Práctica 1: Eficiencia

Algorítmica

Universidad de Granada

15 de marzo de 2016

Indice

1. Introducción
2. Ordenación
   - Burbuja
   - Selección
   - Inserción
3. Ordenación rápida
   - Quick Sort
   - Heap Sort
   - Merge Sort
4. Floyd
5. Hanoi

Introducción

- El objetivo de ésta práctica es analizar eficiencias de forma empírica e híbrida.
- Para ello, hemos recogido los diferentes tiempos de los diferentes algoritmos que se ofrecían y los hemos comparado.
- En nuestro caso concreto, hemos utilizado la biblioteca de C++ más moderna y precisa destinado a obtener tiempos de reloj: la biblioteca **chrono**

Especificaciones de la máquina

- Procesador: Intel Core i5-3337U (2.7GHz x 2)
- Memoria RAM: 4GB
- Disco Duro: 500GB 5400 rpm
- SO: Manjaro Linux 15.2 Capella 64 bits

Ordenación

Según hemos ido hemos estudiado, estos algoritmos que presentamos tienen teóricamente y calculando a partir del código una eficiencia de $O(n^2)$. Como esto es teórico, vamos a ver si efectivamente (o no) los algoritmos proporcionados se parecen a la gráfica de n^2 recogiendo la información de 99 posibilidades distintas en cada algoritmo.

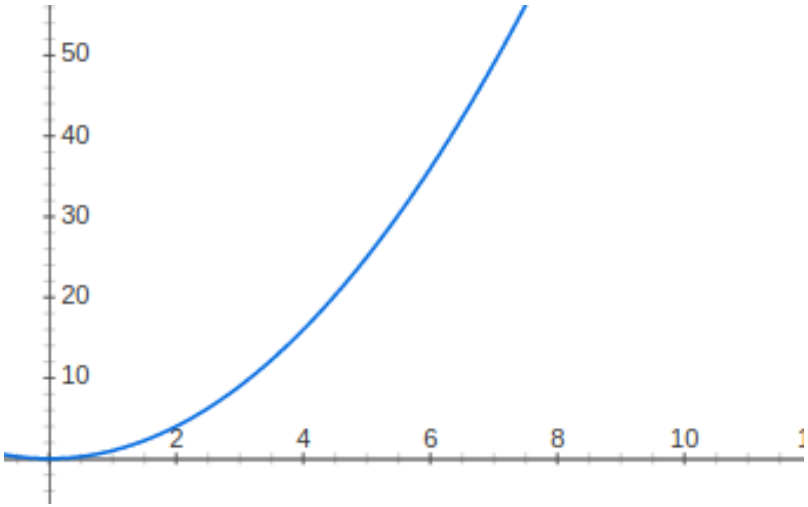

Figura : Gráfica de la función n^2

Burbuja

La gráfica empírica obtenida ha sido:

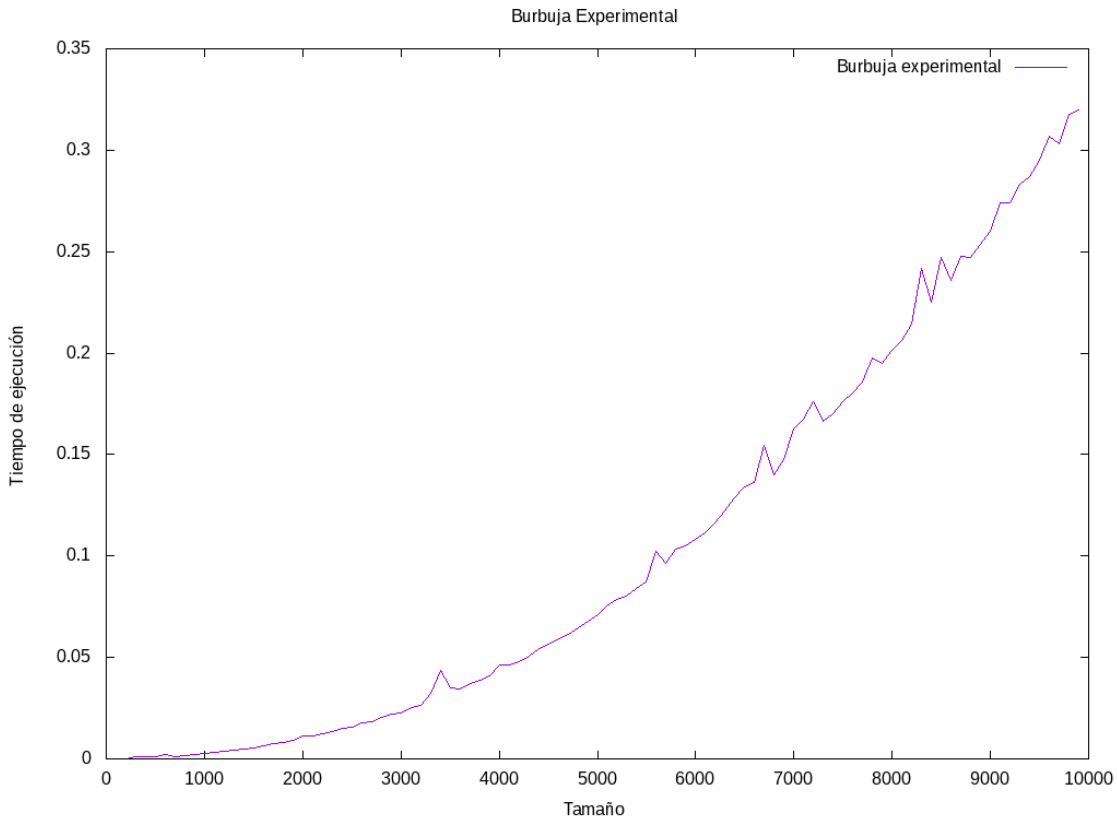

Figura : Algoritmo de Burbuja, gráfica empírica.

Burbuja

La gráfica híbrida obtenida ha sido:

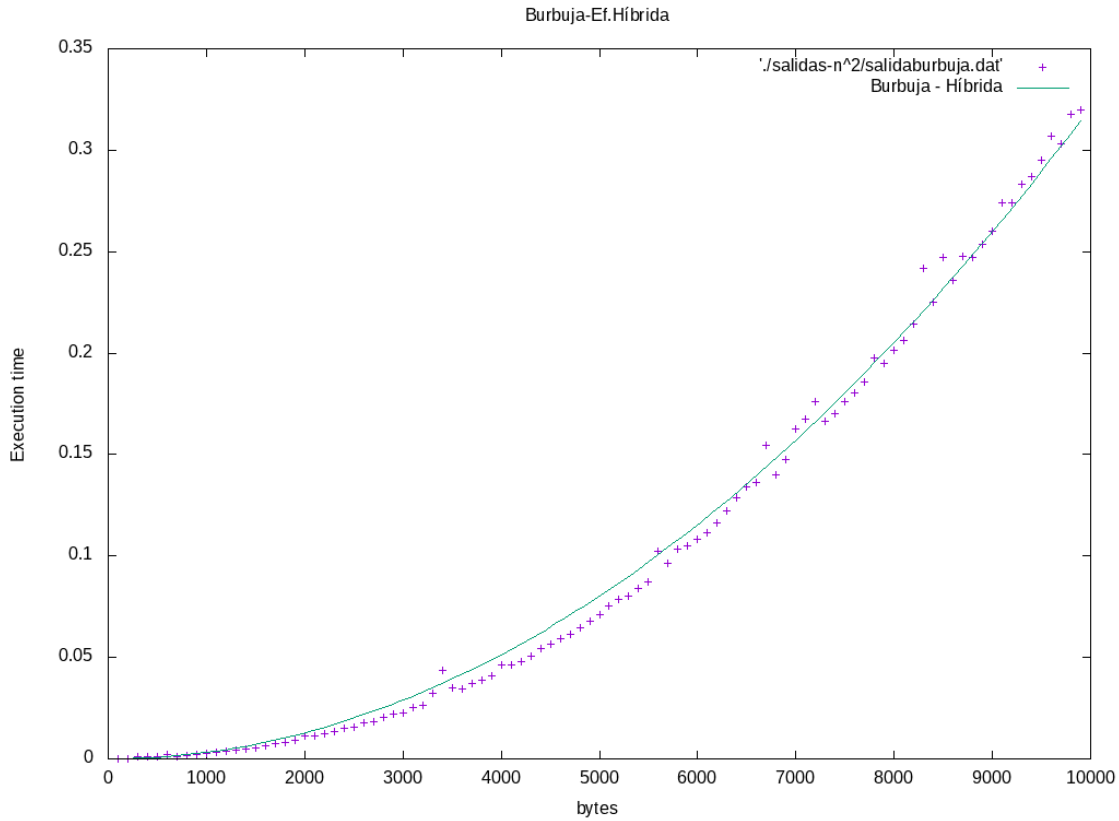

Figura : Ajuste híbrido algoritmo Burbuja y función n^2

Selección

La gráfica empírica obtenida ha sido:

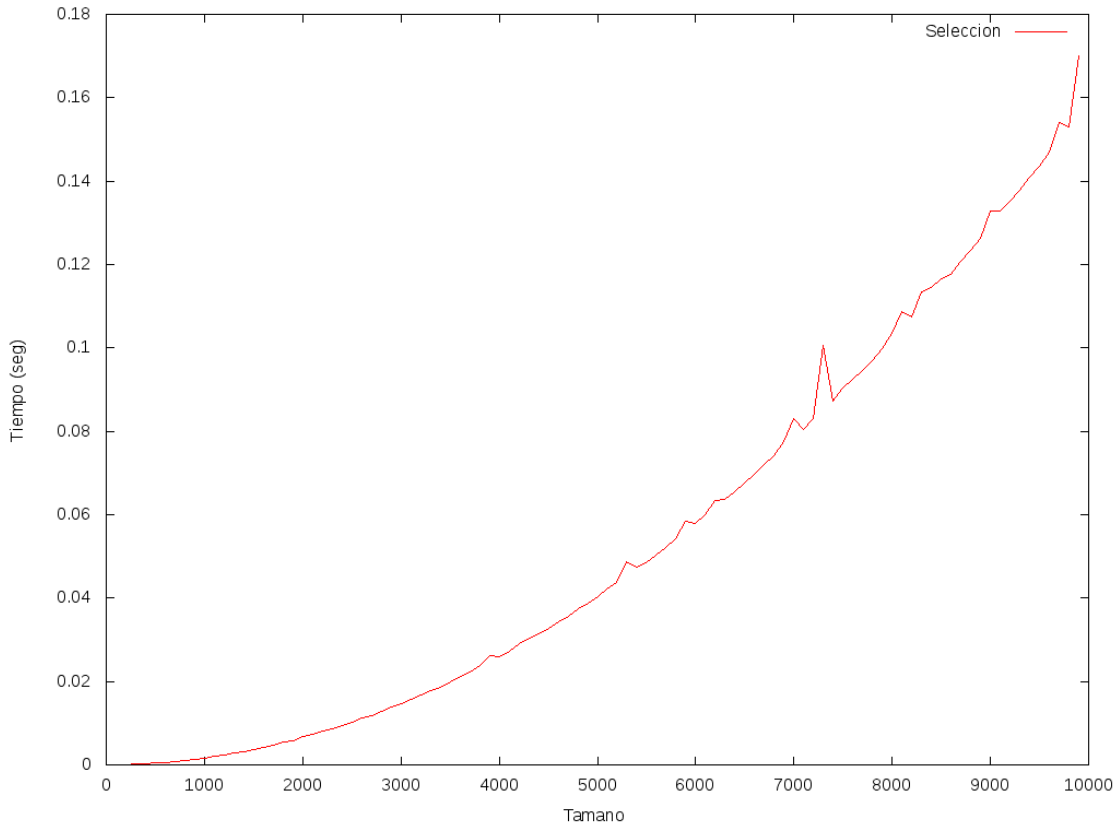

Figura : Algoritmo de selección, gráfica empírica.

Selección

La gráfica híbrida obtenida ha sido:

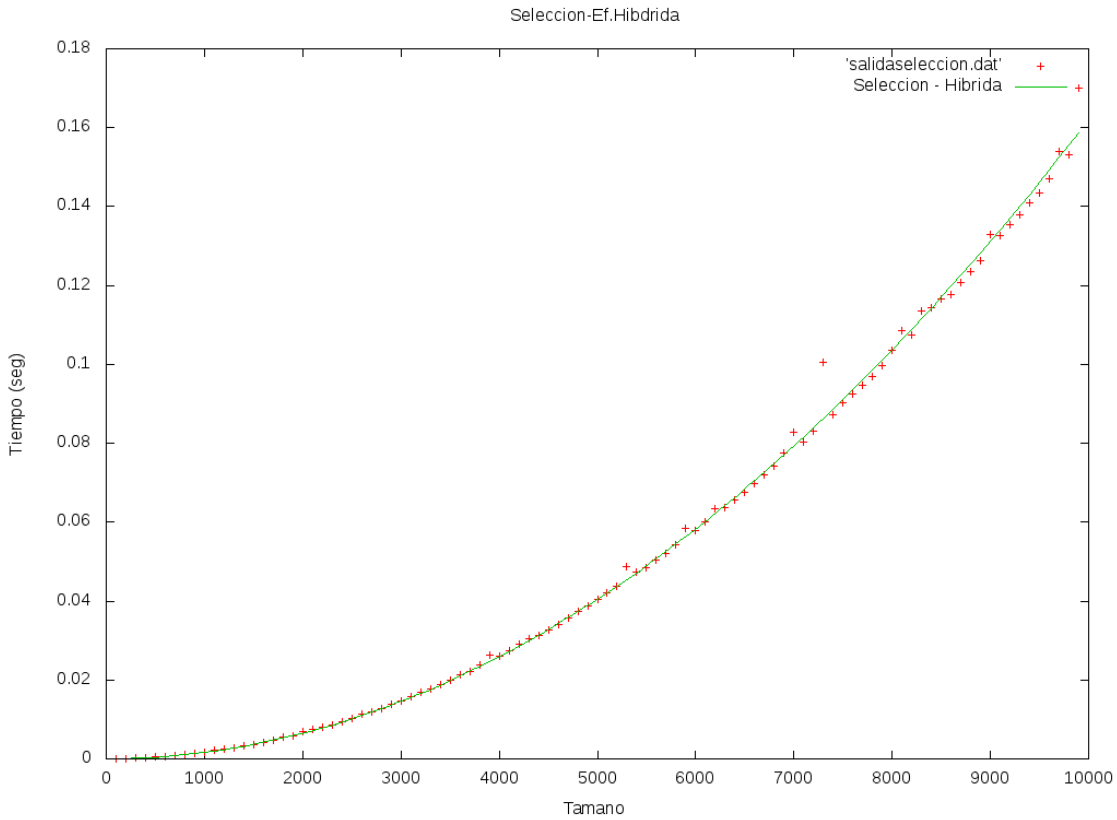

Figura : Gráfica ajuste híbrido. Selección y n^2

Inserción

La gráfica empírica obtenida ha sido:

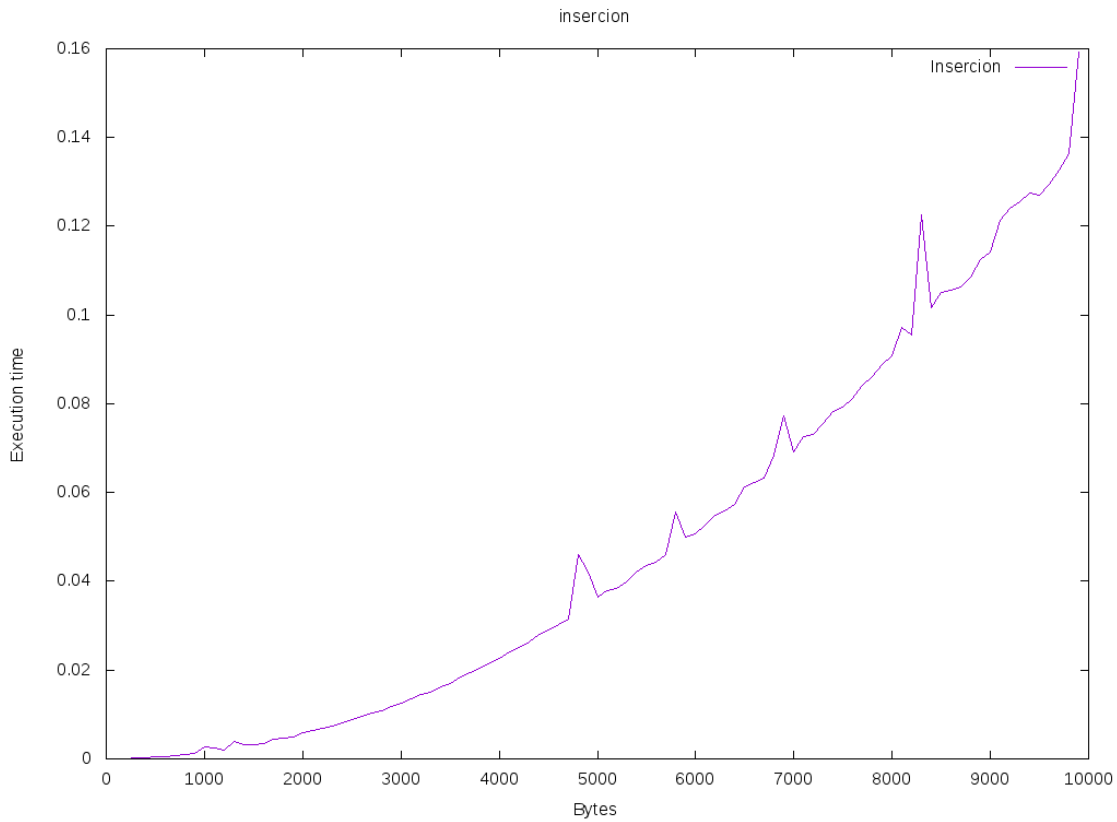

Figura : Algoritmo de Inserción, gráfica empírica.

Inserción

La gráfica híbrida obtenida ha sido:

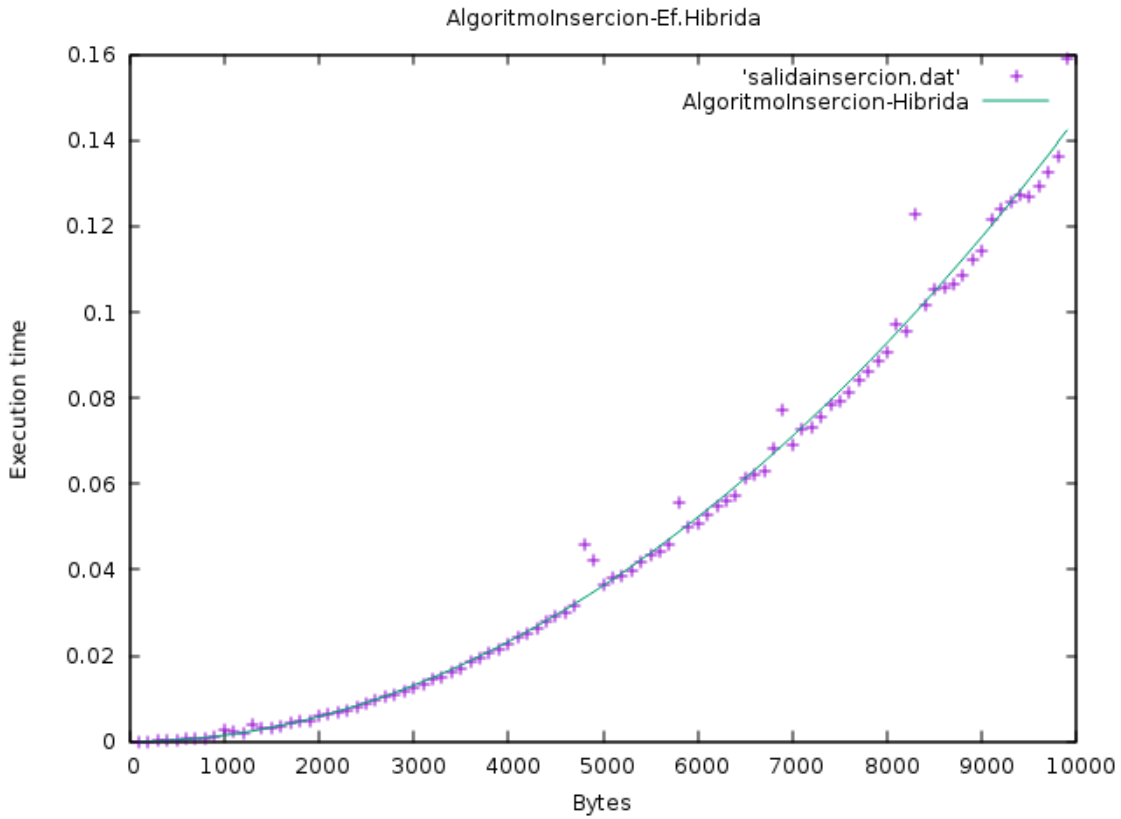

Figura : Gráfica ajuste híbrido. Inserción y n^2

Algoritmos n^2

Por último, mostramos el porcentaje de error así como las constantes ocultas obtenidas.

Algoritmo	**Constante Oculta**	**Error**
Burbuja	a0 = 3.20873e-09	+/- 1.403e-11 (0.4372 %)
Selección	a0 = 1.61988e-09	+/- 4.818e-12 (0.2975 %)
Inserción	a0 = 1.45151e-09	+/- 8.337e-12 (0.5743 %)

Ordenación rápida

Según hemos ido estudiado, estos algoritmos que presentamos tienen teóricamente y calculando a partir del código una eficiencia de $O(n * \log(n))$. Como esto es teórico, vamos a ver si efectivamente (o no) los algoritmos proporcionados se parecen a la gráfica de $n * \log(n)$ recogiendo la información de 99 posibilidades distintas en cada algoritmo.

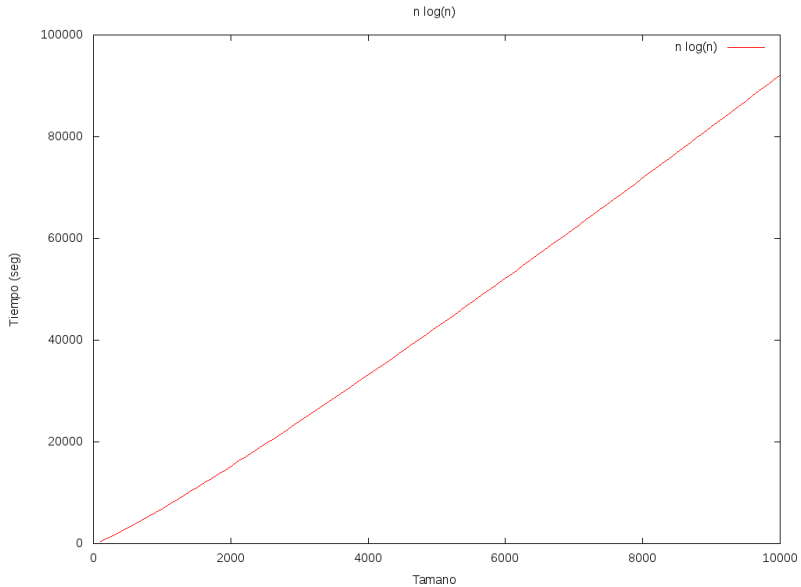

Figura : Gráfica de la función $n * log(n)$

Quick Sort

La gráfica empírica obtenida ha sido:

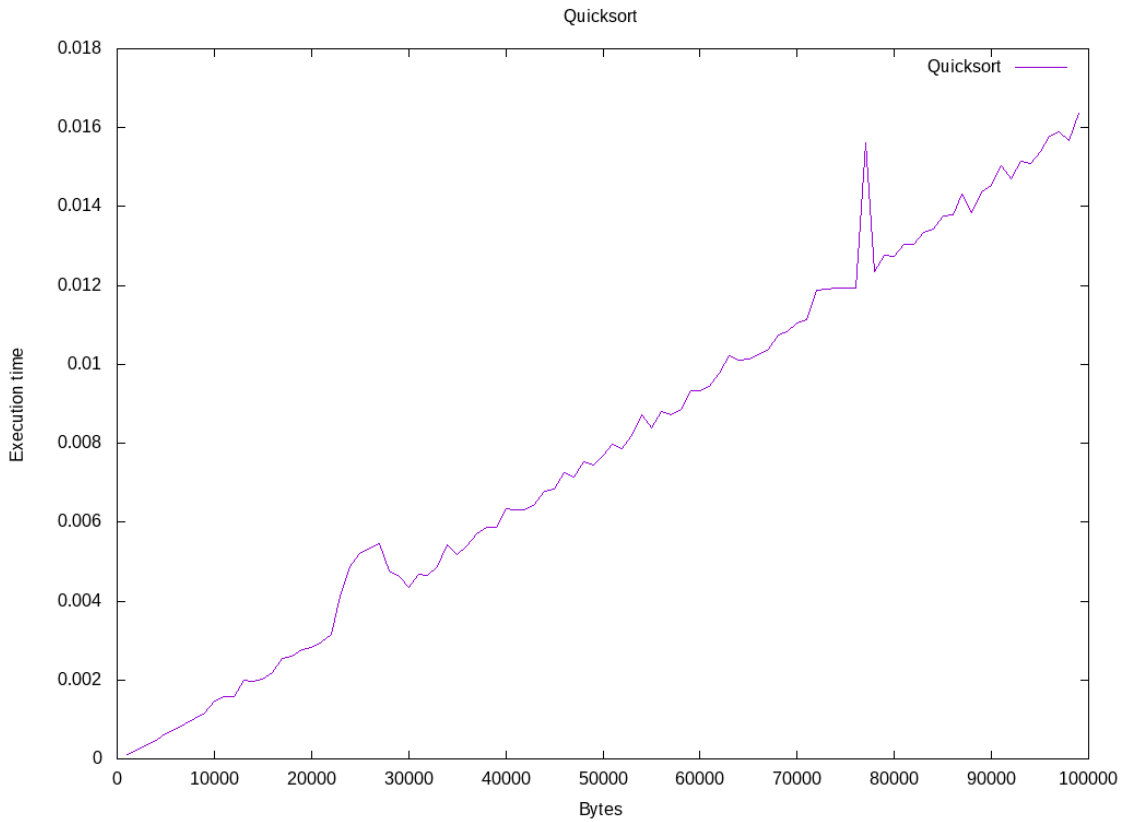

Figura : Algoritmo Quicksort, gráfica empírica.

Quick Sort

La gráfica híbrida obtenida ha sido:

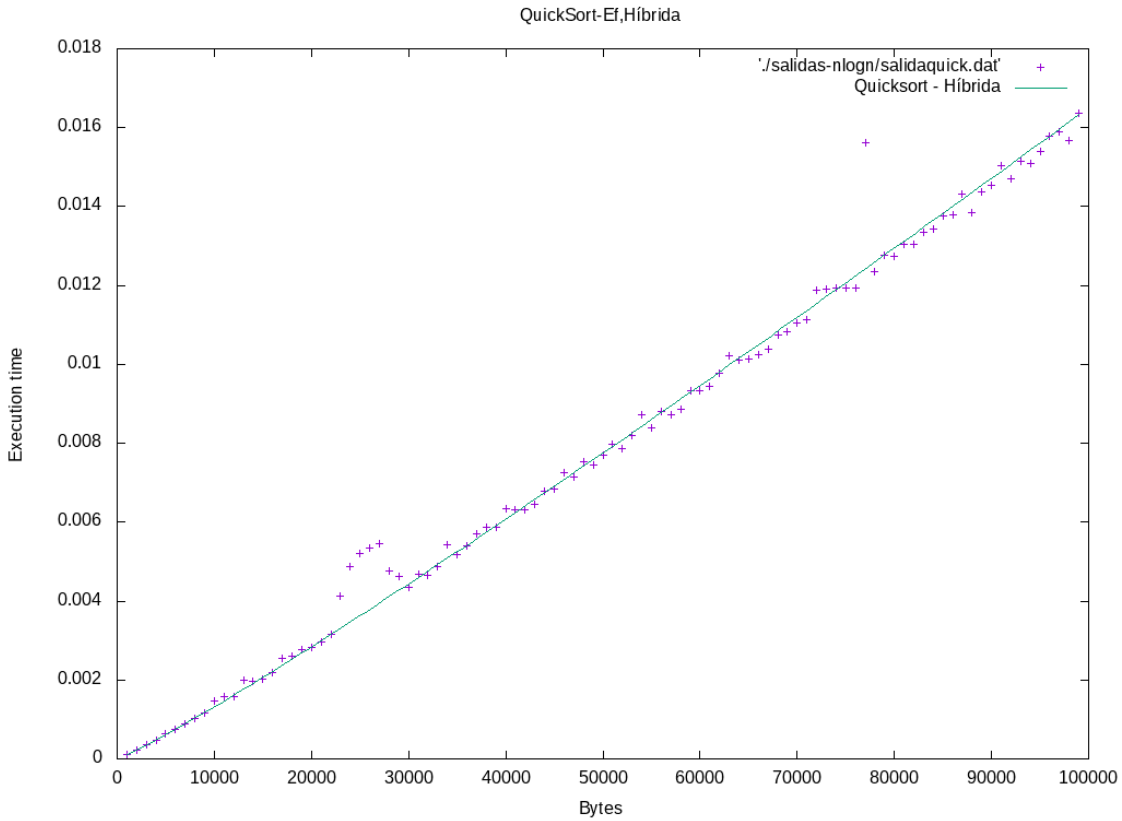

Figura : Gráfica híbrida Quicksort.

Heap Sort

La gráfica empírica obtenida ha sido:

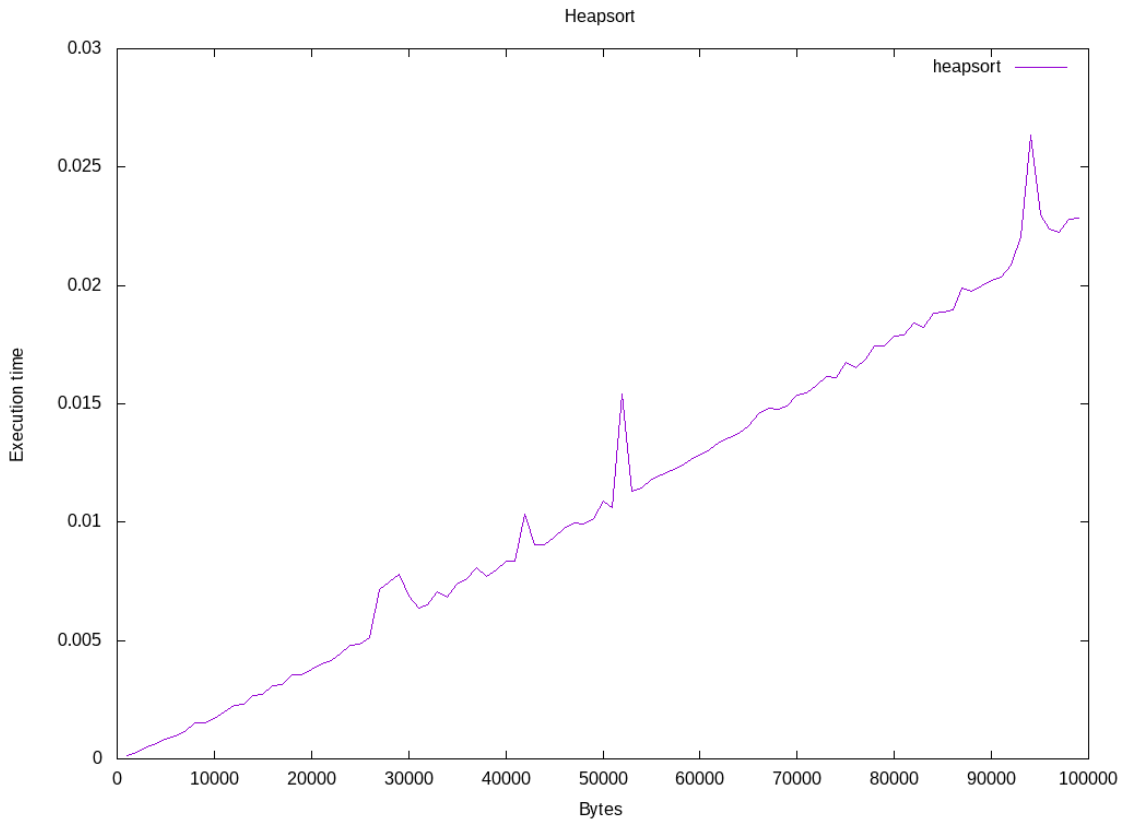

Figura : Heapsort, gráfica empírica.

Heap Sort

La gráfica híbrida obtenida ha sido:

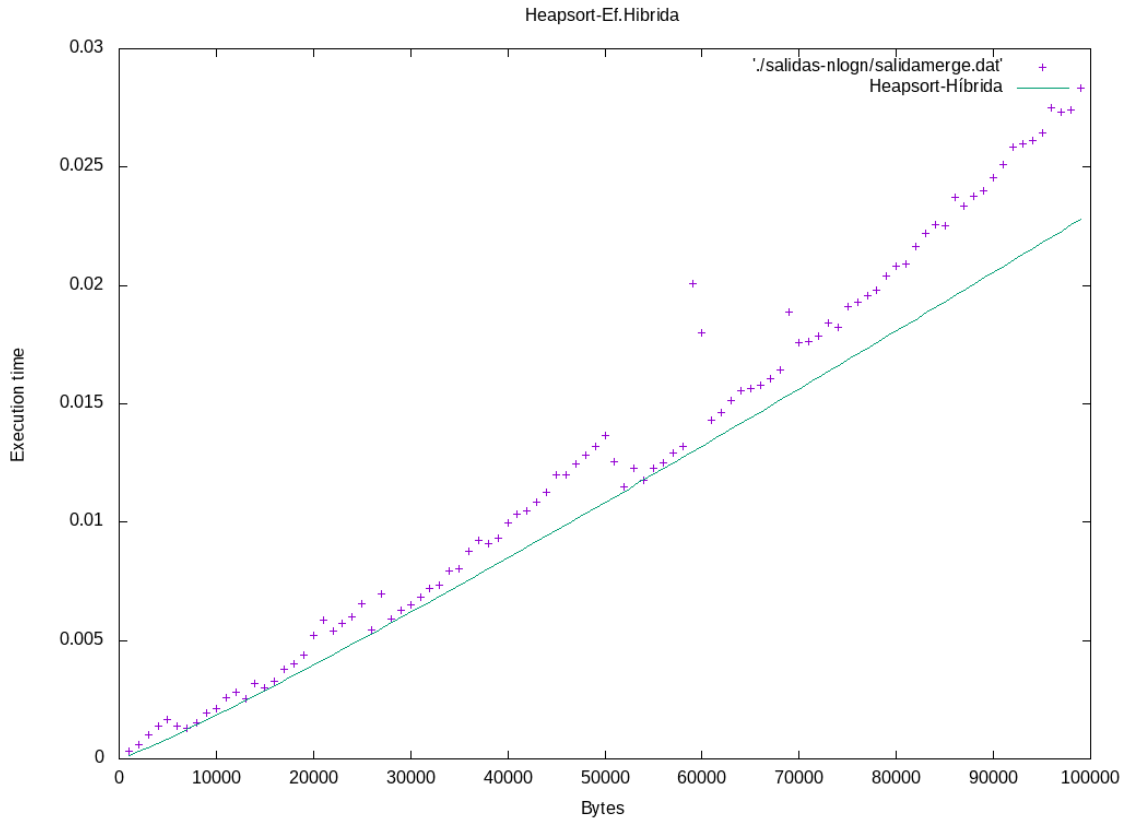

Figura : Heapsort, gráfica híbrida.

Heap Sort

Este algoritmo lo hemos ejecutado en otra máquina más potente, de esta manera, podemos ver como varían los tiempos de ejecución del mismo algoritmo, ejecutado en dos máquinas distintas.

- Procesador (frecuencia): Intel Core i7-5500U (3.0 GHz x 2)
- Memoria RAM: 4 GB
- Disco duro: SSD 256 GB
- S.O: Linux Mint 17.3 Cinnamon 64-bit

Heap Sort

El ajuste de ambas gráficas es el siguiente:

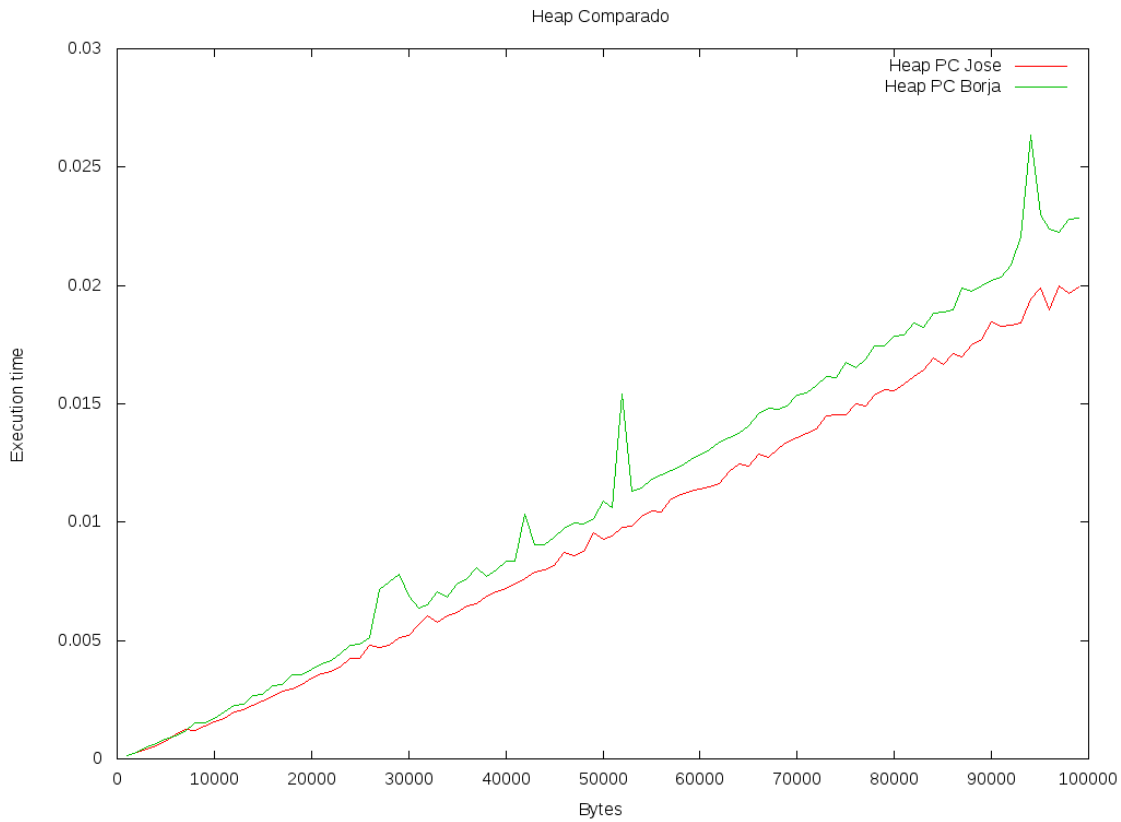

Figura : Comparación en dos máquinas. Heapsort.

Merge Sort

La gráfica empírica obtenida ha sido:

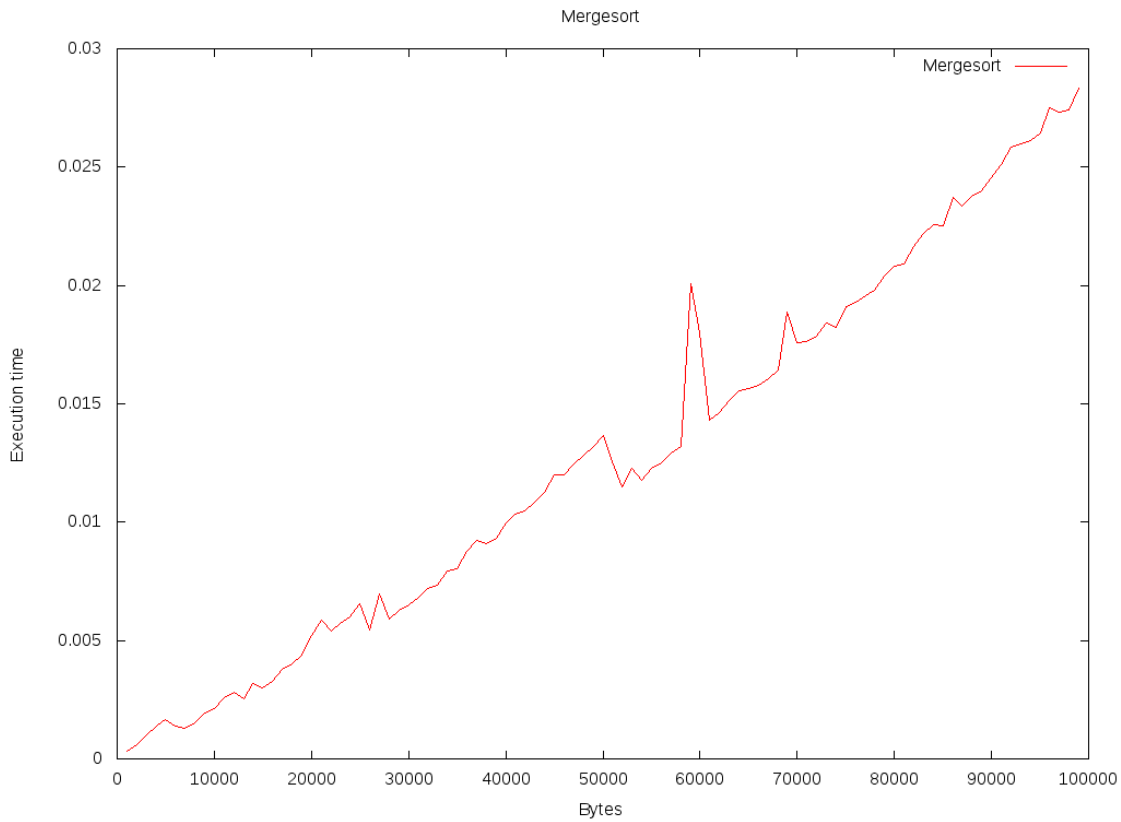

Figura : Mergesort, gráfica empírica.

Merge Sort

La gráfica híbrida obtenida ha sido:

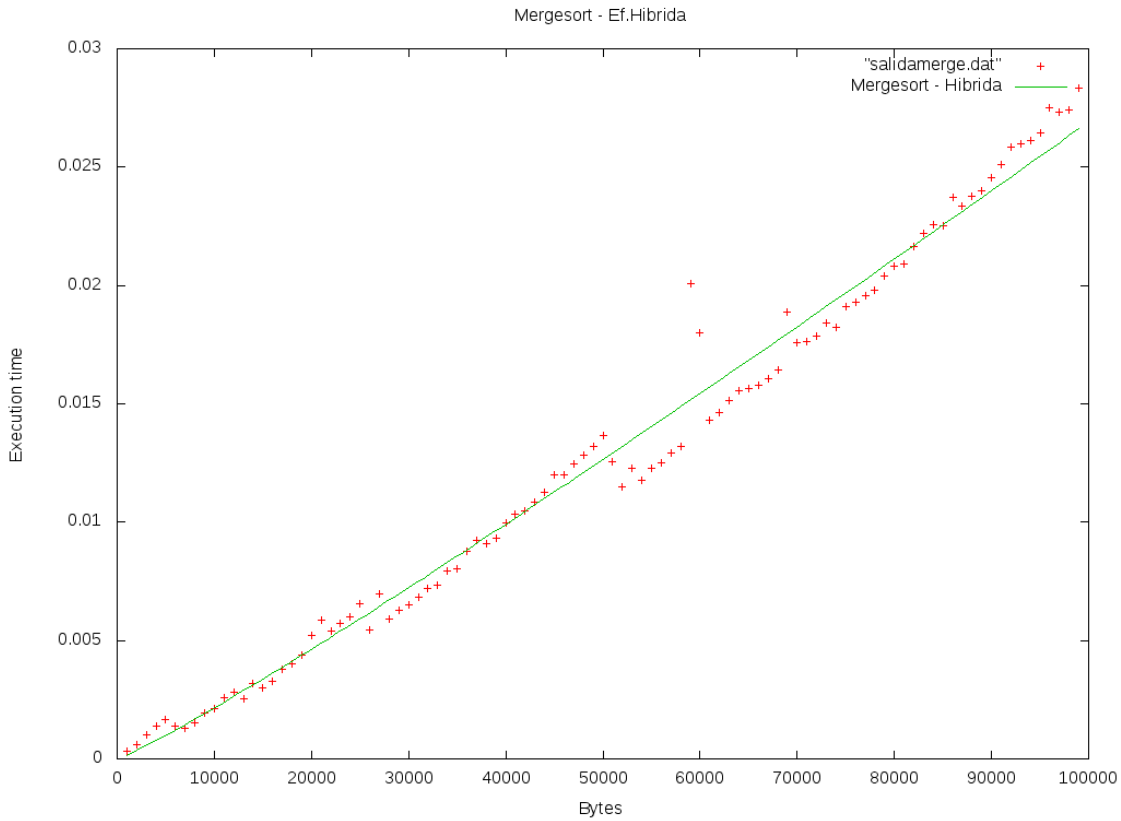

Figura : Mergesort, gráfica híbrida.

Algoritmos $n * log(n)$

Por último, mostramos el porcentaje de error así como las constantes ocultas.

Algoritmo	**Constante Oculta**	**Error**
Mergesort	a0 = 2.33821e-08	+/- 1.564e-10 (0.6689 %)
Quicksort	a0 = 1.43368e-08	+/- 7.621e-11 (0.5315 %)
Heapsort	a0 = 2.00227e-08	+/- 1.222e-10 (0.6104 %)

Floyd

El algoritmo de Floyd, a diferencia de los anteriores, no es un algoritmo de ordenación. Su función, es la de encontrar el camino mínimo en grafos. La eficiencia teórica de este algoritmo es $O(n^3)$. Además de analizar la eficiencia empírica e híbrida, hemos realizado un ajuste erróneo, para demostrar que efectivamente la eficiencia de este algoritmo es la anteriormente mencionada.

Floyd

La gráfica empírica obtenida ha sido:

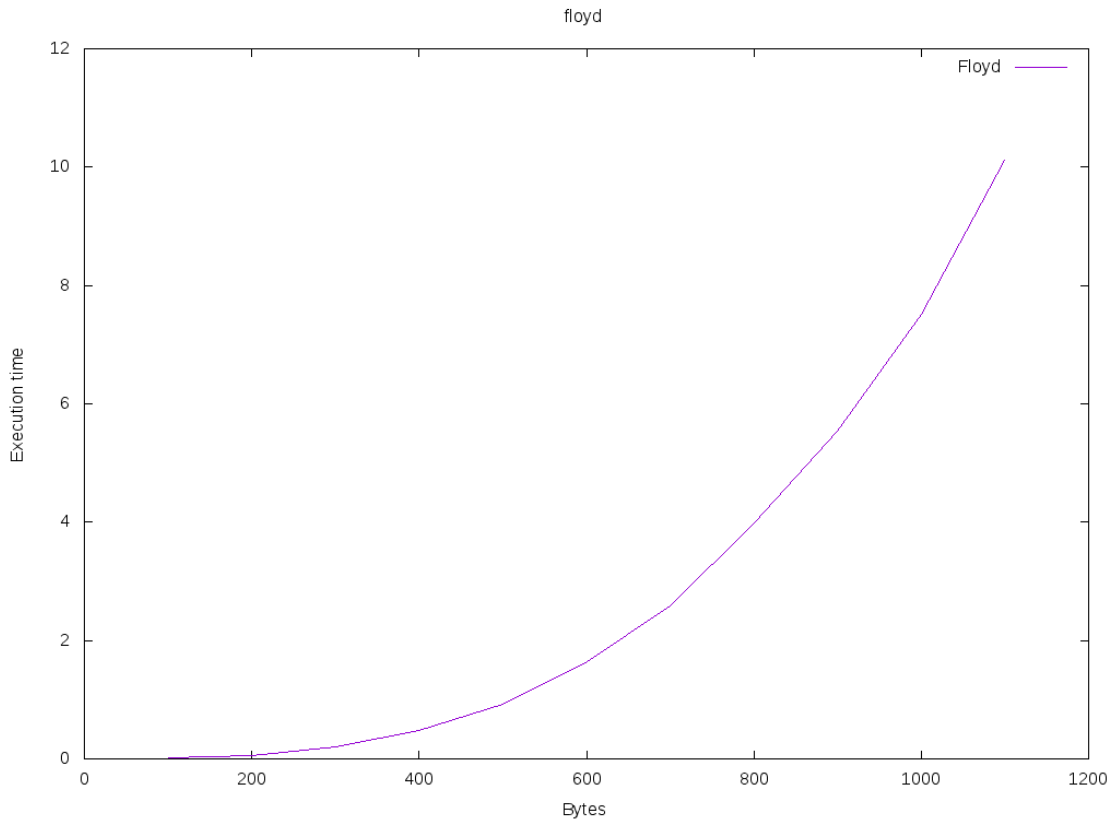

Figura : Floyd, gráfica empírica.

Floyd

La gráfica híbrida obtenida ha sido:

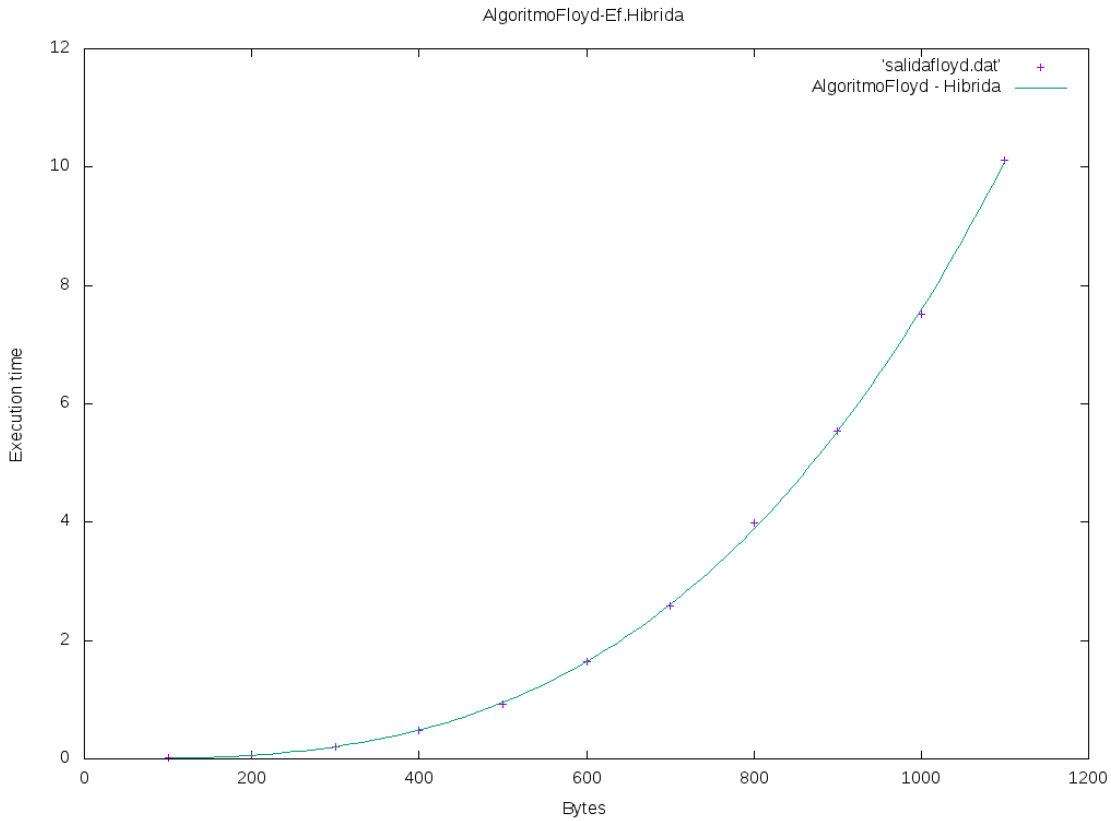

Figura : Floyd, gráfica híbrida.

Floyd

Ajuste híbrido erróneo ($O(n^3)$ ajustada a una función $O(n * log(n))$.

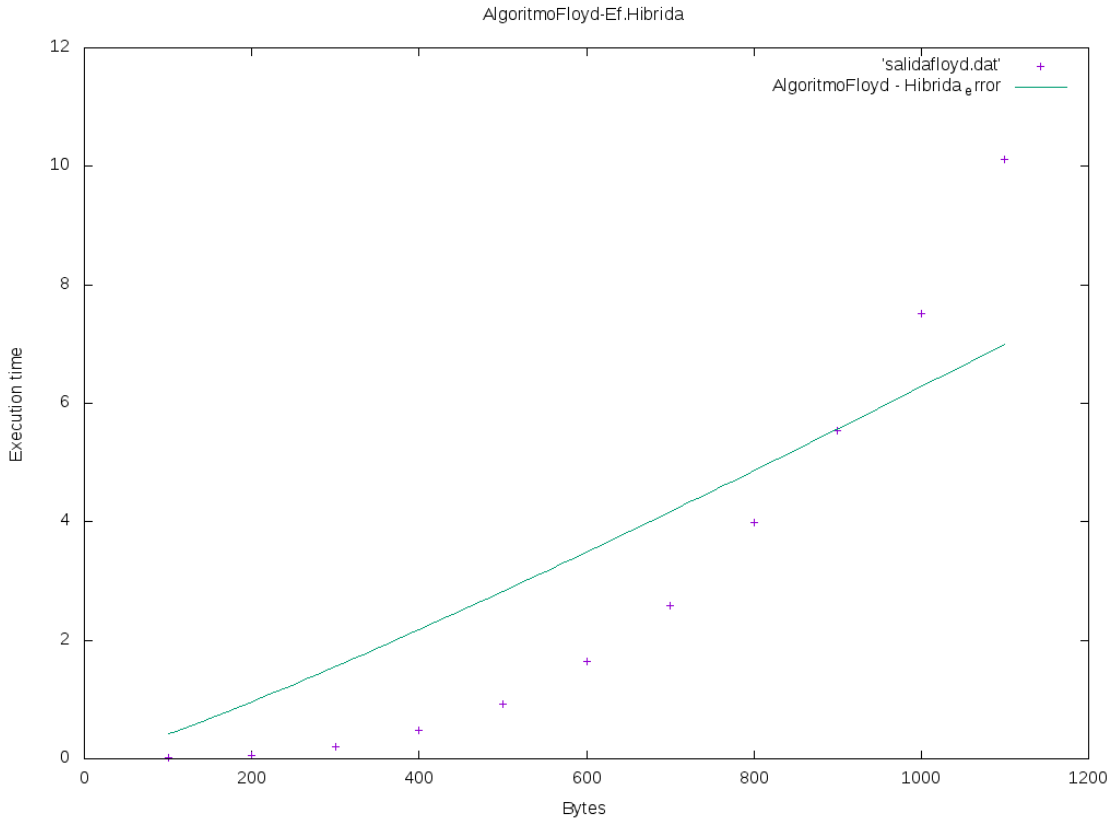

Figura : Floyd, ajuste erróneo.

Floyd $O(n^3)$

Por último, mostramos el porcentaje de error así como las constantes ocultas de ambos ajustes.

Algoritmo	**Constante Oculta**	**Error**
Floyd	a0 = 7.59068e-09	+/- 2.163e-11 (0.2849 %)
Floyd erróneo	a0 = 0.000908818	+/- 0.0001093 (12.03 %)

Hanoi

El algoritmo de Hanoi, se encarga específicamente de resolver el problema de las torres de Hanoi. Este algoritmo presenta una eficiencia teórica de $O(2^n)$, lo que hace que el número de entradas para este algoritmo sea muy reducido.

Hanoi

La gráfica empírica obtenida ha sido:

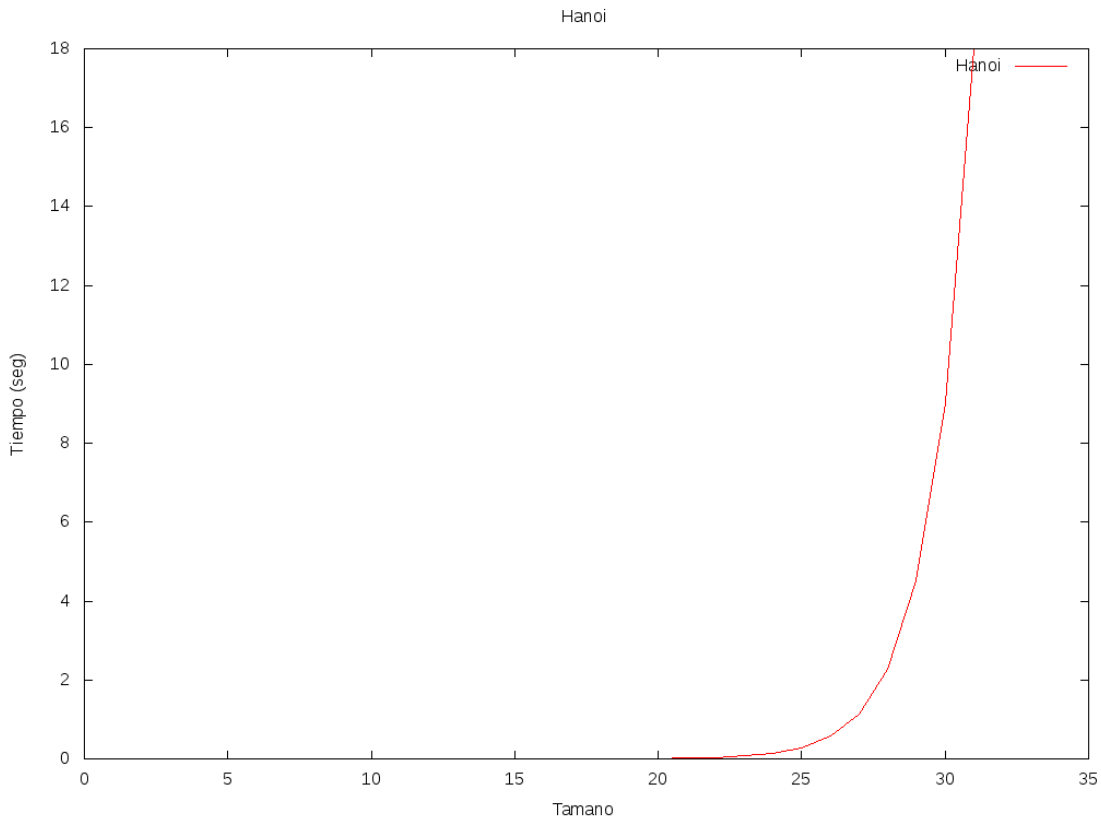

Figura : Hanoi, gráfica empírica.

Hanoi

La gráfica híbrida obtenida ha sido:

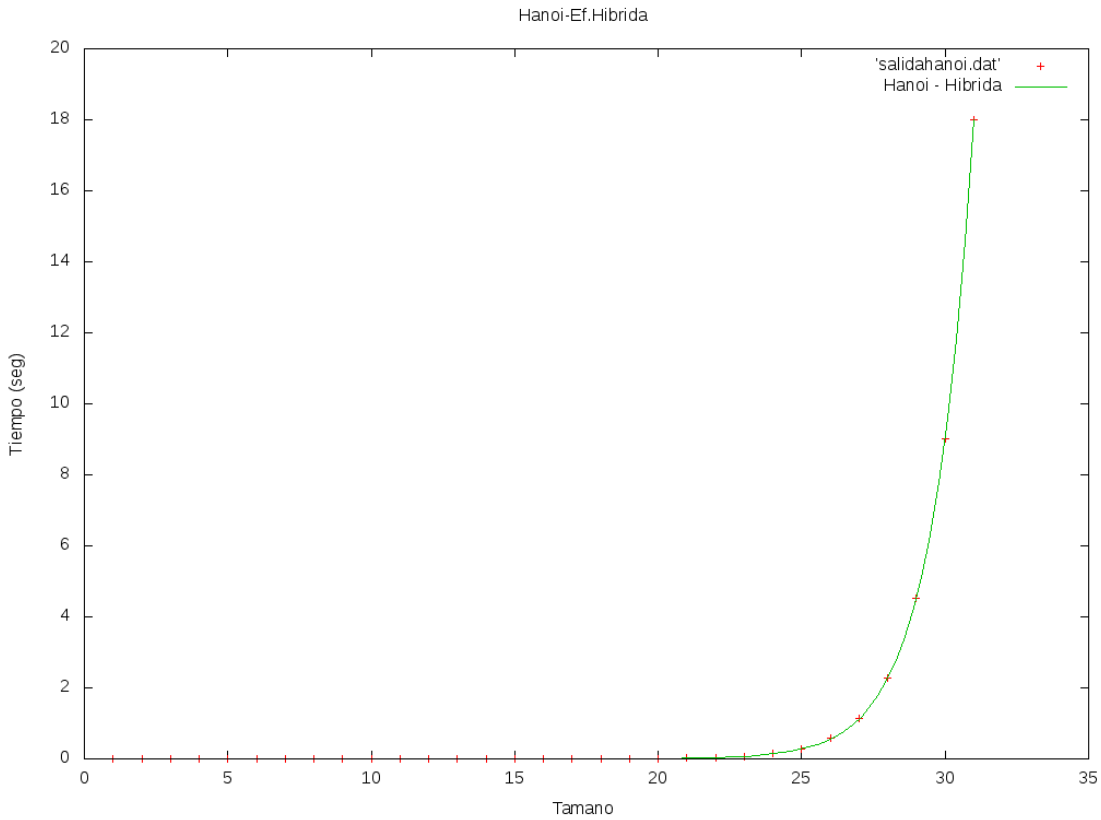

Figura : Hanoi, gráfica híbrida.

Hanoi

En este algoritmo hemos hecho varias comparaciones:

- Usando diferentes optimizaciones a la hora de compilar.
- Utilizando un lenguaje de programación distinto (Python vs C++).

Hanoi en Python

La gráfica híbrida de la ejecución en Python frente a la ejecución en C++ es:

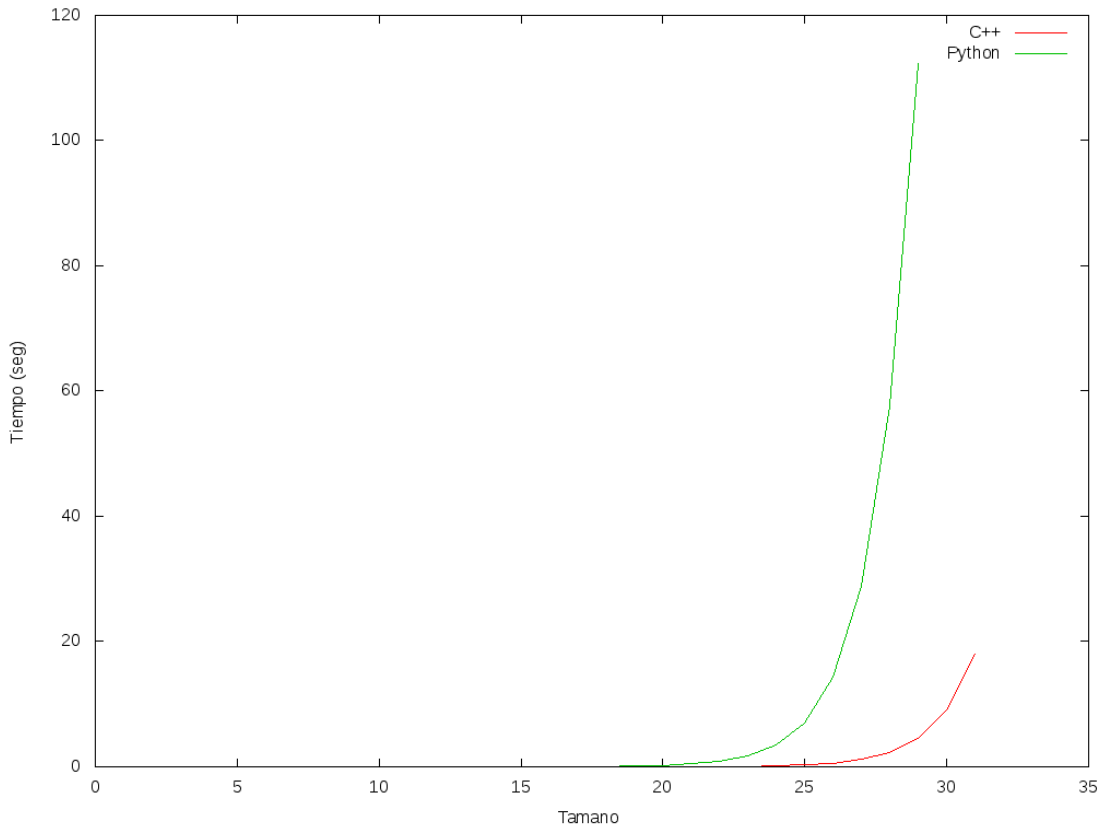

Figura : Hanoi, Python - C++

Hanoi con diferentes optimizaciones

A continuación mostramos una gráfica con los tiempos del algoritmo en función de su optimización:

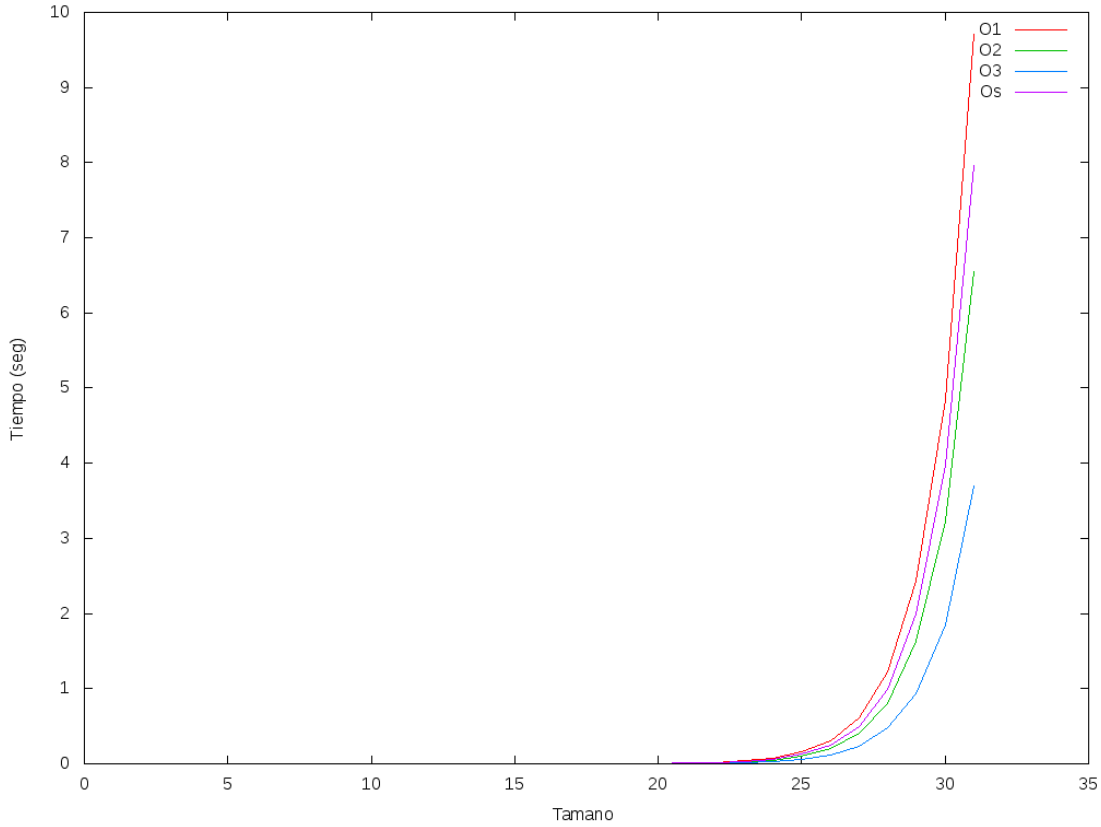

Figura : Hanoi, gráfica con diferentes eficiencias.

Hanoi $O(2^n)$

Por último, mostramos el porcentaje de error así como las constantes ocultas.

Algoritmo	**Constante Oculta**	**Error**
Hanoi	a0 = 8.38461e-09	+/- 3.095e-12 (0.03691 %)

¿Preguntas?