Algorítmica I (2018-2019)
Doble Grado en Ingeniería Informática Y Matemáticas
Universidad de Granada

Memoria Práctica 1

Javier Alcántara García

11 de marzo de 2019

Índice

Índice de figuras

Índice de tablas

1. Eficiencia teórica

1.1. Algoritmo 1

Vamos a obtener la eficiencia teórica del Algoritmo 1. Para ello vamos a considerar el siguiente código que implementa la ordenación de un vector desde la posición inicial a la final de este.

```
1  int pivotar(double *v, const int ini, const int fin) {
2      double pivote=v[ini], aux;
3      int i=ini+1, j=fin;
4
5      while (i<=j) {
6          while (v[i]<pivote && i<=j) i++;
7          while (v[j]>=pivote && j>=i) j--;
8
9          if(i<j)
10             aux=v[i]; v[i]=v[j]; v[j]=aux;
11     }
12
13     if(j>ini) {
14         v[ini]=v[j];
15         v[j]=pivote;
16     }
17     return j;
18 }
```

1.2. Algoritmo 2

1.3. Algoritmo 3

2. Eficiencia empírica

Siguiendo las indicaciones dadas en el guión de prácticas, he calculado la eficiencia empírica de los 3 algoritmos dados. Para ello, he usado la herramienta Gnuplot además de la librería chrono.
La herramienta Gnuplot es bien sencilla. Permite mostrar archivos de datos (recogidos previamente usando chrono) con un simple comando: *plot 'datos.dat' title 'Gráfica Sencilla' with lines*. Esto me devolvería una gráfica basada en los datos del archivo *datos.dat*, titulado *Gráfica Sencilla* y siguiendo una línea. También se puede elegir una gráfica de discontinuidad por puntos cambiando *lines* por *points*.
Por otro lado, la librería chrono me permite declarar variables de tipo *high_resolution_clock::now()* para poder calcular el tiempo que un algoritmo tarda en ejecutarse. Este proceso es bien sencillo. Basta con medir el tiempo de reloj antes y después de la ejecución de dicho algoritmo.

2.1. Cálculo Algoritmo 1

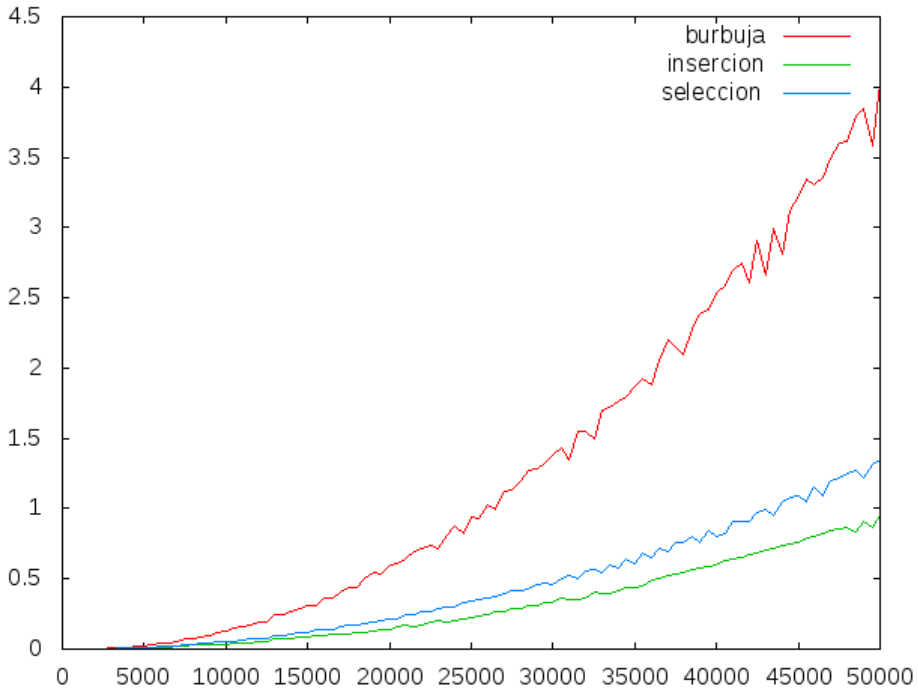

Figura 2.1: Gráfica del Algoritmo 1

3. Comparación gráfica

Generando los gráficos correspondientes (a partir de las tablas anteriores) haciendo uso de la herramienta Gnuplot, obtengo varias comparaciones entre los distintos algoritmos. El proceso para generar dichas tablas es bien sencillo y ya se ha explicado anteriormente. Teniendo las tablas de *datos.dat*, basta con usar el comando *plot 'datos.dat' title 'Gráfica Sencilla' with lines* dentro de la plataforma Genuplot en la terminal. Así, obtengo los siguientes resultados:

3.1. Comparación O(n^2)

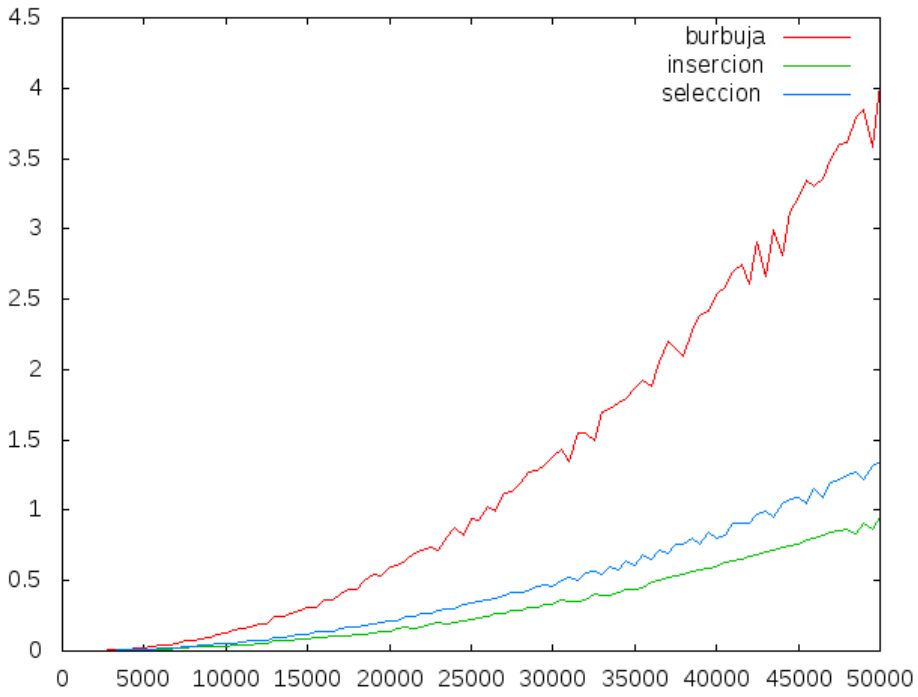

Figura 3.1: Gráfica de algoritmos con eficiencia O(n^2)

Aquí ya no hay duda alguna sobre qué algoritmo cuadrático es más eficiente. Además, queda claro que el método de burbuja es el peor con una enorme diferencia. Este método no se debe usar, si se conoce alguno de los otros dos.

3.2. Comparación O($n * log(n)$)

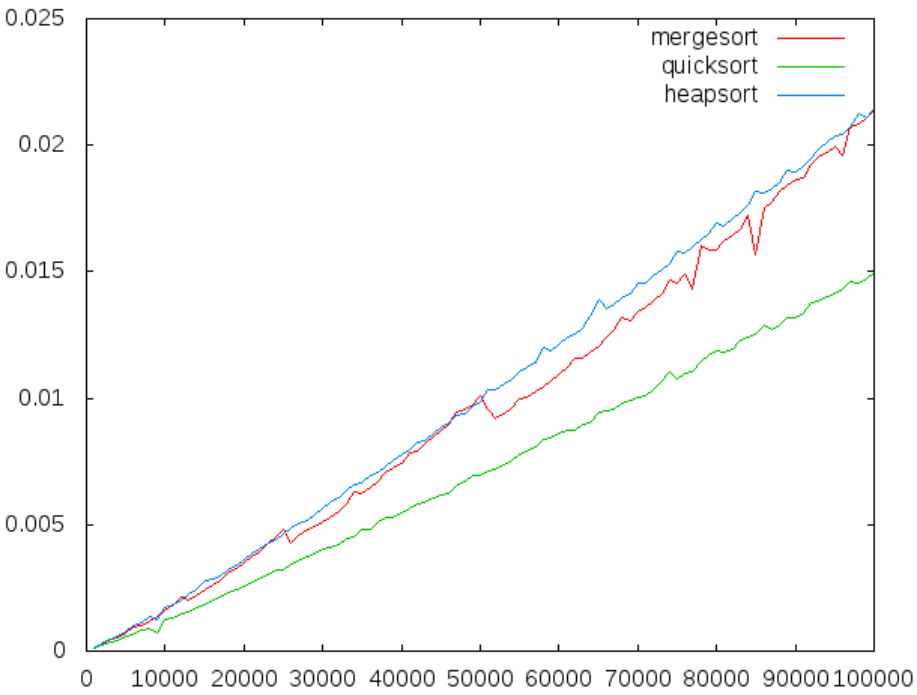

Figura 3.2: Gráfica de algoritmos con eficiencia O($n * log(n)$)

En este caso, la gráfica clarifica que *Mergesort* y *Heapsort* presentan una eficiencia prácticamente igual. Sin embargo, aquí puedo aclarar que *Mergesort* es levemente más rápido. Y, como ya decía, *Quicksort* es el más rápida de los tres.

3.3. Comparación distintos métodos de ordenación

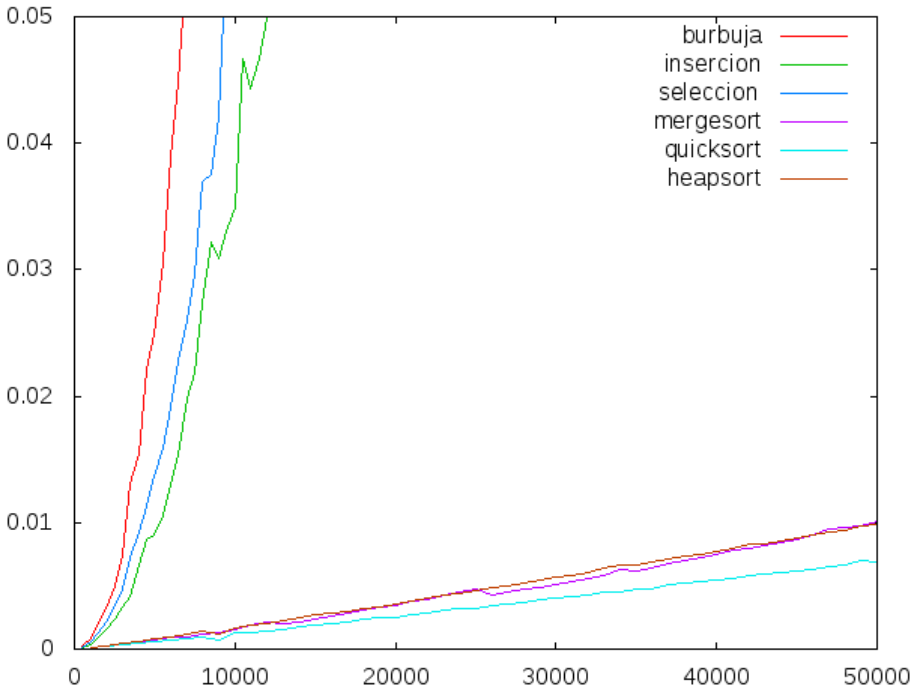

Figura 3.3: Gráfica de algoritmos con eficiencia O(n^2) y O($n * log(n)$)

Como ya era de esperar, la eficiencia entre los tres primeros métodos (aquellos con eficiencia O(n^2)) son bastante más lentos que los tres últimos (aquellos con eficiencia O($n * log(n)$)). Es inmediato darse cuenta de que una función cuadrática crece extremadamente más rápido que la logarítmica*n. Tanto que he tenido que ajustar los límites de la gráfica para poder apreciar las funciones logarítmicas junto a las cuadráticas.

3.4. Análisis O(n^3)

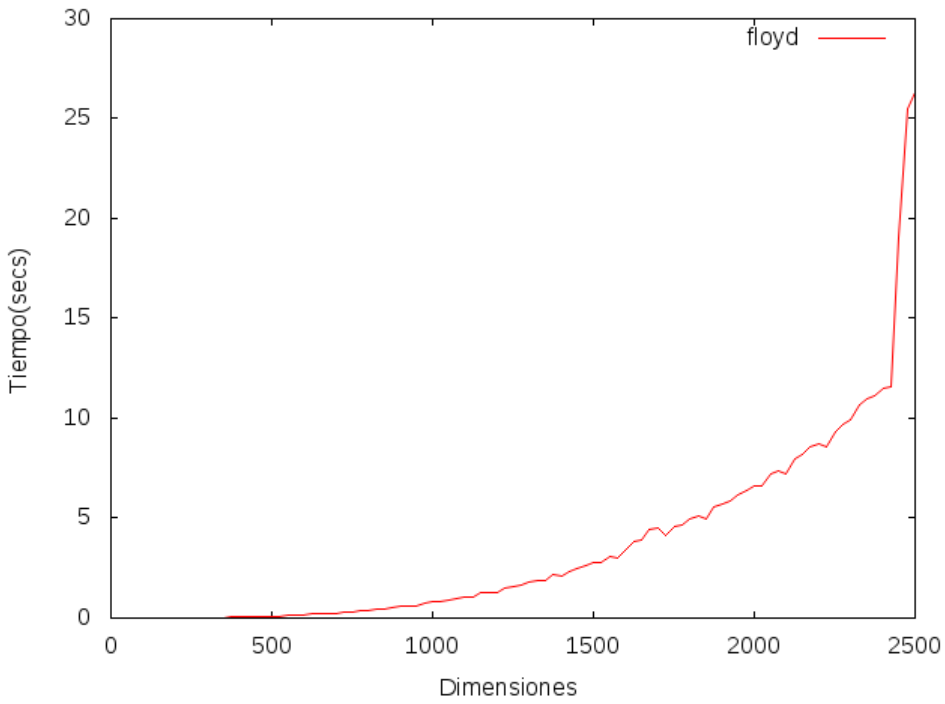

Figura 3.4: Gráfica de algoritmo con eficiencia O(n^3)

Tanto en este caso como en el siguiente, la comparación se resume en un análisis de un solo algoritmo. El algoritmo de *Floyd* es el único algoritmo de orden cúbico en esta práctica. Cabe destacar que para valores tan bajos como 2500 el tiempo ya roza los 30 segundos. Por encima de ahí, el crecimiento es exponencial, tal que mi PC se quedaba congelado.

3.5. Análisis O(2^n)

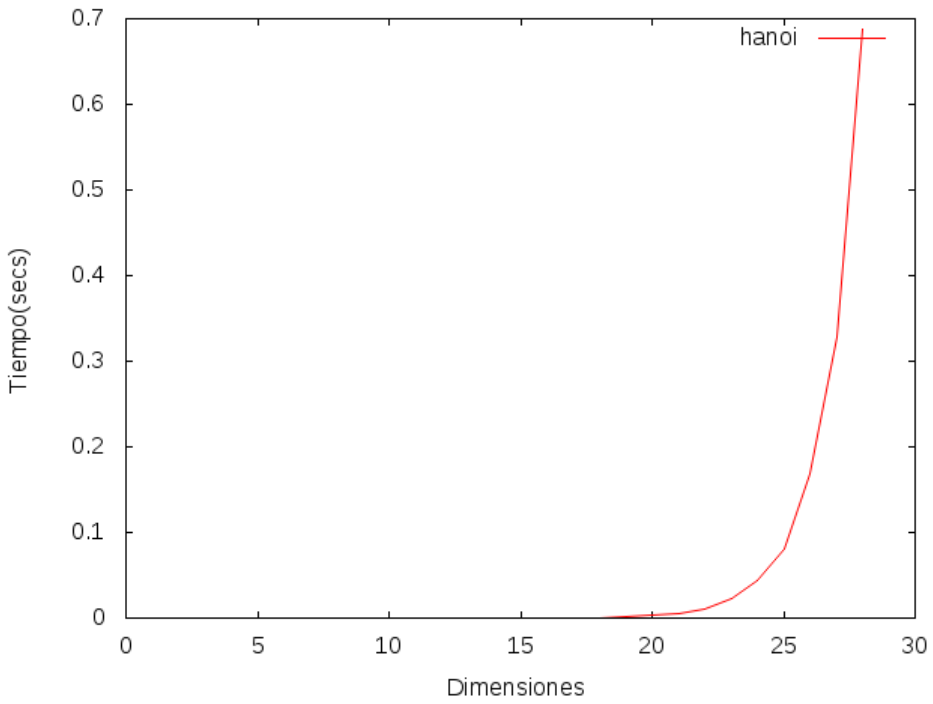

Figura 3.5: Gráfica de algoritmo con eficiencia O(2^n)

En este ejemplo, las distintas dimensiones no pasan de 30 elementos. Siendo el algoritmo de *Hanoi* una función potencial, el crecimiento de la función es exponencial. Por esto, muy rápidamente el PC empezaba a calentarse en exceso.

4. Eficiencia híbrida

Para el cálculo de la eficiencia híbrida, he de tener en cuenta tanto la teórica como la empírica. Teniendo la empírica ya calculada, es sencillo deducir la expresión inicial de su eficiencia teórica. (e.g. eficiencia cuadrática: $T(n) = a0 * x^2 + a1 * x + a2$). Siguiendo este ejemplo, haciendo uso de Genuplot calculo las constantes ocultas para cada función T(n) en función de cada tabla de datos. Esto se hace mediante el comando *fit f(x) 'datos.dat' via a0,a1,a2.*
Después, ajusto la función resultante a su correspondiente tabla de datos. Usando el comando *plot 'datos.dat', f(x) title 'Curva Ajustada'*, se obtiene el resultado pedido(en el caso de O(n^2) y O($n * log(n)$), los tres casos son idénticos respectivamente. Por eso, muestro un algoritmo para cada uno de ellos en cada caso. Los otros dos resultantes son exactamente igual. Siguiendo el mismo proceso ya explicado para cada tabla de datos):

4.1. Cálculo O(n^2)

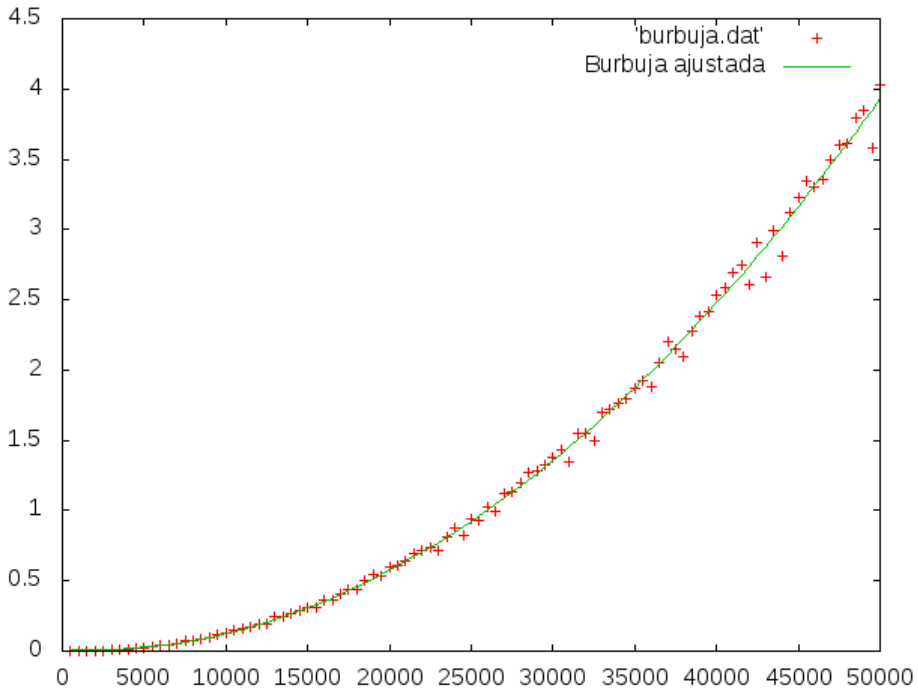

Figura 4.1: Gráfica híbrida del algoritmo de Burbuja

Como se puede observar, la función se ajusta casi perfectamente a nuestros datos. Esto mismo ocurre para los otros dos métodos de ordenación con eficiencia O(n^2).

4.2. Cálculo O($n * log(n)$)

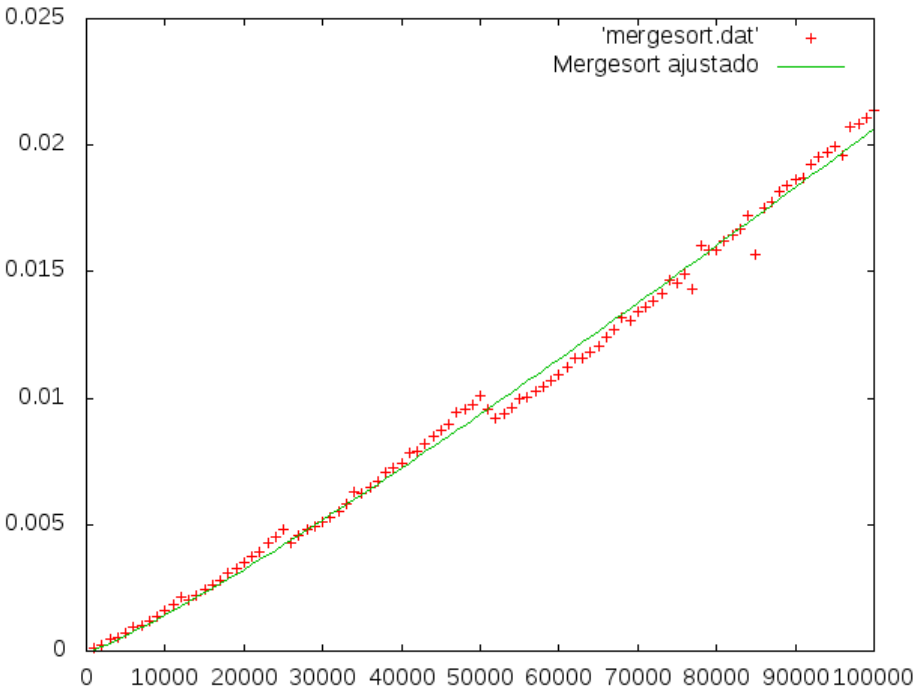

Figura 4.2: Gráfica híbrida del algoritmo de Mergesort

Nuevamente, se tiene un fácil ajuste de la función con respecto a los datos que se obtuvieron anteriormente. Un proceso sencillo y rutinario. Este mismo proceso otorga equidad de resultados con los algoritmos de *Quicksort* y *Heapsort*.

4.3. Cálculo O(n^3)

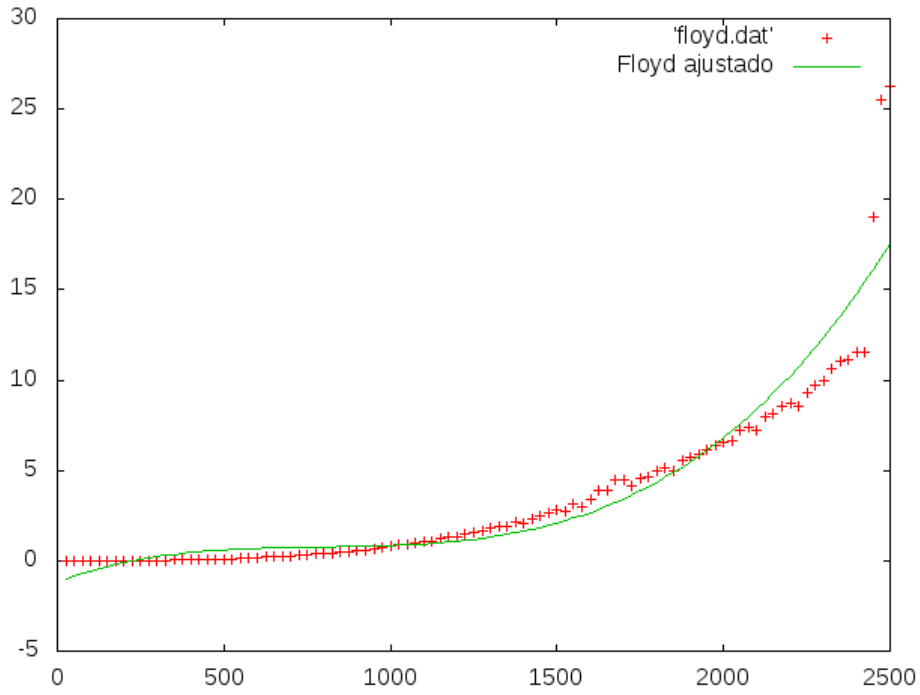

Figura 4.3: Gráfica híbrida del algoritmo de Floyd

En este caso, llegando a la dimensión 2500, la gráfica de datos tiene un salto exponencial casi anormal. Sin embargo, la función teórica sigue una curva más suave, propia de una función cúbica. Aún así, el ajuste sigue siendo prácticamente ideal.

4.4. Cálculo O(2^n)

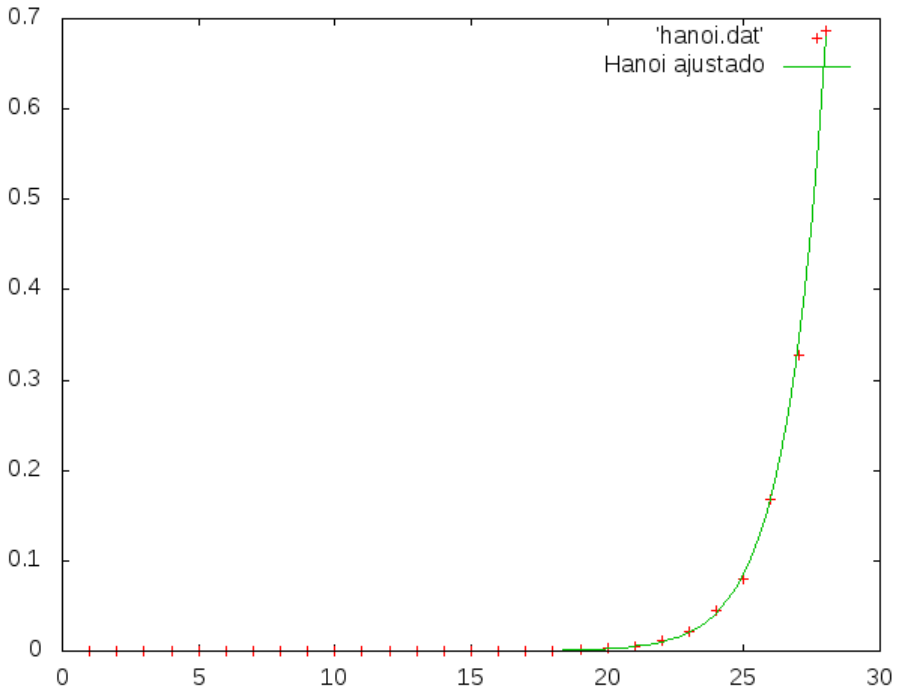

Figura 4.4: Gráfica híbrida del algoritmo de Hanoi

Ahora el ajuste si se puede decir perfecto con respecto al montón de datos previamente recogidos. En este último caso he tomado puntos de uno en uno hasta el valor 30. Por esto es que el ajuste resulta más sencillo pues la exactitud de la gráfica que definía los datos era más bien pobre, con respecto a las otras.

5. Variación por factores externos

Haciendo uso de dos máquinas totalmente distintas, he podido comprobar que esto no afecta al análisis de eficiencia de un algoritmo. Haciendo uso de dos máquinas con *Ubuntu* y *MacOS* he obtenido resultados prácticamente iguales.
Cierto es que los algoritmos corrían en menor tiempo en la máquina de Apple, pero, la eficiencia en sí era la misma. Aunque para iguales datos el tiempo fuese menor, seguía representado el mismo orden de eficiencia. Es por esto que usando no solo dispositivos distintos pero sistemas operativos diferentes, la eficiencia se mantiene.
Por esto es que, el análisis de eficiencia se puede realizar desde cualquier dispositivo sin pérdida de generalidad. Lo que no se debe hacer es comparar distintos algoritmos ejecutados en distintas máquinas, obviamente. Pero, ejecutados en la misma, siempre se obtiene el mismo orden de eficiencia. Por ello, concluyo que el análisis de eficiencia empírica no varía en función de parámetros externos. Es la propia máquina la que hará que TODOS los algoritmos se ejecuten más o menos rápido. Al igual que el sistema operativo. Pero, todos en relación cumplen la misma eficiencia.