Algorítmica (2015-2016)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 1: Análisis de eficiencia de algoritmos

Francisco Carrillo Pérez,Borja Cañavate Bordons,
Miguel Porcel Jiménez,Jose Manuel Rejón Santiago,Jose Arcos Aneas

15 de marzo de 2016

Índice

Índice de figuras

Índice de tablas

1. 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**
La máquina que hemos utilizado tiene las siguientes características:

- 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

2. 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.

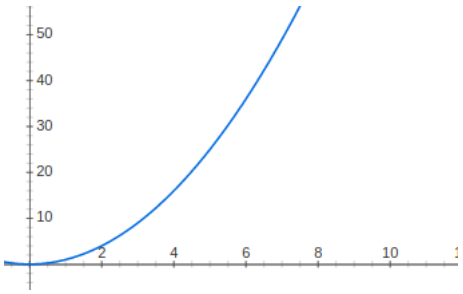

Figura 2.1: Gráfica de la función n^2

2.1. Burbuja

La gráfica empírica obtenida ha sido:

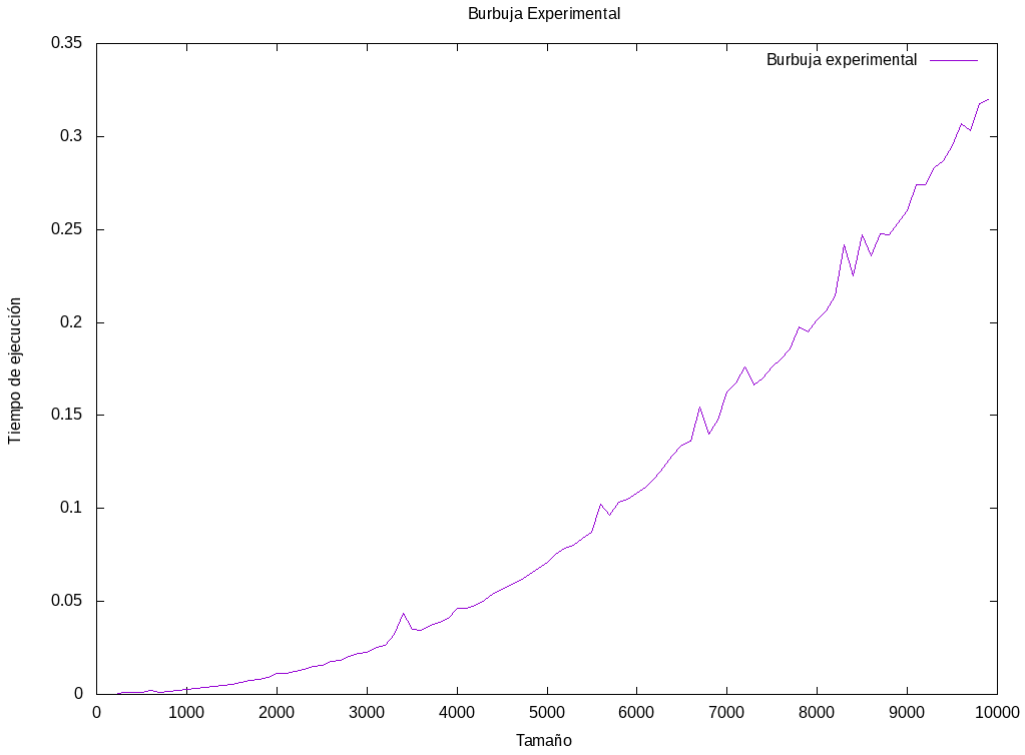

Figura 2.2: Algoritmo de Burbuja, gráfica empírica.

La gráfica híbrida obtenida ha sido:

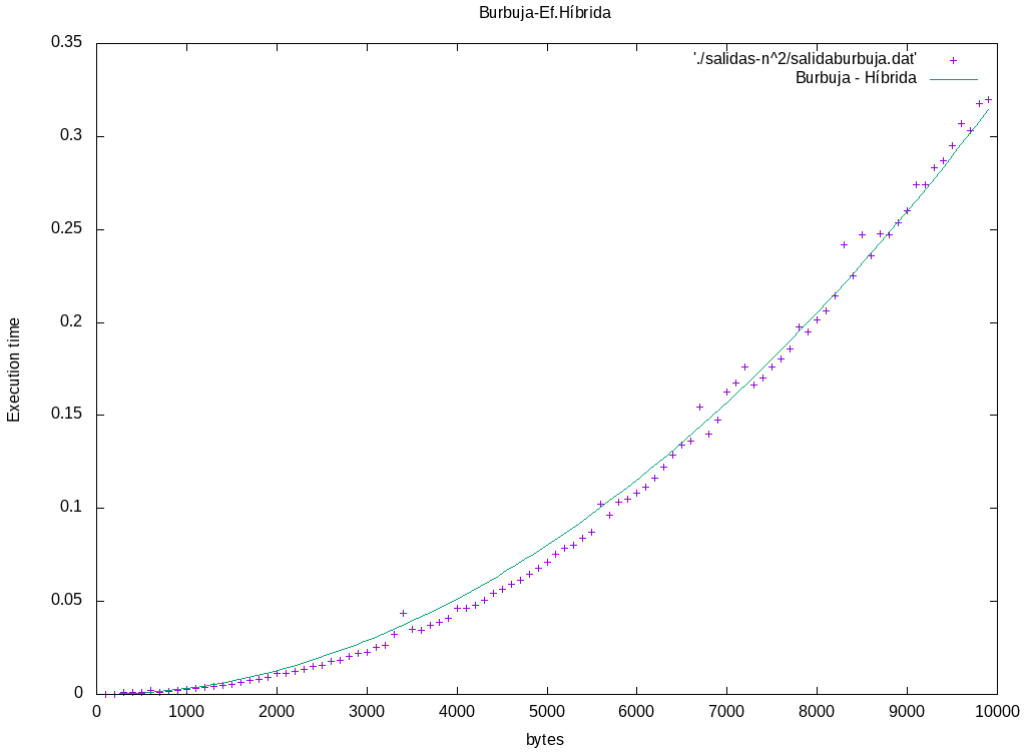

Figura 2.3: Ajuste híbrido algoritmo Burbuja y función n^2

2.2. Selección

La gráfica empírica obtenida ha sido:

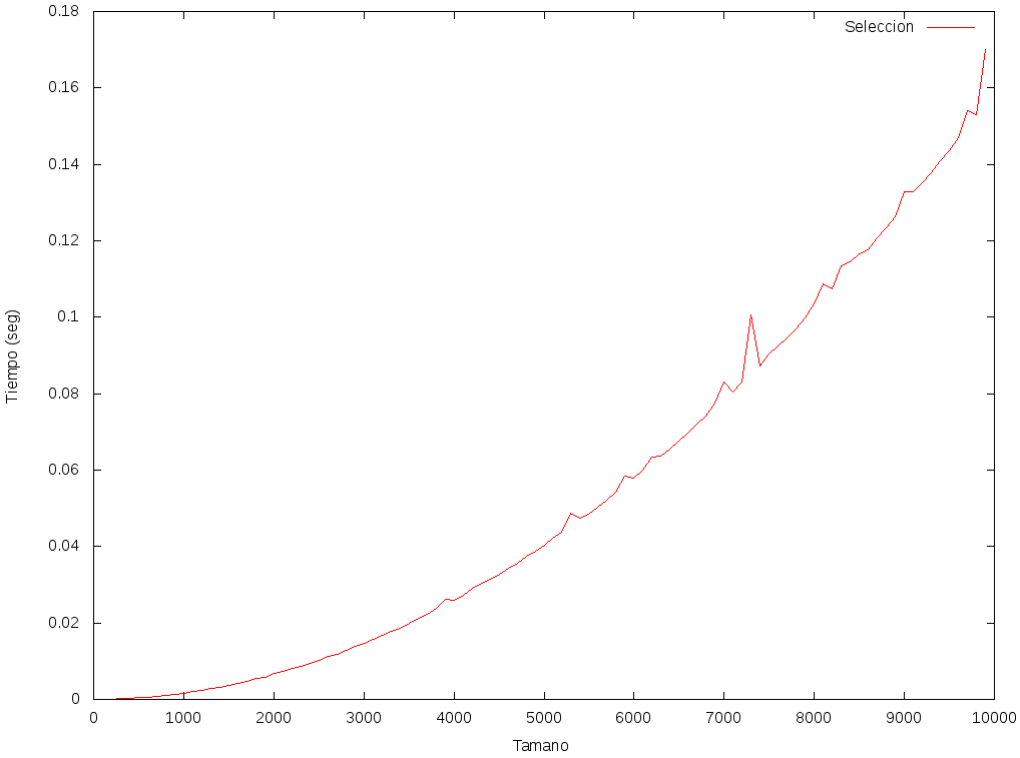

Figura 2.4: Algoritmo de selección, gráfica empírica.

La gráfica híbrida obtenida ha sido:

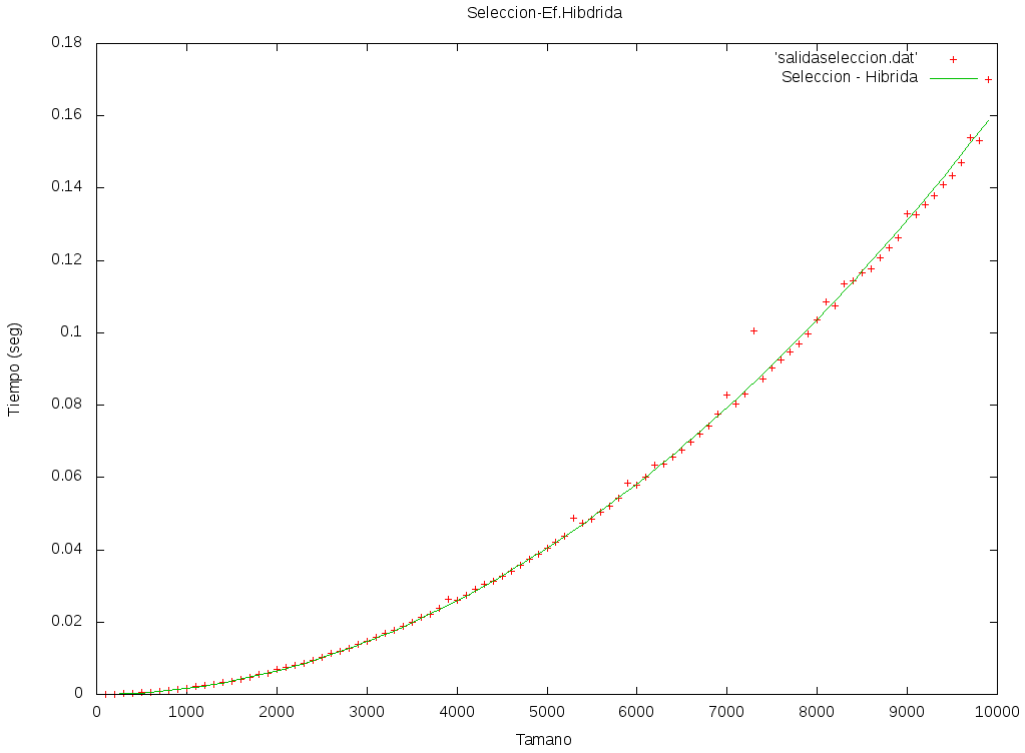

Figura 2.5: Gráfica ajuste híbrido. Selección y n^2

2.3. Inserción

La gráfica empírica obtenida ha sido:

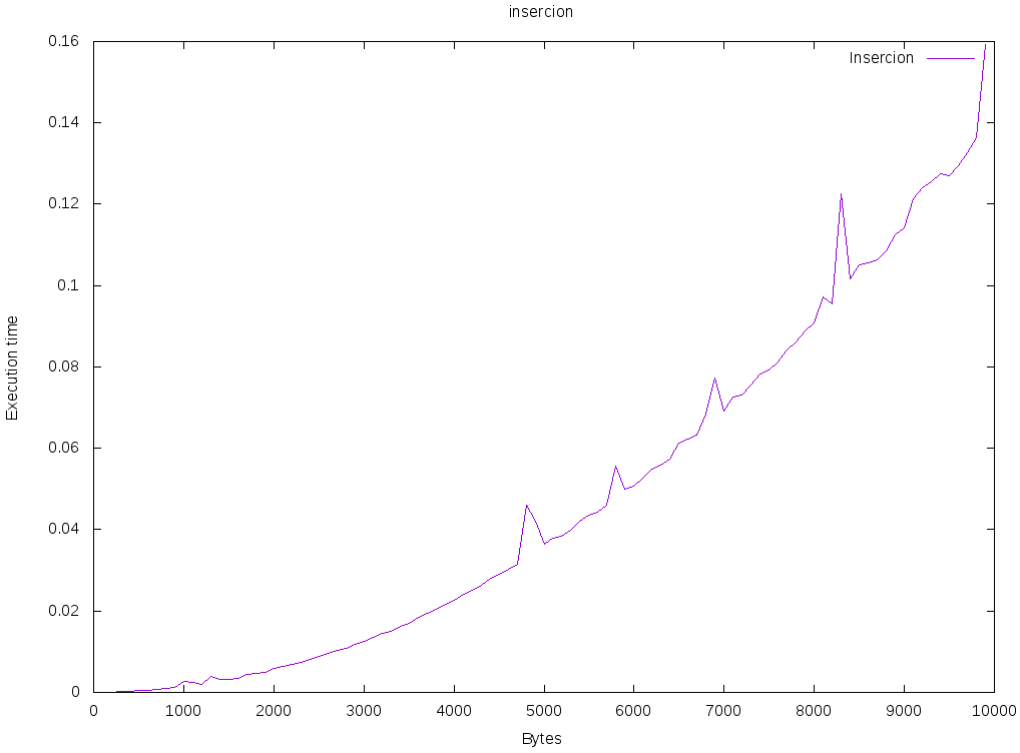

Figura 2.6: Algoritmo de Inserción, gráfica empírica.

La gráfica híbrida obtenida ha sido:

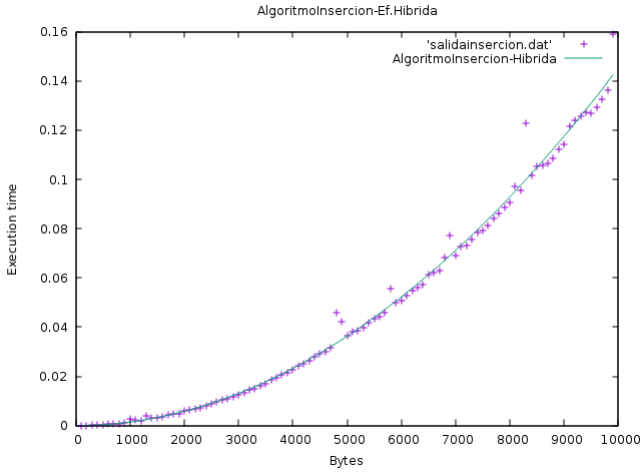

Figura 2.7: Gráfica ajuste híbrido. Inserción y n^2

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 %)

3. 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.

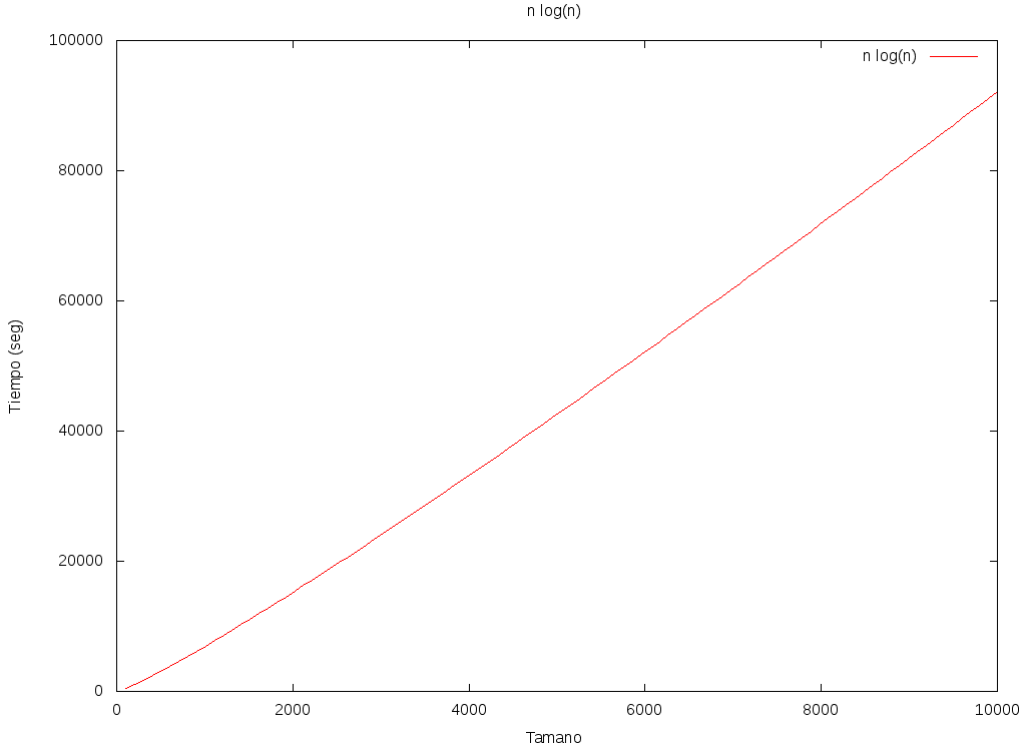

Figura 3.1: Gráfica de la función $n * log(n)$

3.1. Quick Sort

La gráfica empírica obtenida ha sido:

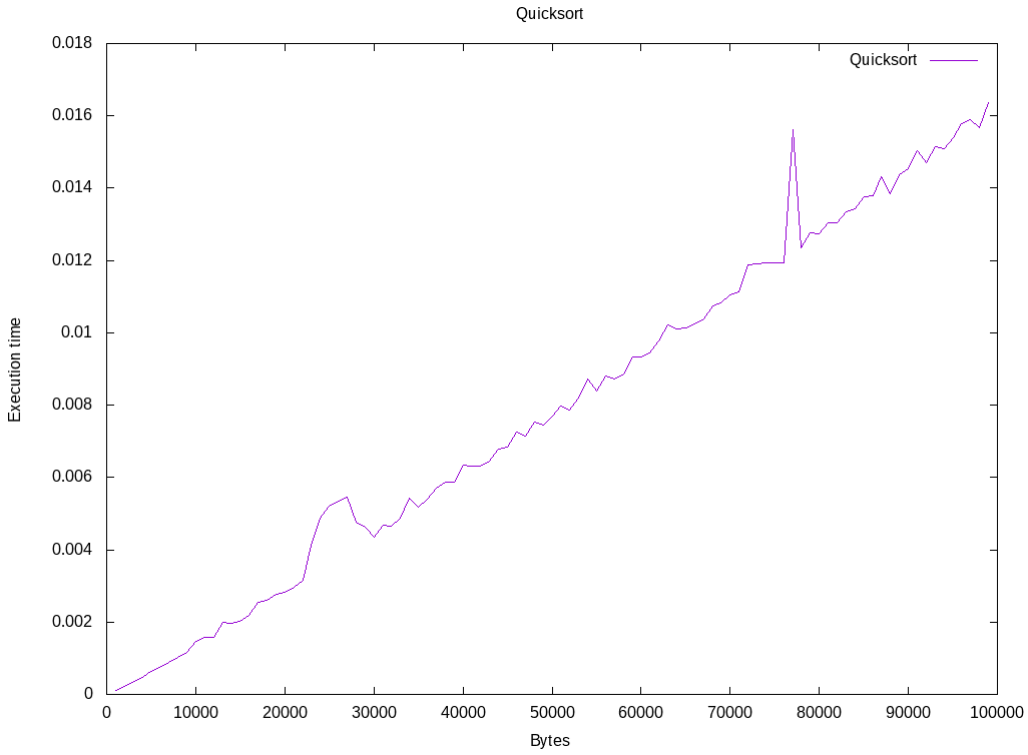

Figura 3.2: Algoritmo Quicksort, gráfica empírica.

La gráfica híbrida obtenida ha sido:

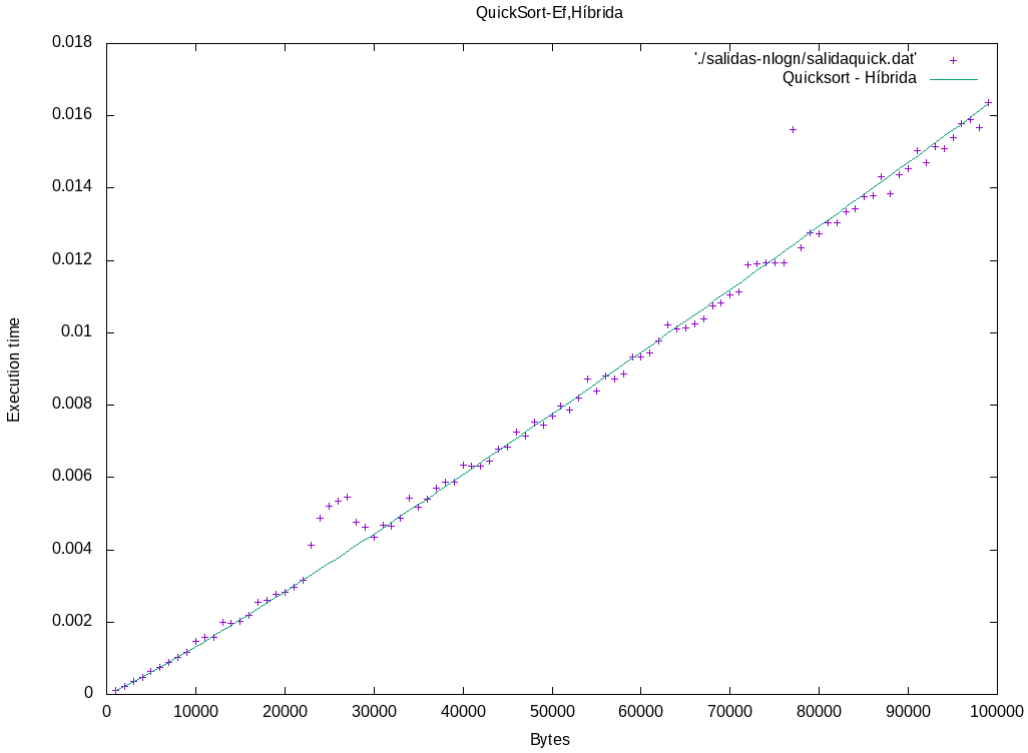

Figura 3.3: Gráfica híbrida Quicksort.

3.2. Heap Sort

La gráfica empírica obtenida ha sido:

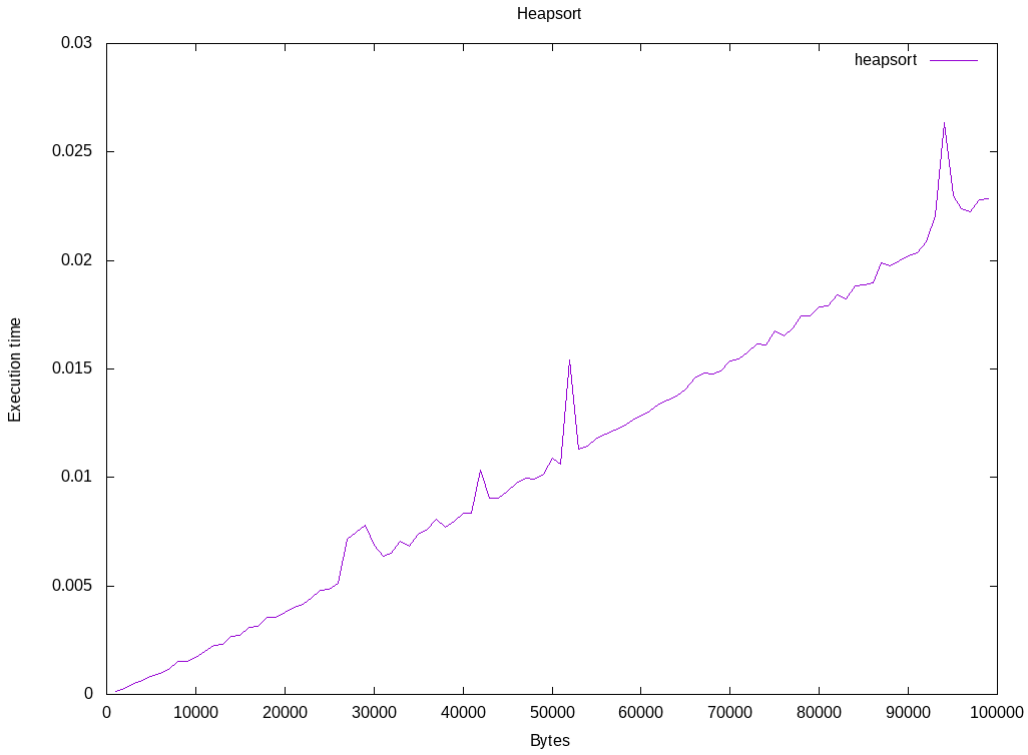

Figura 3.4: Heapsort, gráfica empírica.

La gráfica híbrida obtenida ha sido:

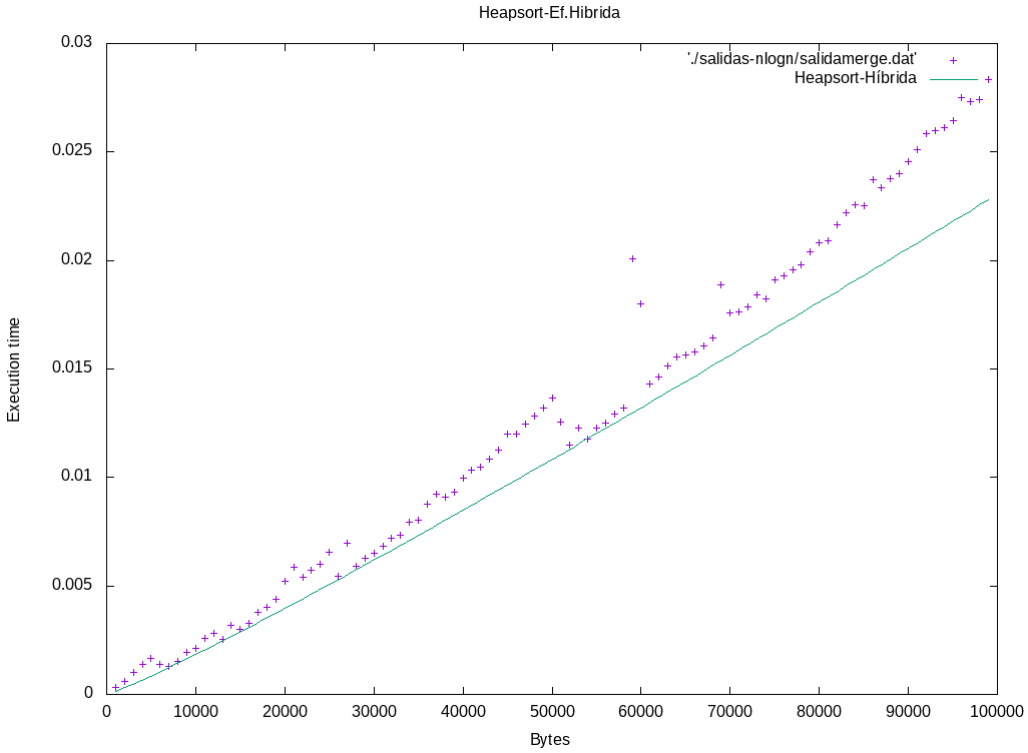

Figura 3.5: Heapsort, gráfica híbrida.

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

El ajuste de ambas gráficas es el siguiente:

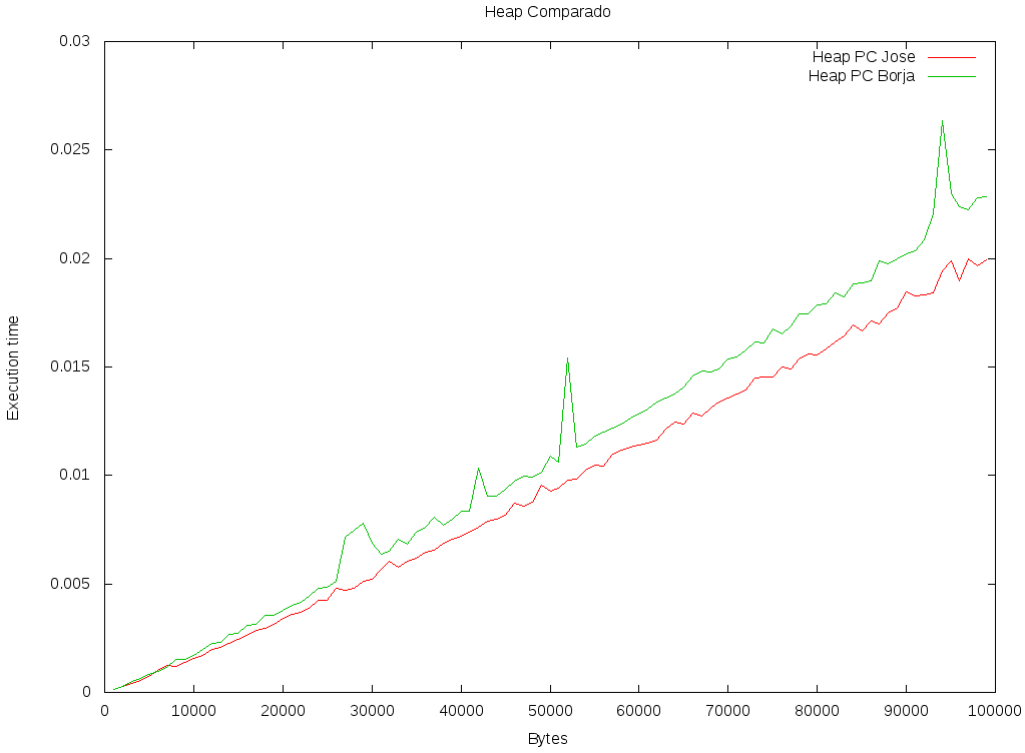

Figura 3.6: Comparación en dos máquinas. Heapsort.

3.3. Merge Sort

La gráfica empírica obtenida ha sido:

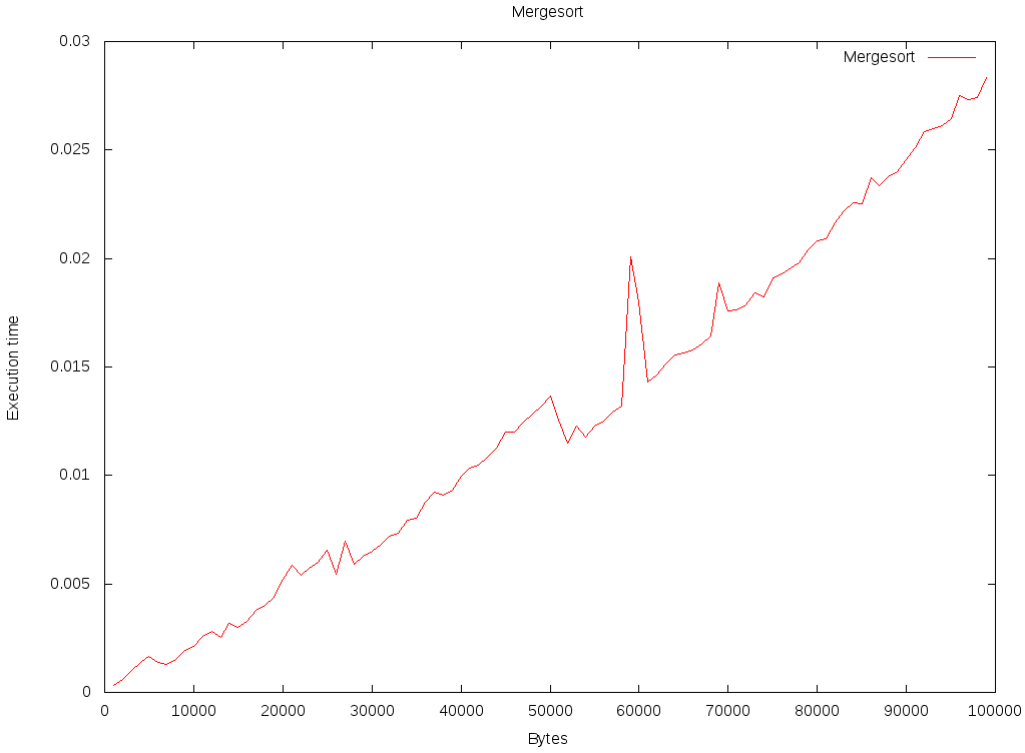

Figura 3.7: Mergesort, gráfica empírica.

La gráfica híbrida obtenida ha sido:

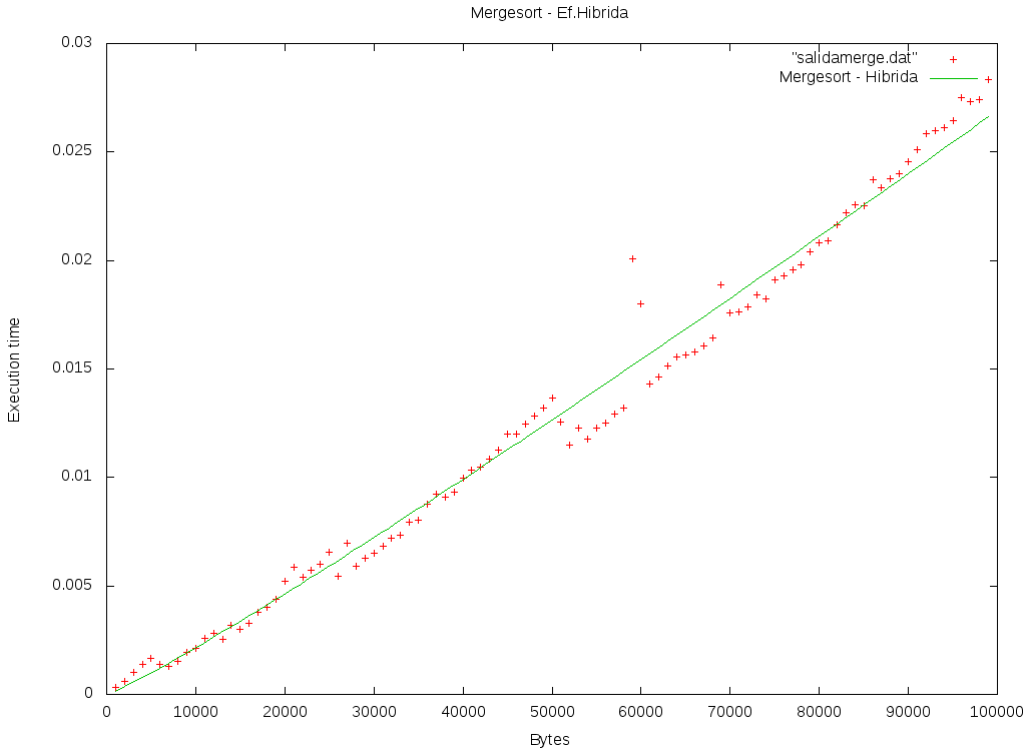

Figura 3.8: Mergesort, gráfica híbrida.

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 %)

4. 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.

4.1. Gráficas

Gráfica Empírica

La gráfica empírica obtenida ha sido:

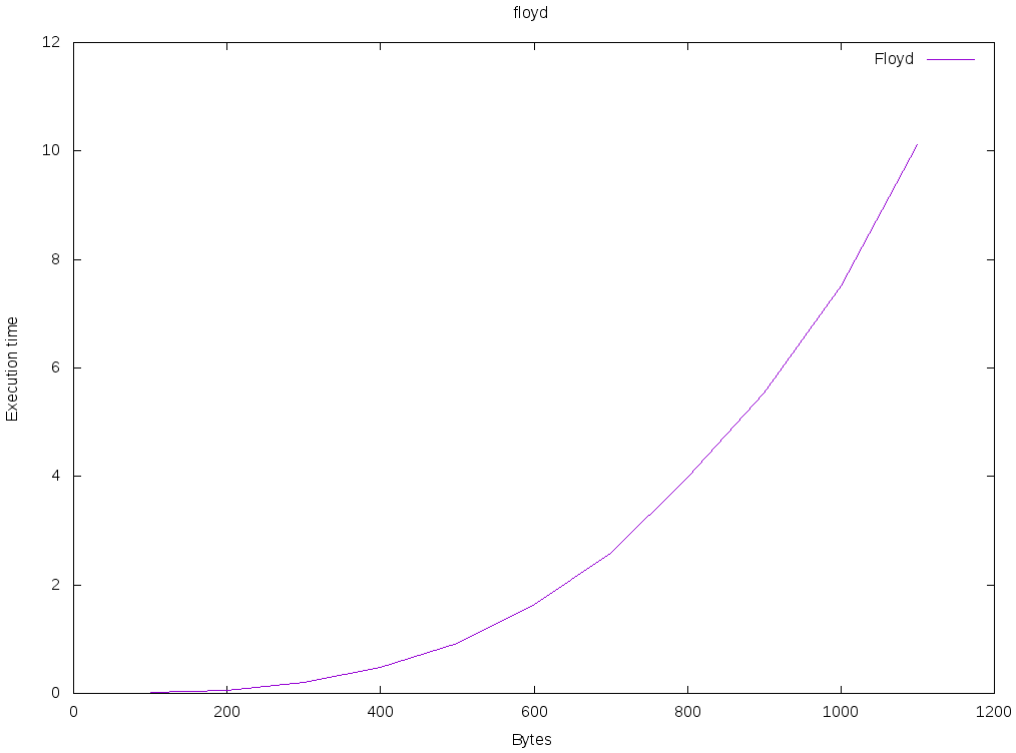

Figura 4.1: Floyd, gráfica empírica.

Gráfica Híbrida

La gráfica híbrida obtenida ha sido:

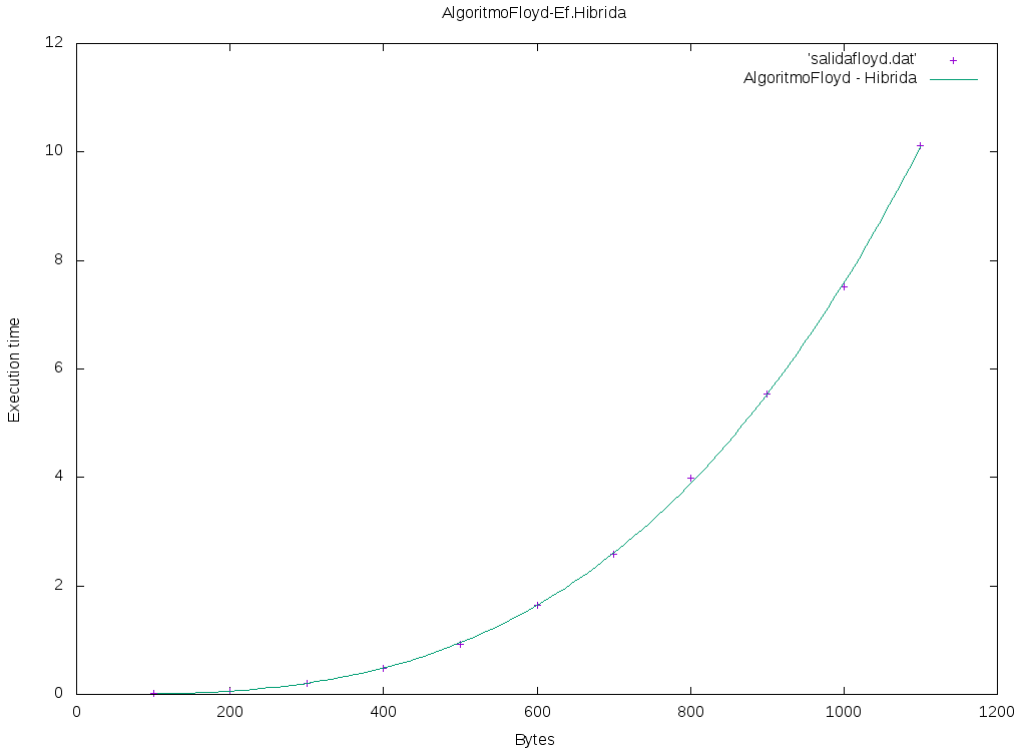

Figura 4.2: Floyd, gráfica híbrida.

Ajuste Híbrido Erróneo

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

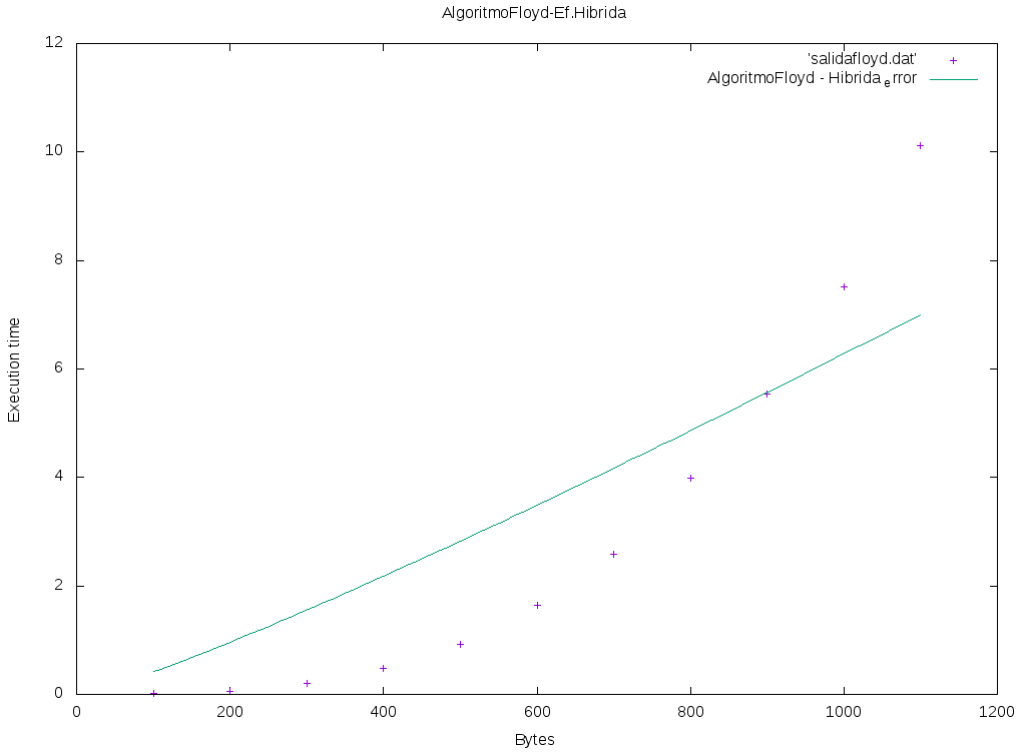

Figura 4.3: Floyd, ajuste erróneo.

4.2. Porcentaje de error y constantes ocultas

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 %)

5. 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.

5.1. Gráficas

La gráfica empírica obtenida ha sido:

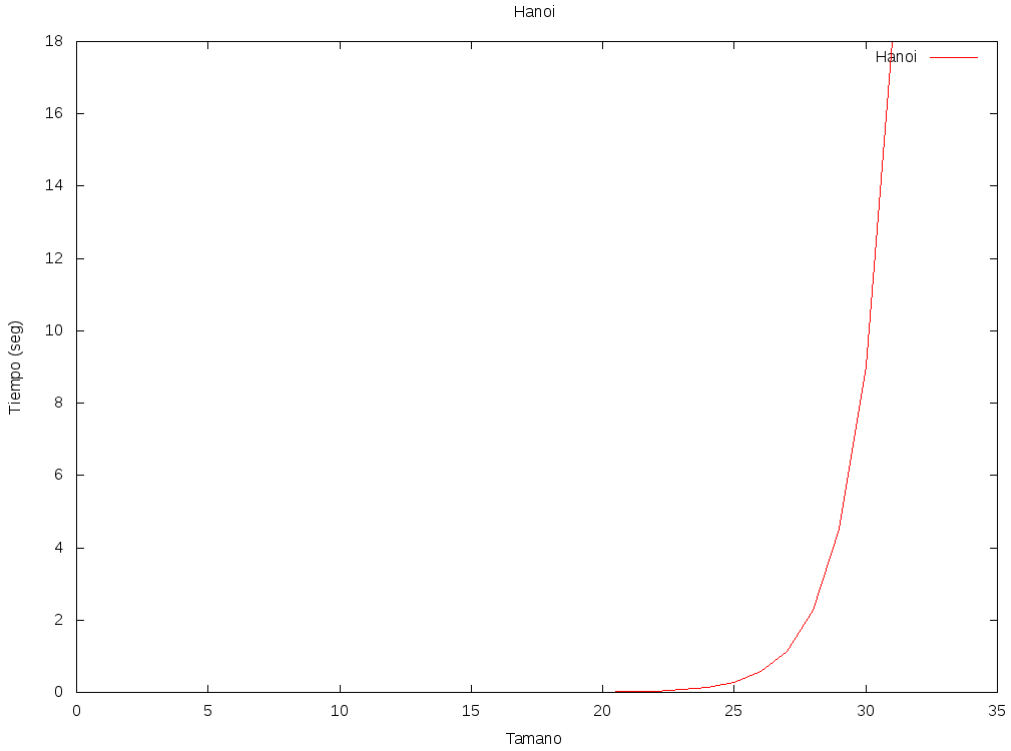

Figura 5.1: Hanoi, gráfica empírica.

La gráfica híbrida obtenida ha sido:

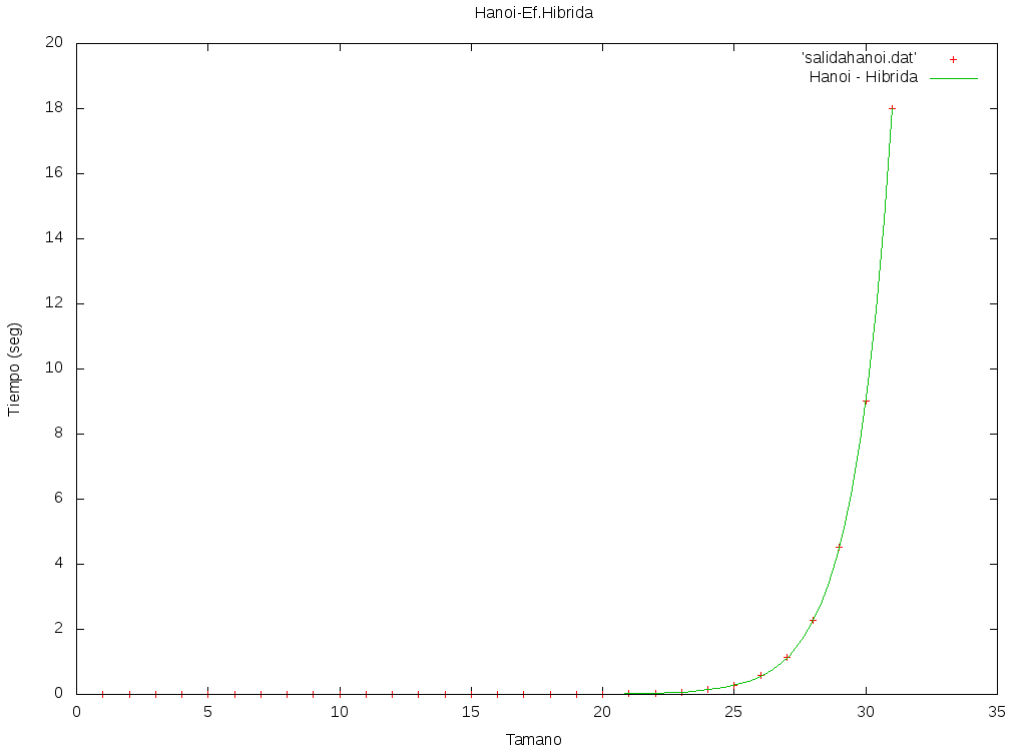

Figura 5.2: Hanoi, gráfica híbrida.

5.2. Comparaciones

En este algoritmo hemos hecho varias comparaciones:

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

5.3. Hanoi en Python

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

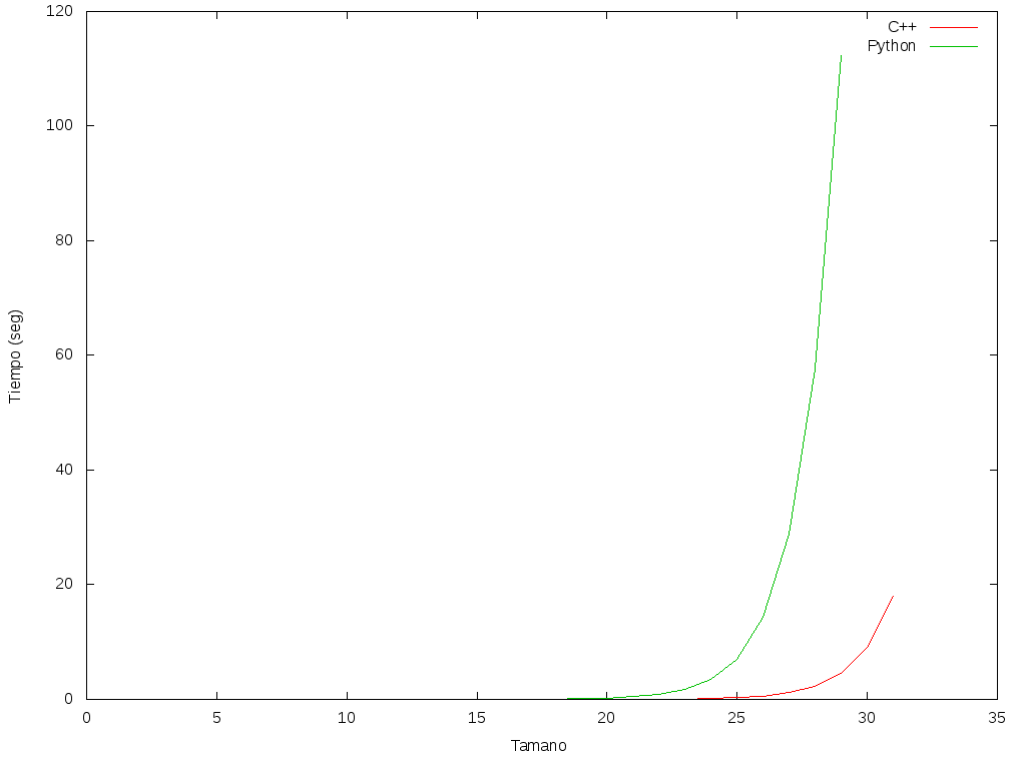

Figura 5.3: Hanoi, Python - C++

5.4. Hanoi con diferentes optimizaciones

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

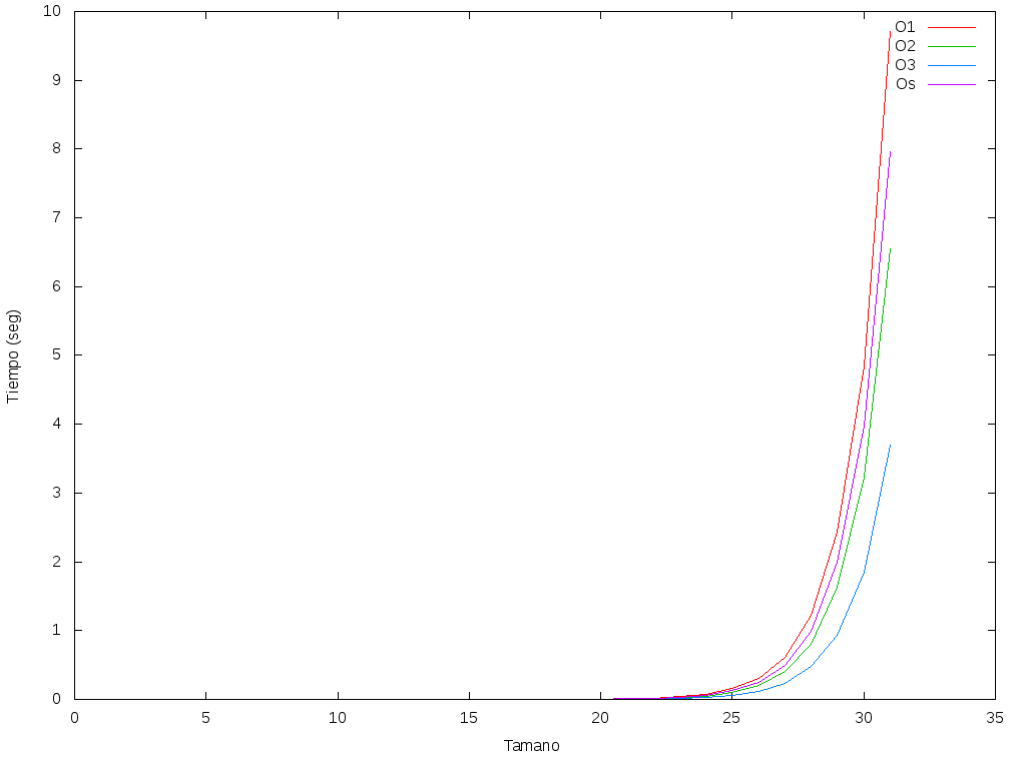

Figura 5.4: Hanoi, gráfica con diferentes eficiencias.

5.5. Porcentaje de error y constantes ocultas

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 %)