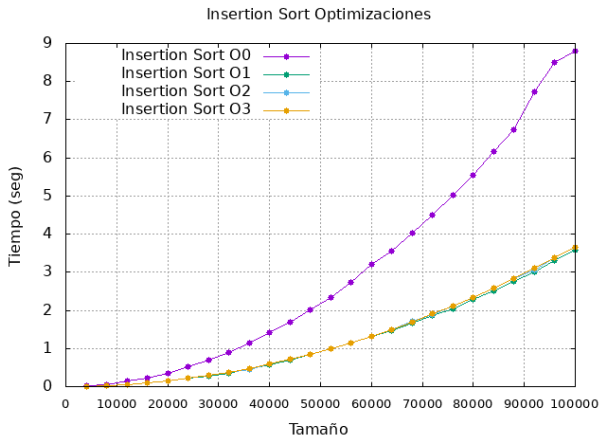


Podemos observar gráficamente como el tiempo de ejecución del algoritmo Bubblesort es bastante peor cuando lo compilamos por defecto. El resto de optimizaciones reducen el tiempo casi el doble que cuando lo hacemos por defecto; y es también levemente visible que -O1 es el parámetro de optimización más óptimo.

Tabla de tiempos Bubblesort

Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
4000	0.02560	0.00684	0.01248	0.00812
8000	0.10372	0.02752	0.06812	0.03812
12000	0.265	0.093	0.140	0.187
16000	0.515	0.187	0.281	0.328
20000	0.921	0.375	0.468	0.531
24000	1.453	0.578	0.687	0.781
28000	2.046	0.828	0.984	1.062
32000	2.671	1.125	1.296	1.421
36000	3.421	1.500	1.687	1.812
40000	4.156	1.860	2.093	2.156
44000	5.078	2.312	2.546	2.671
48000	6.171	2.781	3.046	3.218
52000	7.578	3.296	3.593	3.687
56000	8.531	3.875	4.203	4.406
60000	10.202	4.562	4.828	5.078
64000	11.546	5.093	5.500	5.734
68000	12.937	5.859	6.250	6.375
72000	14.312	6.546	7.031	7.156
76000	15.656	7.343	7.843	7.921
80000	17.343	8.171	8.703	8.765
84000	19.171	9.031	9.656	9.703
88000	21.171	9.906	10.640	10.625
92000	23.093	10.906	11.593	11.594
96000	25.140	11.875	12.687	12.671
100000	27.390	12.937	14.140	13.734

4.1.2 Insertion Sort

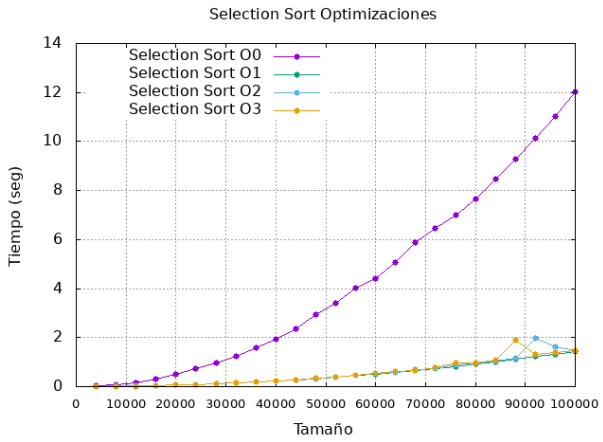


Podemos destacar de nuevo como cuando no usamos optimización el algoritmo Insertion Sort es peor que usando cualquier parámetro de optimización mayor o igual a -O1. En este caso no podemos ver con claridad cual de las optimizaciones es mejor ya que están bastante igualados los tiempos de ejecución entre ellas, pero es destacable como todas ellas mejoran más del doble el tiempo de ejecución.

Tabla de tiempos Insertion Sort

Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
4000	0.01372	0.00560	0.00624	0.00624
8000	0.05624	0.02248	0.02312	0.02312
12000	0.140	0.062	0.062	0.062
16000	0.235	0.093	0.094	0.093
20000	0.359	0.156	0.156	0.156
24000	0.515	0.218	0.218	0.218
28000	0.687	0.281	0.296	0.296
32000	0.906	0.360	0.375	0.375
36000	1.140	0.484	0.453	0.468
40000	1.406	0.578	0.593	0.609
44000	1.703	0.688	0.718	0.719
48000	2.015	0.843	0.843	0.843
52000	2.344	0.984	1.000	1.000
56000	2.734	1.140	1.156	1.140
60000	3.203	1.312	1.312	1.312
64000	3.546	1.468	1.500	1.500
68000	4.031	1.671	1.718	1.703
72000	4.500	1.859	1.890	1.921
76000	5.015	2.047	2.125	2.125
80000	5.546	2.281	2.343	2.343
84000	6.156	2.515	2.578	2.578
88000	6.734	2.750	2.843	2.828
92000	7.734	3.016	3.062	3.109
96000	8.500	3.296	3.375	3.375
100000	8.796	3.578	3.656	3.656

4.1.3 Selection Sort

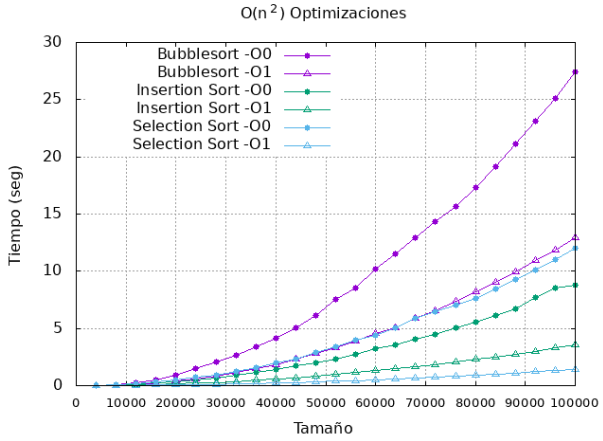


Es destacable que cuando ejecutamos Selection Sort sin optimización la curva cuadrática es bastante apreciable; sin embargo, cuando lo hacemos con optimización casi podríamos decir que el tiempo es lineal y que este se reduce muchísimo a su vez. En este caso tampoco podríamos decir con seguridad cuál de las tres opciones de optimización es más óptima.

Tabla de tiempos Selection Sort

Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
4000	0.01936	0.00248	0.00312	0.00312
8000	0.07812	0.01000	0.02372	0.01000
12000	0.172	0.031	0.031	0.015
16000	0.312	0.046	0.046	0.031
20000	0.500	0.062	0.062	0.062
24000	0.718	0.093	0.093	0.093
28000	0.953	0.110	0.125	0.109
32000	1.250	0.156	0.156	0.156
36000	1.578	0.187	0.203	0.187
40000	1.953	0.234	0.250	0.235
44000	2.359	0.281	0.281	0.265
48000	2.937	0.343	0.328	0.343
52000	3.390	0.406	0.406	0.391
56000	4.015	0.453	0.453	0.453
60000	4.390	0.515	0.531	0.546
64000	5.062	0.593	0.593	0.609
68000	5.890	0.671	0.687	0.671
72000	6.468	0.734	0.750	0.781
76000	7.015	0.828	0.875	0.953
80000	7.656	0.921	0.984	0.968
84000	8.453	1.015	1.031	1.078
88000	9.281	1.109	1.141	1.890
92000	10.140	1.234	1.983	1.312
96000	11.031	1.312	1.609	1.390
100000	12.015	1.437	1.484	1.468

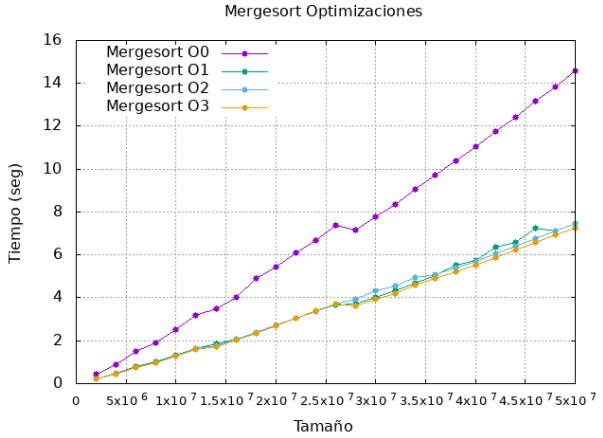
4.1.4 Comparación entre algoritmos O(n^2)



Tanto Bubblesort y su optimización siguen siendo los algoritmos más lentos; lo que si cabe destacar es que Selection Sort sin optimizar es más lento que Insertion Sort y su optimización, sin embargo la optimización de Selection Sort es más rápida que cualquiera de las opciones anteriores.

4.2 O(nlogn)

4.2.1 Mergesort

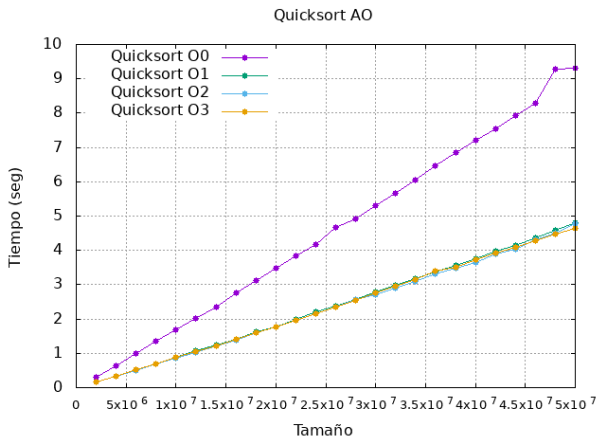


Podemos apreciar en la gráfica cómo el tiempo es casi lineal para el algoritmo Mergesort. En tamaños menores a $3 * 10^7$ ninguna de las optimizaciones parece destacar (todas aportan una mejora de casi el doble en velocidad), sin embargo a partir de dicho punto si que podemos apreciar una leve mejora con la optimización -O3.

Tabla de tiempos Mergesort

Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
2000000	0.437	0.233	0.218	0.218
4000000	0.906	0.484	0.468	0.453
6000000	1.499	0.811	0.765	0.733
8000000	1.906	1.031	0.985	0.952
10000000	2.499	1.313	1.282	1.266
12000000	3.204	1.641	1.625	1.578
14000000	3.485	1.845	1.781	1.719
16000000	4.031	2.078	2.063	2.017
18000000	4.906	2.390	2.406	2.359
20000000	5.454	2.704	2.719	2.687
22000000	6.078	3.031	3.030	3.063
24000000	6.656	3.390	3.375	3.391
26000000	7.375	3.687	3.719	3.734
28000000	7.157	3.720	3.938	3.640
30000000	7.765	4.015	4.312	3.922
32000000	8.344	4.329	4.532	4.218
34000000	9.079	4.688	4.953	4.594
36000000	9.719	5.031	5.094	4.922
38000000	10.406	5.516	5.390	5.203
40000000	11.047	5.734	5.703	5.531
42000000	11.749	6.359	6.062	5.875
44000000	12.406	6.593	6.406	6.235
46000000	13.187	7.249	6.750	6.578
48000000	13.827	7.094	7.109	6.954
50000000	14.594	7.453	7.484	7.250

4.2.2 Quicksort

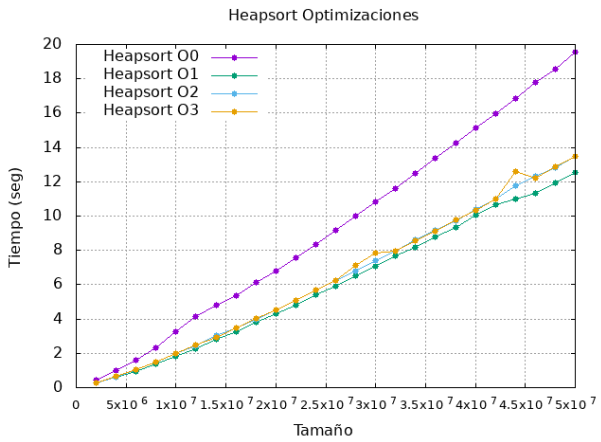


Seguimos obteniendo una gráfica casi completamente lineal. En este caso no podemos decir con seguridad que ninguna optimización sea mejor pues todas están muy superpuestas; pero si que se puede observar que la mejora sigue siendo de más o menos el doble.

Tabla de tiempos Quicksort

Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
2000000	0.313	0.156	0.156	0.156
4000000	0.641	0.328	0.328	0.328
6000000	0.985	0.531	0.500	0.531
8000000	1.344	0.687	0.688	0.688
10000000	1.672	0.890	0.860	0.875
12000000	2.015	1.078	1.015	1.047
14000000	2.359	1.235	1.219	1.204
16000000	2.750	1.422	1.391	1.422
18000000	3.125	1.625	1.610	1.594
20000000	3.468	1.781	1.781	1.781
22000000	3.829	2.000	1.953	1.969
24000000	4.172	2.219	2.141	2.156
26000000	4.656	2.375	2.343	2.359
28000000	4.907	2.578	2.562	2.532
30000000	5.297	2.797	2.703	2.750
32000000	5.656	2.985	2.906	2.953
34000000	6.062	3.188	3.093	3.141
36000000	6.453	3.375	3.313	3.406
38000000	6.844	3.562	3.468	3.500
40000000	7.203	3.766	3.657	3.734
42000000	7.547	3.969	3.906	3.921
44000000	7.922	4.141	4.031	4.078
46000000	8.296	4.375	4.312	4.281
48000000	9.281	4.578	4.516	4.469
50000000	9.297	4.797	4.781	4.640

4.2.3 Heapsort

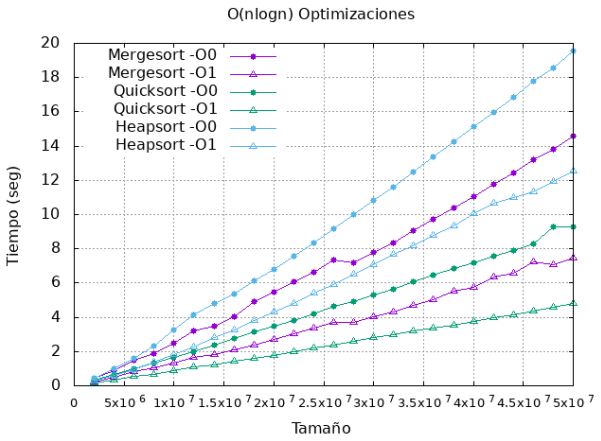


Nos encontramos de nuevo con una gráfica lineal. La mejor optimización es -O1, llegando a diferenciarse en más de un segundo con las demás optimizaciones. En ningún caso se alcanza una mejora tan grande como en el resto de algoritmos de O(nlogn).

Tabla de tiempos Heapsort

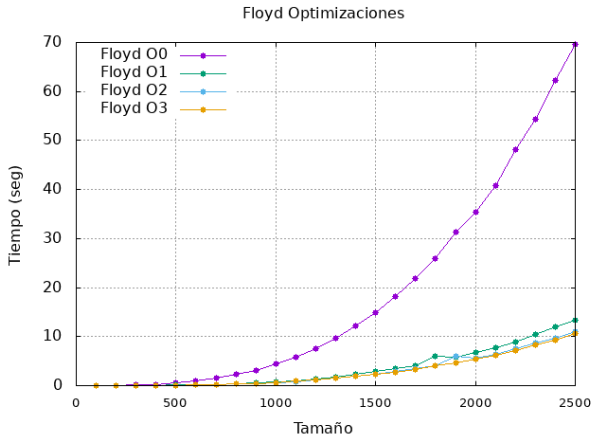
Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
2000000	0.421	0.265	0.265	0.265
4000000	1.000	0.593	0.625	0.640
6000000	1.609	0.953	1.047	1.062
8000000	2.328	1.391	1.484	1.500
10000000	3.235	1.844	1.969	1.985
12000000	4.125	2.282	2.454	2.469
14000000	4.813	2.828	3.016	2.954
16000000	5.359	3.250	3.484	3.484
18000000	6.109	3.829	4.000	4.016
20000000	6.797	4.312	4.547	4.547
22000000	7.594	4.828	5.109	5.093
24000000	8.360	5.422	5.672	5.672
26000000	9.188	5.937	6.250	6.234
28000000	10.016	6.515	6.797	7.125
30000000	10.828	7.063	7.422	7.828
32000000	11.625	7.688	7.953	7.969
34000000	12.500	8.187	8.641	8.578
36000000	13.344	8.797	9.156	9.141
38000000	14.250	9.312	9.734	9.765
40000000	15.125	10.032	10.390	10.343
42000000	15.969	10.672	11.016	10.968
44000000	16.859	11.016	11.750	12.594
46000000	17.766	11.313	12.312	12.219
48000000	18.578	11.922	12.844	12.860
50000000	19.547	12.515	13.469	13.500

4.2.4 Comparación entre algoritmos O(nlogn)



En esta gráfica podemos observar como Quicksort es el algoritmo más rápido de los 3, y su optimización sigue haciendo a este algoritmo ganador. La optimización de Mergesort supera en velocidad al algoritmo Quicksort y a la optimización de Heapsort; sin embargo Heapsort solo logra superar a Mergesort con su optimización.

4.3 O(n^3) Floyd

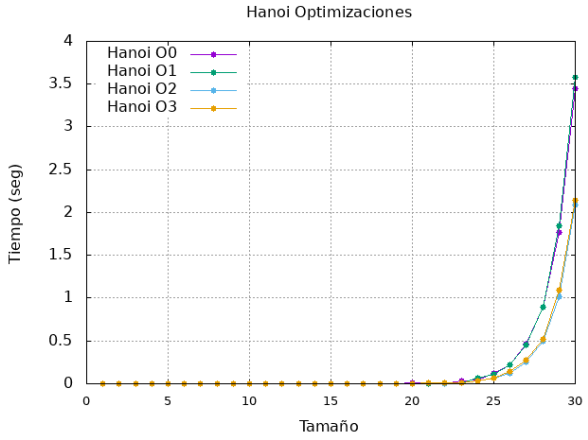


El algoritmo de Floyd es de orden de eficiencia O(n^3), y en su gráfica podemos observar que evidentemente este es cúbica. Ninguna optimización destaca por su gran mejora pero si que destaca que -O1 es la peor de las optimizaciones; aun así la mejora en tiempo es muy grande en cualquiera de ellas, llegan en el mayor tamaño a no superar los 14 segundos en contra del tiempo sin optimización que es de unos 70 segundos.

Tabla de tiempos Floyd

Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
100	0.000	0.000	0.000	0.000
200	0.031	0.015	0.000	0.015
300	0.125	0.031	0.015	0.015
400	0.281	0.062	0.046	0.046
500	0.562	0.109	0.078	0.078
600	0.968	0.171	0.156	0.140
700	1.515	0.265	0.218	0.218
800	2.250	0.391	0.328	0.328
900	3.187	0.578	0.468	0.453
1000	4.360	0.812	0.640	0.625
1100	5.875	1.046	0.859	0.875
1200	7.610	1.391	1.140	1.125
1300	9.719	1.812	1.531	1.453
1400	12.156	2.281	1.891	1.843
1500	14.922	2.813	2.375	2.266
1600	18.266	3.406	2.828	2.796
1700	21.828	4.109	3.531	3.343
1800	25.828	5.938	4.063	3.984
1900	31.343	5.766	6.015	4.687
2000	35.375	6.719	5.688	5.453
2100	40.875	7.735	6.406	6.281
2200	48.172	8.922	7.453	7.234
2300	54.250	10.531	8.625	8.250
2400	62.250	11.953	9.688	9.375
2500	69.547	13.313	10.953	10.594

4.4 $O(2^n)$ Hanoi



Con el algoritmo Hanoi es curioso observar como tanto sin optimización como con el parámetro de optimización -O1 los tiempos tomados son prácticamente iguales llegando a solaparse sus tiempos en la gráfica. El resto de las optimizaciones (-O2 y -O3) son igual de buenas; obteniendo también tiempos muy similares. Estas últimas optimizaciones producen mejoras más notables del tiempo en tanto que el tamaño del problema aumenta.

Tabla de tiempos Hanoi

Tamaño	-O0 (seg)	-O1 (seg)	-O2 (seg)	-O3 (seg)
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0
16	0	0	0	0
17	0	0	0	0
18	0	0	0	0
19	0	0	0	0
20	0.015	0.000	0.000	0.000
21	0.000	0.000	0.015	0.015
22	0.000	0.015	0.000	0.015
23	0.031	0.016	0.015	0.015
24	0.046	0.062	0.031	0.031
25	0.125	0.109	0.062	0.063
26	0.218	0.218	0.125	0.140
27	0.468	0.453	0.250	0.281
28	0.890	0.890	0.500	0.515
29	1.765	1.843	1.015	1.093
30	3.453	3.578	2.093	2.141

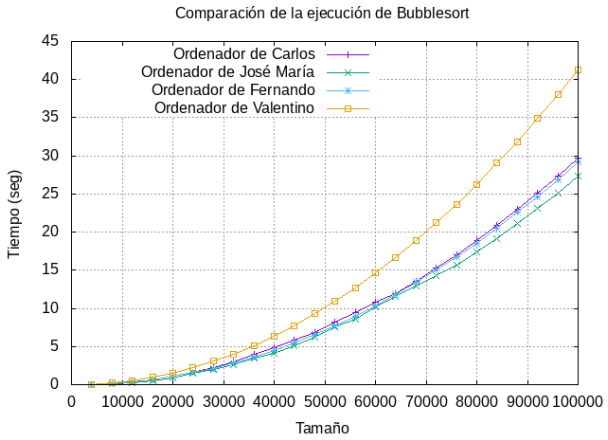
5 Comparación de tiempos de ejecución de los algoritmos en diferentes ordenadores

En este apartado vamos a comparar las gráficas resultantes de recoger los tiempos de ejecución de los algoritmos en los diferentes ordenadores de los integrantes de este grupo de trabajo. Las características de los ordenadores utilizados son las siguientes:

- El ordenador de José María es un HP Pavilion con procesador Intel Core i5-1035G1 con 8 GB de RAM.
- El ordenador de Fernando es un Asus x550jf con procesador Intel Core i5-4200h con 8 GB de RAM.
- El ordenador de Valentino es un VIT P2412 con procesador Intel Core i3-4000M con 16 GB de RAM.
- El ordenador de Carlos es un HP Elitebook 840 g5 con procesador Intel Core i5-8250U con 8 GB de RAM.

5.1 $O(n^2)$

5.1.1 Bubblesort

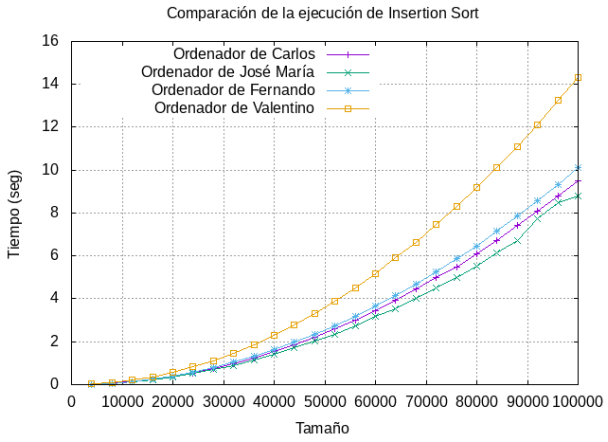


Podemos apreciar que el ordenador de Valentino realiza las ejecuciones en un tiempo notablemente mayor que el resto conforme aumentamos el tamaño, lo cual se debe a que presenta un procesador con una potencia menor a los demás. Vemos también que los ordenadores de Fernando y Carlos realizan las ejecuciones en unos tiempos similares, mientras que el ordenador de José María ejecuta el algoritmo en unos tiempos menores a los dos anteriores, ya su ordenador presenta el procesador más moderno de los que se están comparando.

Tabla de tiempos de ejecución de Bubblesort

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
4000	0.038225	0.0256	0.032133	0.0454007
8000	0.167734	0.10372	0.150913	0.213439
12000	0.375468	0.265	0.364485	0.517417
16000	0.676016	0.515	0.673819	0.959662
20000	1.08752	0.921	1.07899	1.51935
24000	1.6576	1.453	1.57197	2.21796
28000	2.29772	2.046	2.17109	3.06455
32000	3.01229	2.671	2.85155	4.03014
36000	3.96004	3.421	3.62939	5.14678
40000	4.81668	4.156	4.53165	6.38172
44000	5.78344	5.078	5.48191	7.76172
48000	6.82366	6.171	6.56645	9.28973
52000	8.22331	7.578	7.74341	10.9457
56000	9.45819	8.531	8.99433	12.7077
60000	10.756	10.202	10.3486	14.6557
64000	11.9873	11.546	11.7966	16.6904
68000	13.6009	12.937	13.4722	18.8736
72000	15.2361	14.312	15.0062	21.2116
76000	16.9817	15.656	16.7689	23.6651
80000	18.9518	17.343	18.5697	26.2697
84000	20.9104	19.171	20.5226	29.0771
88000	22.9716	21.171	22.5686	31.8298
92000	25.0846	23.093	24.6605	34.9241
96000	27.3304	25.14	26.8412	38.0679
100000	29.7309	27.39	29.2117	41.2817

5.1.2 Insertion Sort

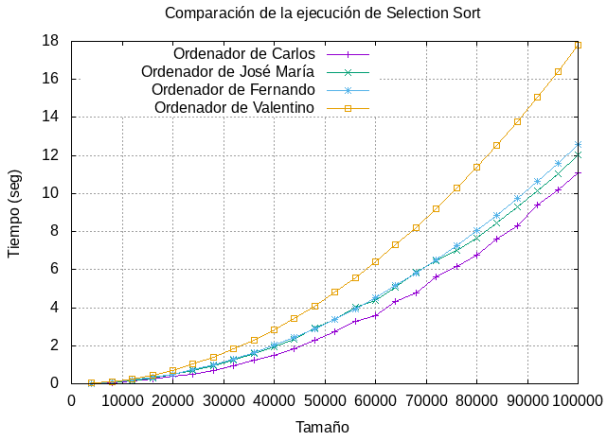


Vemos, como en la comparativa de Bubblesort, que el ordenador de Valentino nos da tiempos notablemente mayores conforme aumentamos el tamaño de la muestra. Sin embargo los resultados de los otros 3 ordenadores no se muestran tan similares unos de otros como en Bubblesort, aunque el ordenador de José María siga dando los mejores resultados.

Tabla de tiempos de ejecución de Insertion Sort

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
4000	0.020534	0.01372	0.0162783	0.0234958
8000	0.067109	0.05624	0.0649249	0.0918305
12000	0.146492	0.14	0.148658	0.204869
16000	0.248336	0.235	0.262948	0.37066
20000	0.382913	0.359	0.409053	0.57452
24000	0.552187	0.515	0.587391	0.829675
28000	0.756851	0.687	0.796026	1.12425
32000	0.988853	0.906	1.03937	1.47221
36000	1.23948	1.14	1.31277	1.84968
40000	1.52779	1.406	1.62903	2.28597
44000	1.86334	1.703	1.97142	2.79301
48000	2.2055	2.015	2.33632	3.31055
52000	2.58991	2.344	2.73456	3.8922
56000	3.02327	2.734	3.19385	4.50278
60000	3.4402	3.203	3.67869	5.15972
64000	3.93555	3.546	4.16568	5.90455
68000	4.45192	4.031	4.69136	6.63618
72000	4.98166	4.5	5.26035	7.46763
76000	5.49843	5.015	5.86523	8.30853
80000	6.10323	5.546	6.46994	9.18898
84000	6.73097	6.156	7.17104	10.1146
88000	7.41877	6.734	7.87254	11.0948
92000	8.07097	7.734	8.56408	12.1065
96000	8.78151	8.5	9.34269	13.2593
100000	9.51208	8.796	10.1154	14.3351

5.1.3 Selection Sort



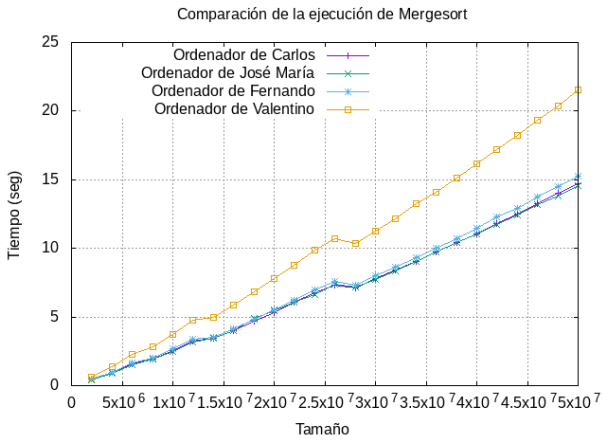
En el caso de Selection Sort, sigue habiendo una notable diferencia entre el procesador de Valentino y los demás. Sin embargo por debajo, vemos que los resultados obtenidos en el ordenador de José María y el de Fernando son similares y en este caso, en el ordenador de Carlos se han obtenido unos tiempos algo menores al resto.

Tabla de tiempos de ejecución de Selection Sort

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
4000	0.018001	0.01936	0.0204037	0.0288958
8000	0.063358	0.07812	0.0809568	0.114459
12000	0.133498	0.172	0.184992	0.258962
16000	0.245271	0.312	0.324831	0.458811
20000	0.38715	0.5	0.506158	0.714326
24000	0.521397	0.718	0.727546	1.02887
28000	0.707219	0.953	0.990691	1.39848
32000	0.921253	1.25	1.29303	1.82677
36000	1.22853	1.578	1.63435	2.30912
40000	1.48297	1.953	2.0155	2.85118
44000	1.85903	2.359	2.43951	3.44659
48000	2.27038	2.937	2.90173	4.10165
52000	2.71941	3.39	3.40428	4.81327
56000	3.27737	4.015	3.94957	5.58099
60000	3.59638	4.39	4.534	6.40833
64000	4.33285	5.062	5.15626	7.28879
68000	4.78542	5.89	5.82046	8.22901
72000	5.6436	6.468	6.5274	9.22217
76000	6.18822	7.015	7.26786	10.2779
80000	6.75925	7.656	8.05662	11.3837
84000	7.61136	8.453	8.86594	12.5507
88000	8.30335	9.281	9.74483	13.776
92000	9.38668	10.14	10.6468	15.0509
96000	10.1968	11.031	11.5952	16.3964
100000	11.1043	12.015	12.5801	17.7789

5.2 O(nlogn)

5.2.1 Mergesort

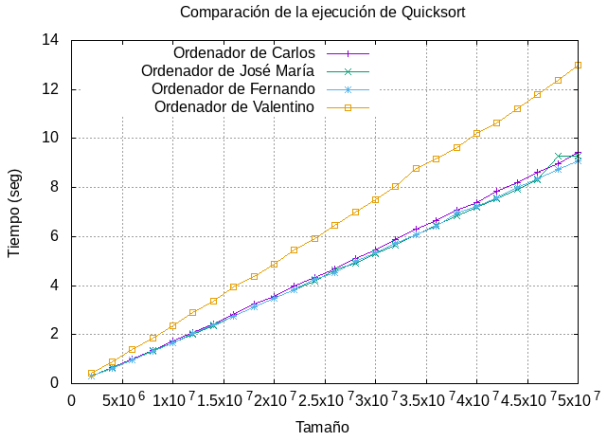


En este algoritmo apreciamos de nuevo una diferencia notable entre los resultados obtenidos en el ordenador de Valentino y los demás. Destaca en este caso que los otros 3 ordenadores obtienen resultados bastante similares durante la mayoría de los tamaños puestos, notándose un poco más la diferencia en los últimos tamaños.

Tabla de tiempos de ejecución de Mergesort

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
2000000	0.454686	0.437	0.462336	0.647641
4000000	0.935899	0.906	0.958385	1.35462
6000000	1.56642	1.499	1.62835	2.29038
8000000	1.94474	1.906	2.00063	2.81338
10000000	2.57176	2.499	2.66015	3.75801
12000000	3.25618	3.204	3.3799	4.74964
14000000	3.43472	3.485	3.52983	4.97347
16000000	4.03715	4.031	4.15953	5.89426
18000000	4.66377	4.906	4.82326	6.8064
20000000	5.3305	5.454	5.51577	7.79412
22000000	6.05301	6.078	6.22941	8.78675
24000000	6.74637	6.656	6.98554	9.84894
26000000	7.31984	7.375	7.58081	10.6734
28000000	7.13995	7.157	7.33639	10.3732
30000000	7.77388	7.765	7.99849	11.2427
32000000	8.39284	8.344	8.63294	12.1631
34000000	9.06533	9.079	9.33372	13.2603
36000000	9.72781	9.719	10.0294	14.1151
38000000	10.4319	10.406	10.7332	15.127
40000000	11.0794	11.047	11.468	16.1615
42000000	11.7877	11.749	12.2883	17.1618
44000000	12.5019	12.406	12.9332	18.2275
46000000	13.2501	13.187	13.7625	19.3132
48000000	13.9923	13.827	14.4877	20.402
50000000	14.7348	14.594	15.2733	21.5537

5.2.2 Quicksort

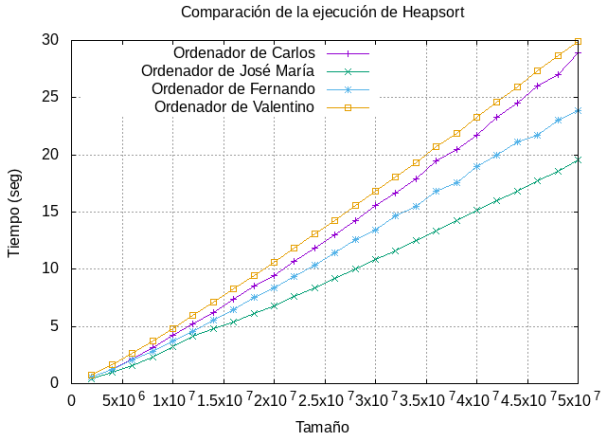


Se puede apreciar que los datos obtenidos por todos los ordenadores (Salvo el de Valentino por razones expuestas anteriormente), son bastante similares entre sí, aunque en este caso, en los valores del final la diferencia de tiempos es menos visible en la gráfica de lo que lo era en Mergesort.

Tabla de tiempos de ejecución de Quicksort

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
2000000	0.314941	0.313	0.302476	0.429395
4000000	0.655127	0.641	0.637463	0.893218
6000000	1.00526	0.985	0.974953	1.37943
8000000	1.35849	1.344	1.32871	1.86091
10000000	1.72534	1.672	1.66224	2.35514
12000000	2.0816	2.015	2.05545	2.89206
14000000	2.4518	2.359	2.3854	3.37493
16000000	2.83347	2.75	2.75052	3.93911
18000000	3.23422	3.125	3.12498	4.37557
20000000	3.5433	3.468	3.47784	4.88243
22000000	3.98544	3.829	3.82975	5.46657
24000000	4.34709	4.172	4.23815	5.92379
26000000	4.67188	4.656	4.54067	6.45146
28000000	5.12007	4.907	4.9863	7.0018
30000000	5.44894	5.297	5.34872	7.50709
32000000	5.89692	5.656	5.72577	8.0257
34000000	6.30982	6.062	6.08233	8.77063
36000000	6.64544	6.453	6.42044	9.16339
38000000	7.06408	6.844	6.95723	9.63802
40000000	7.39183	7.203	7.2189	10.2044
42000000	7.85039	7.547	7.57862	10.6432
44000000	8.20871	7.922	7.99303	11.2284
46000000	8.63117	8.296	8.35755	11.7943
48000000	8.97401	9.281	8.72503	12.3693
50000000	9.42294	9.297	9.10453	12.9819

5.2.3 Heapsort

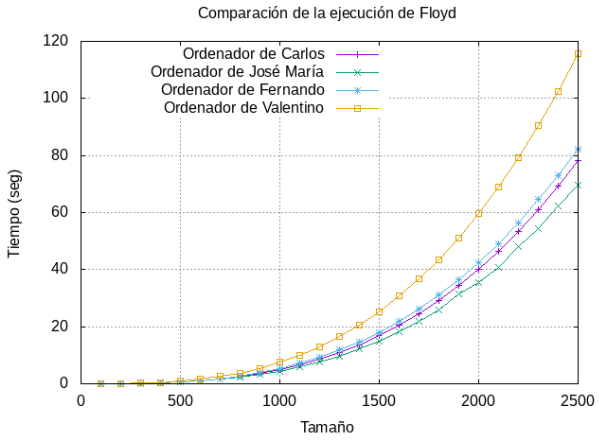


En el caso de Heapsort podemos ver que los resultados del ordenador de José María se desmarca de una forma más clara por debajo respecto a los demás. Además cabe destacar que ahora la diferencia de tiempos entre los resultados del ordenador de Valentino no se diferencian tanto del siguiente ordenador más lento, que en este caso es el de Carlos.

Tabla de tiempos de ejecución de Heapsort

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
2000000	0.545117	0.421	0.542522	0.739498
4000000	1.2418	1	1.24752	1.67554
6000000	2.1722	1.609	2.03784	2.68704
8000000	3.18846	2.328	2.8495	3.7336
10000000	4.23569	3.235	3.74796	4.83078
12000000	5.19928	4.125	4.58797	5.94822
14000000	6.23919	4.813	5.58601	7.09888
16000000	7.35151	5.359	6.49861	8.25001
18000000	8.52494	6.109	7.50533	9.44086
20000000	9.47554	6.797	8.35058	10.6328
22000000	10.6899	7.594	9.39312	11.8481
24000000	11.8352	8.36	10.3612	13.058
26000000	13.0459	9.188	11.4544	14.2786
28000000	14.293	10.016	12.564	15.5475
30000000	15.588	10.828	13.4265	16.7931
32000000	16.6321	11.625	14.6564	18.0657
34000000	17.8991	12.5	15.4951	19.33
36000000	19.4537	13.344	16.8176	20.7247
38000000	20.4865	14.25	17.5891	21.8856
40000000	21.732	15.125	19.0147	23.2894
42000000	23.3188	15.969	19.9958	24.6015
44000000	24.5466	16.859	21.1495	25.9224
46000000	25.997	17.766	21.6883	27.3435
48000000	27.0026	18.578	23.059	28.6372
50000000	28.8955	19.547	23.84	29.938

5.3 O(n^3) Floyd

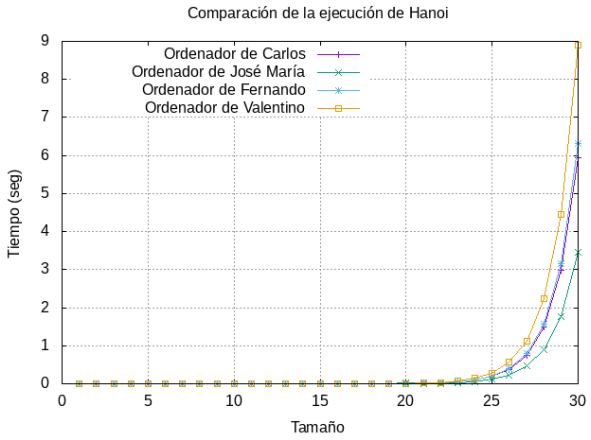


En el algoritmo de Floyd volvemos a apreciar esa diferencia entre los tiempos obtenidos por el ordenador de Valentino y los obtenidos por el segundo por arriba (el ordenador de Fernando). Por debajo, también se aprecia una ligera diferencia entre los tiempos obtenidos por el ordenador de Carlos y los del ordenador de José María, ya que estos últimos son visiblemente mejores en tamaños mayores.

Tabla de tiempos de ejecución de Floyd

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
100	0.006378	0	0.005598	0.009209
200	0.043939	0.031	0.043879	0.060489
300	0.135267	0.125	0.143285	0.201534
400	0.327425	0.281	0.339269	0.477764
500	0.628854	0.562	0.659233	0.926686
600	1.068	0.968	1.13466	1.59849
700	1.6958	1.515	1.80833	2.5411
800	2.54234	2.25	2.70379	3.80499
900	3.61681	3.187	3.87963	5.42943
1000	4.92116	4.36	5.36983	7.50204
1100	6.56611	5.875	7.17361	10.0139
1200	8.54384	7.61	9.3167	13.0104
1300	10.9186	9.719	11.7914	16.5157
1400	13.6697	12.156	14.7001	20.6496
1500	16.8305	14.922	18.0648	25.2944
1600	20.4285	18.266	21.8979	30.7149
1700	24.5637	21.828	26.207	36.7255
1800	29.102	25.828	31.0799	43.5798
1900	34.3412	31.343	36.3963	51.1791
2000	39.9981	35.375	42.4358	59.6761
2100	46.3464	40.875	48.9462	68.8502
2200	53.3862	48.172	56.3433	79.1134
2300	60.8446	54.25	64.479	90.3892
2400	69.1487	62.25	72.8983	102.562
2500	78.0665	69.547	82.1597	115.626

5.4 O(2^n) Hanoi



Al ser el algoritmo de las torres de Hanoi de orden exponencial, las diferencias entre ordenadores se observan realmente cuando el tamaño sobrepasa los 24-25 discos. A partir de dichos valores, vemos como el ordenador de Valentino nuevamente es el más leno, mientras que el ordenador de José María continúa dando los tiempos más rápidos. Sin embargo cabe a destacar que los otros dos ordenadores a comparar dan resultados muy similares.

Tabla de tiempos de ejecución de Hanoi

Tamaño	Tiempo			
	Carlos	José María	Fernando	Valentino
1	2e-06	0	2e-06	1e-06
2	1e-06	0	1e-06	1e-06
3	1e-06	0	1e-06	2e-06
4	1e-06	0	1e-06	2e-06
5	2e-06	0	1e-06	2e-06
6	2e-06	0	2e-06	3e-06
7	2e-06	0	3e-06	3e-06
8	4e-06	0	3e-06	5e-06
9	6e-06	0	5e-06	8e-06
10	9e-06	0	8e-06	1.3e-05
11	1.7e-05	0	1.4e-05	2.1e-05
12	3e-05	0	2.6e-05	4e-05
13	5.6e-05	0	5e-05	7.7e-05
14	0.000109	0	9.8e-05	0.000151
15	0.000215	0	0.000194	0.000299
16	0.000419	0	0.000406	0.000595
17	0.000823	0	0.000768	0.001189
18	0.001751	0	0.00156	0.002372
19	0.003343	0	0.003177	0.004359
20	0.006455	0.015	0.006355	0.008734
21	0.012614	0	0.012496	0.017462
22	0.024741	0	0.025164	0.034868
23	0.050059	0.031	0.050047	0.071154
24	0.094086	0.046	0.10044	0.140166
25	0.188492	0.125	0.197887	0.279998

6 Conclusiones

Como conclusión de esta práctica, se ha podido observar de primera mano que tan sumamente importante es la eficiencia de un algoritmo, la manera que un algoritmo realiza una tarea puede resultar en comportamientos totalmente diferentes, por ejemplo, los algoritmos de ordenación, los algoritmos cuadráticos y los casi lineales poseen diferencias en tiempo muy contrastantes, o bien, lo rápido que una entrada para el algoritmo de Floyd o Hanoi puede pasar de tardar segundos a minutos. Es algo más fundamental que el hecho de poseer una máquina más rápida o de optimizar la compilación, si bien ayudan a reducir el tiempo por medio de las constantes ocultas pero, el buen diseño de un algoritmo en la mayoría de los casos 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, tomando en cuenta que los mejores beneficios se verán en tamaños de entrada cada vez más grandes, en otros casos podrá ser que no se cumpla, aunque esto ya dependerá del tamaño de las constantes ocultas.