MANUAL de EFICIENCIA

5 de diciembre de 2011

EFICIENCIA

Objetivo

El objetivo de estas sesiones es que el alumno/a comprenda la importancia de analizar la eficiencia de los algoritmos y se familiarice con las formas de llevarlo a cabo. Para ello se mostrará como realizar un estudio teórico y empírico de un algoritmo.

Cálculo del tiempo teórico

A partir de la expresión del algoritmo, se aplicarán las reglas conocidas para contar el número de operaciones que realiza un algoritmo. Este valor será expresado como una función de $T(n)$ que dará el número de operaciones requeridas para un caso concreto del problema caracterizado por tener un tamaño n. En los casos de algoritmos recursivos aparecera una expresión del tiempo de ejecución con forma recursiva, que habrá de resolver con las técnicas estudiadas (ej: resolución de recurrencias por el método de la ecuación característica).

El análisis que nos interesa será el del peor caso. Así, tras obtener la expresión analítica de $T(n)$, calcularemos el orden de eficiencia del algoritmo empleando la notación $O(\cdot)$.

Ejemplo 1: Algoritmo de Ordenación Burbuja

Vamos a obtener la eficiencia teórica del algoritmo de ordenación burbuja. Para ello vamos a considerar el siguiente código que implementa la ordenación de

un vector de enteros, desde la posición inicial a final de este, mediante el método burbuja.

```
1.  void burbuja(int T[], int inicial, int final)
2.  {
3.  int i, j;
4.  int aux;
5.  for (i = inicial; i < final - 1; i+ +)
6.   for (j = final - 1; j > i; j- -)
7.    if (T[j] < T[j-1])
8.    {
9.    aux = T[j];
10.   T[j] = T[j-1];
11.   T[j-1] = aux;
12.   }
13. }
```

La mayor parte del tiempo de ejecución se emplea en el cuerpo del bucle interno. Esta porción de código lo podemos acotar por una constante *a*. Por lo tanto las lineas de 7-12 se ejecutan exactamente un número de veces de final-i-1. A su vez el bucle interno se ejecuta una serie de veces indicado por el bucle externo. En definitiva tendremos una fórmula como la siguiente:

$$\sum_{i=inicial}^{final-2} \sum_{j=i+1}^{final-1} a \qquad (1)$$

Renombrando en la ecuación (1) $final$ como *n* e *inicial* como 1, pasemos a resolver la siguiente ecuación:

$$\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} a \tag{2}$$

Realizando la sumatoria interior en (2) obtenemos:

$$\sum_{i=1}^{n-2} a(n-i-1) \tag{3}$$

Y finalmente tenemos:

$$\frac{a}{2}n^2 - \frac{3a}{2}n + a \tag{4}$$

Claramente $\frac{a}{2}n^2 - \frac{3a}{2}n + a \in O(n^2)$

Diremos por tanto que el método de ordenación es de orden $O(n^2)$ o cuadrático.

Ejemplo 2: Algoritmo de Ordenación Mergesort

En este ejemplo vamos a calcular la eficiencia del algoritmo de ordenación Mergesort que será aplicado siempre y cuando tengamos un número de enteros en el vector (a ser ordenado) mayor de UMBRAL_MS, en otro caso se aplicará el algoritmo de ordenación burbuja.

```
1. void burbuja(int T[], int inicial, int final)
2. {
3.  int i, j;
4.  int aux;
5.  for (i = inicial; i < final - 1; i++)
6.          for (j = final - 1; j > i; j- -)
7.          if (T[j] < T[j-1])
8.           {
```

```
                aux = T[j];
                T[j] = T[j-1];
                T[j-1] = aux;
            }
    }
    void fusion(int T[], int inicial, int final, int U[], int V[])
    {
     int j = 0;
     int k = 0;
     for (int i = inicial; i < final; i++)
     {
            if (U[j] < V[k]) {
                T[i] = U[j];
                j++;
             } else{
                T[i] = V[k];
                k++;
            };
     };
    }
    void mergesort(int T[], int inicial, int final)
    {
```

```
31.  if (final - inicial < UMBRAL_MS)
32.  {
33.   burbuja(T, inicial, final);
34.  } else {
35.       int k = (final - inicial)/2;
36.       int * U = new int [k - inicial + 1];
37.
38.       int l, l2;
39.       for (l = 0, l2 = inicial; l < k; l++, l2++)
40.            U[l] = T[l2];
41.       U[l] = INT_MAX;
42.       int * V = new int [final - k + 1];
43.
44.       for (l = 0, l2 = k; l < final - k; l++, l2++)
45.            V[l] = T[l2];
46.       V[l] = INT_MAX;
47.       mergesort(U, 0, k);
48.       mergesort(V, 0, final - k);
49.       fusion(T, inicial, final, U, V);
50.       delete [] U;
51.       delete [] V;
52.  };
```

53. }

El algoritmo Mergesort lo que hace es dividir en dos partes iguales el vector y aplicar el algoritmo Mergesort recurrente a cada una de estas partes, una vez hecho esto fusionará los dos vectores sobre el vector original de manera que esta parte ya queda ordenada. Como hemos dicho antes si el número de elementos del vector que se esta tratando en cada momento de la recursión es menor que UMBRAL_MS se ordenará mediante el algoritmo burbuja.

Una vez entendido el procedimiento Mergesort vamos a pasar a obtener la eficiencia teórica de este.

Suponiendo que entramos por la rama else (línea 35) podemos acotar la secuencia de instrucciones de las líneas 35-38 por una constante. Sin perder generalidad, vamos a tomar esta constante como c. El bucle **for** de la línea 39 se ejecuta un número de veces igual a $\frac{n}{2}$ (en la primera llamada de mergesort tomaremos inicial como 0 y final como n), por lo tanto la eficiencia hasta aquí sera $O(n)$. Este mismo razonamiento lo tenemos para las instrucciones desde la lnea 41 hasta 46. Aplicando la regla del máximo obtendramos hasta la línea 46 un orden de $O(n)$. A continuación, se hacen dos llamadas de forma recursiva a mergesort (líneas 47 y 48) con los dos nuevos vectores construidos para ordenar en cada uno de ellos $\frac{n}{2}$ elementos. En la línea 49 se llama al procedimiento fusión que como se puede observar tiene una eficiencia de $O(n)$. Por lo tanto para averiguar la eficiencia de mergesort podemos formular la siguiente ecuación:

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + n & si \quad n \geq UMBRAL\_MS \\ n^2 & si \quad n < UMBRAL\_MS \end{cases} \tag{5}$$

Vamos a resolver la recurrencia en (5):

$$T(n) = 2T(\frac{n}{2}) + n \qquad si \qquad n \geq UMBRAL\_MS \tag{6}$$

Haciendo en (6) el cambio de variable $n = 2^m$ obtenemos lo siguiente:

$$T(2^m) = 2T(2^{m-1}) + 2^m \qquad si \qquad m \geq log_2(UMBRAL\_MS) \tag{7}$$

$$T(2^m) - 2T(2^{m-1}) = 2^m \tag{8}$$

notando $T(2^m) = t_m$ obtenemos

$$t_m - 2t_{m-1} = 2^m \tag{9}$$

La ecuación de recurrencia que se deduce de (9) es :

$$(x-2)^2 = 0 \tag{10}$$

Por lo tanto, la solución general será:

$$t_m = c_1 2^m + c_2 m 2^m \tag{11}$$

deshaciendo el cambio de variable que hicimos en (6) obtenemos:

$$t_n = c_1 n + c_2 n log_2 n \tag{12}$$

Por lo tanto $t_n \in O(nlog_2 n)$

En este ejemplo cabe preguntarse por qué hemos hecho uso de un algoritmo de ordenación de orden cuadrático (líneas 31-34), como es la eficiencia del algoritmo burbuja de ordenación, para el caso base cuando el algoritmo Mergesort es mas eficiente. La respuesta se verá cuando os expongan la técnica de resolución de problemas “Divide y Vencerás”. Aquí solamente deciros que el algoritmo de ordenación burbuja a pesar "de ser cuadrático" es lo suficientemente eficiente para aplicarlo sobre un vector con pocos elementos, como se da en este caso con un valor de UMBRAL_MS lo suficientemente pequeño. También apreciareis que los algoritmos recursivos a igual orden de eficiencia son menos eficientes que los algoritmos iterativos (hacen uso de llamadas recursivas donde internamente se debe gestionar una Pila donde guardar los valores de las variables en cada recursiósn).

Cálculo de la eficiencia empírica

Se trata de llevar a cabo un estudio puramente empírico. Es decir, estudiar experimentalmente el comportamiento del algoritmo. Para ello mediremos los recursos empleados (tiempo) para cada tamaño dado de las entradas.

En el caso de los algoritmos de ordenación el tamaño viene dado por el número

de componentes del vector a ordenar. En otro tipo de problemas, como es el caso del algoritmo para obtener el factorial de un número o la sucesión de fibonacci, la eficiencia depender del valor del entero.

Para obtener el tiempo empírico de una ejecución de un algoritmo lo que vamos a hacer es definir en el código dos variables como las siguientes:

clock_t tantes;

clock_t tdespues;

De esta forma, en la variable ***tantes*** capturamos el valor del reloj antes de la ejecución del algoritmo al que queremos medir el tiempo. La variable $tdespues$ contendrá el valor del reloj despues de la ejecución del algoritmo en cuestión.

Así, si deseamos obtener el tiempo del algoritmo de ordenación burbuja tendremos que poner algo parecido a lo siguiente:

$tantes = clock()$; *//Captura el valor del reloj antes de la llamada a burbuja*
$burbuja(T,0,n)$;*// Llama al algoritmo de ordenación burbuja*
$tdespues = clock()$;*//Captura el valor del reloj despues de la ejecución de burbuja*

Para obtener el número de segundos simplemente emitiremos un mensaje como este:
$cout << (double)(tdespues - tantes)/CLOCKS\_PER\_SEC << endl$;

Con esta instrucción lo que hacemos es obtener la diferencia entre los dos instantes y pasarlo a segundos mediante la constante *CLOCKS_PER_SEC*. Para hacer uso de estas sentencias teneis que usar la biblioteca *ctime*.

OBSERVACION: Para casos muy pequeños, el tiempo medido es muy pequeño, por lo que el resultado será de 0 segundos. Estos tiempos tan pequeños se pueden medir de forma indirecta ejecutando la sentencia que nos interesa muchas veces y despues dividiendo el tiempo total por el número de veces que se ha ejecutado. Por ejemplo:

```
#include <ctime>
...


clock_t tantes,tdespues;
double tiempo_transcurrido;
const int NUM_VECES=10000;
int i;
tantes=clock();
for (i=0; i<NUM_VECES;i++)
//Sentencia cuyo tiempo se pretende medir
tdespues = clock();
tiempo_transcurrido=((double)(tdespues-tantes)/(CLOCKS_PER_SEC*
(double)NUM_VECES));
```

Para obtener la eficiencia empírica deberemos de ejecutar el mismo algoritmo para diferentes ejemplos. Así para un algoritmo de ordenación lo ejecutaremos para diferentes tamaños del vector a ordenar y obtendremos el tiempo. Estos tiempos los almacenaremos en un fichero.

Podeis hacer uso de una macro, como la que se muestra a continuación, para obtener el tiempo empirico de un algoritmo.

```
#!/bin/csh -vx
@ i = 10
echo ""> burbuja.dat
while ( $i < 10000 )
echo " $i ‘burbuja $i‘" >> burbuja.dat
@ i += 100
end
```

Así en esta macro se va a escribir en el fichero ***burbuja.dat*** el tiempo en segundos que tarda el algoritmo de ordenación en ordenar vectores de 10 a 10000

elementos. Las muestras se han tomado de 100 en 100. Para poder ejecutar esta macro debeis darle a esta permisos de ejecución. Por ejemplo, mediante la siguiente sentencia:

chmod +x macro

y a continuación ejecutar la macro como

./macro

NOTA: Cuando redactais un informe sobre la eficiencia empírica de un algoritmo deberán aparecer todos los detalles relevantes al proceso de medida: tipo de ordenador utilizado, compilador empleado, opciones de compilación, etc.

Como mostrar los datos.

Para mostrar la eficiencia empírica haremos uso de tablas que recojan el tiempo invertido para cada caso o bien podemos hacer uso de gráficas.

Para mostrar los datos en gráfica pondremos en el eje X el tamaño de los casos y en el eje Y el tiempo, medido en segundos, requerido por la implementación del algoritmo.

Para hacer esta representación de los datos podemos hacer uso del software grace (xmgrace). Este software trabaja sobre LINUX aunque ha sido exportado a plataformas como Windows 2000/NT/XP. Podeis encontrarlo en http://plasma-gate.weizmann.ac.il/Grace/.

XMGRACE.-

El programa se inicia mediante xmgrace. Al ejecutar el programa os debe aparecer una ventana como la que se muestra en la figura 1. (Todas las imágenes que se muestran a continuación fueron obtenidas con la versión de xmgrace 5.1).

A continuación debemos introducir el fichero donde tenemos almacenados los datos empíricos. Esto se puede ver como se realiza en las figuras 2 y 3.

La gráfica se podra representar como se muestra en la figura 4.

Para salvar el plot debemos irnos a File>Print Setup como se muestra en la figura 5. A continuación le damos el nombre al fichero de salida y el formato de

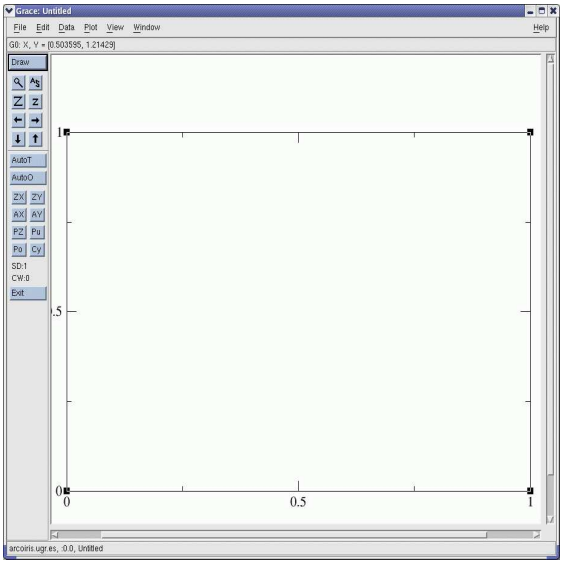

Figura 1: Inicio de xmgrace

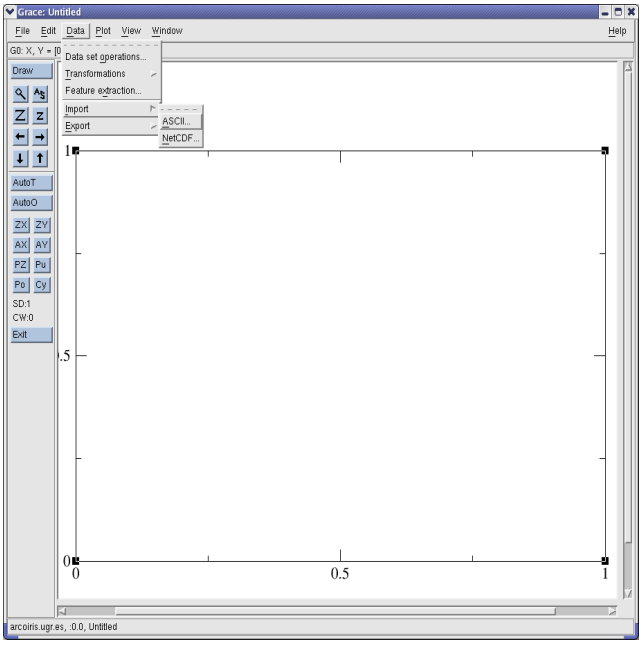

Figura 2: Abrir un fichero ASCII

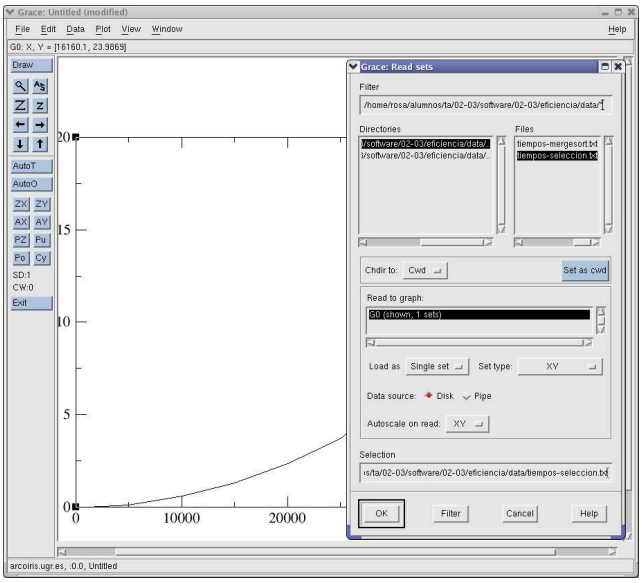

Figura 3: Seleccionar y representar gráficamente los datos

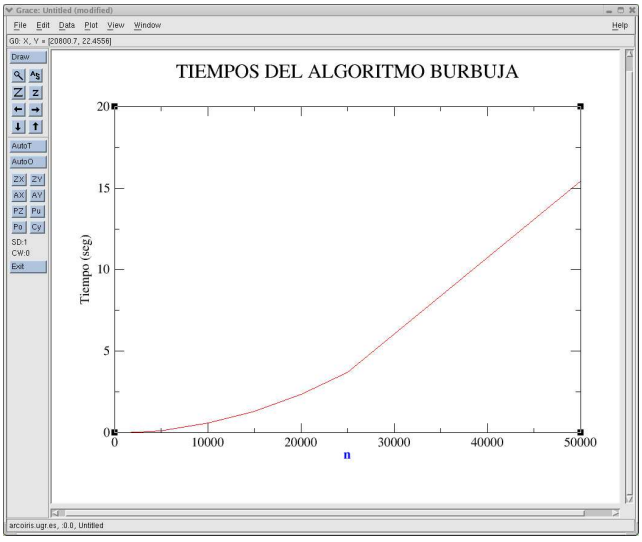

Figura 4: Gráfica con los tiempo del algoritmo Burbuja

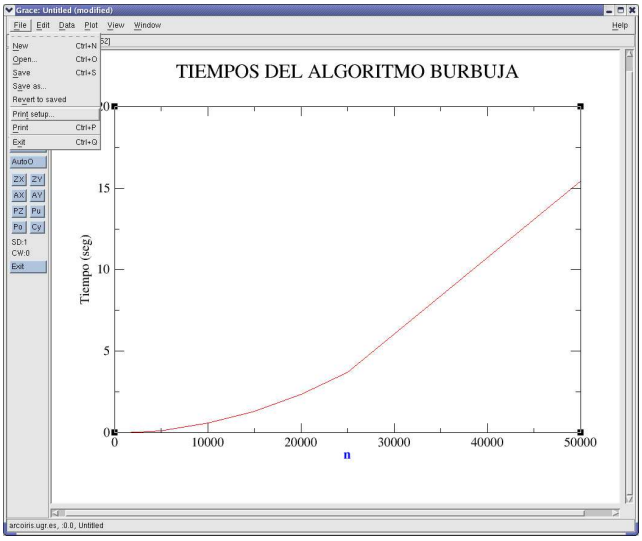

Figura 5: Exportar la gráfica

imagen en que queremos salvarlo. En la figura 6 se ha salvado la gráfica sobre el fichero burbuja.jpg en formato JPEG. Finalmente debemos de pulsar FILE>Print

Cálculo de la eficiencia híbrida.

El cálculo teórico del tiempo de ejecución de un algoritmo nos da mucha información. Es suficiente para comparar dos algoritmos cuando los suponemos aplicados a casos de tamaño arbitrariamente grande. Sin embargo, cuando se va a aplicar el algoritmo en una situación concreta, es decir, especificadas la implementación, el compilador utilizado, el ordenador sobre el que se ejecuta, etc., nos interesa conocer de la forma mas exacta posible la ecuación del tiempo.

Así, el cálculo teórico nos da la expresión general, pero asociada a cada término de esta expresión aparece una constante de valor desconocido. Para describir completamente la ecuación del tiempo, necesitamos conocer el valor de esas constantes. La forma de averiguar estos valores es ajustar la función a un conjunto de puntos.

En nuestro caso, la función es la que resulta del cálculo teórico, el conjunto de puntos lo forman los resultados del análisis empírico y para el ajuste emplearemos regresión por mínimos cuadrados. Por ejemplo en el algoritmo de ordenación la

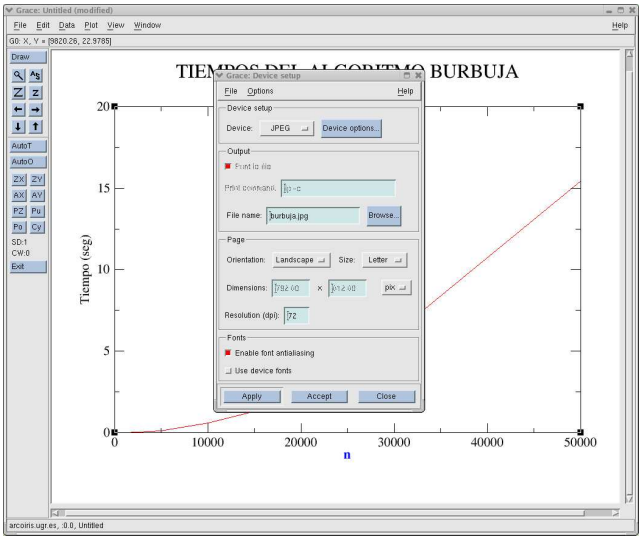

Figura 6: Salvar la gráfica en formato jpeg

función que vamos a ajustar a los puntos obtenidos en el cálculo de la eficiencia empírica sera:

$$T(n) = a_o * x^2 + a_1 * x + a_2$$

Al ajustar a los puntos obtenidos por mínimos cuadrados obtendremos los valores de a_0, a_1y a_2 es decir las constantes ocultas.

Como obtener las constantes ocultas

Para facilitar la labor del ajuste, emplearemos el programa xmgrace. Para ello nos iremos al menu que se muestra en la figura 7.

A continuación introduciremos la forma funcional de $T(n)$ e indicaremos que número de constantes ocultas tiene. En la figura 8 se puede ver que hemos introducido $a0 * x\hat{}2 + a1 * x + a2$ y hemos dado un número de constantes ocultas igual a 3 .

A continuación le damos al botón *Apply y* aparecerá una ventana como la que se muestra en la figura 9. En esta ventana se muestra que valores han adoptado las constantes ocultas además de obtener información de estadísticos como Chi-cuadrado, el coeficiente de correlación, el error cuadrático medio, entre otros. De

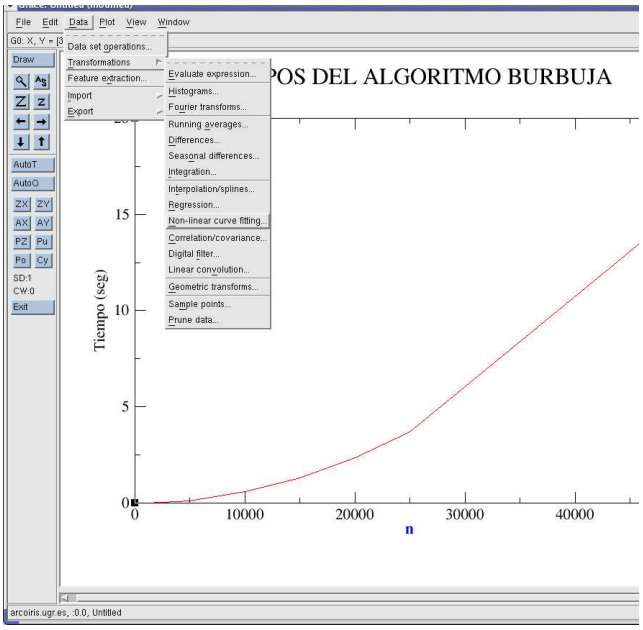

Figura 7: Cómo realizar el ajuste de las constantes ocultas

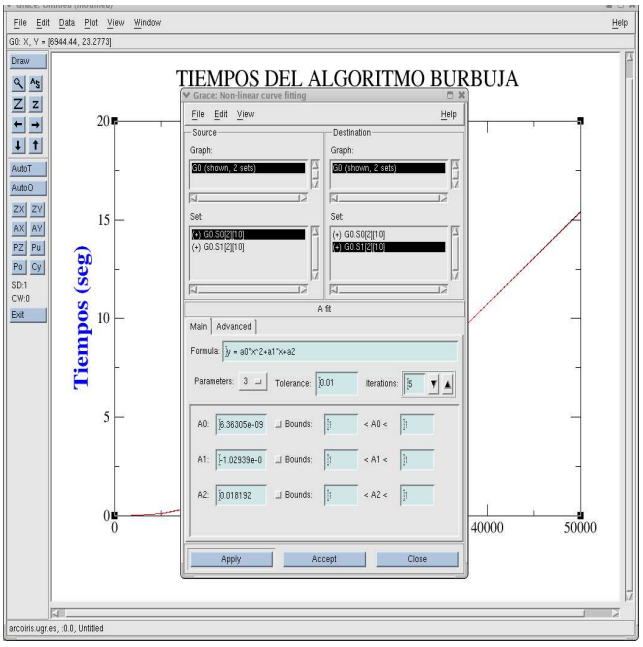

Figura 8: Introducir T(n) y el número de constantes ocultas

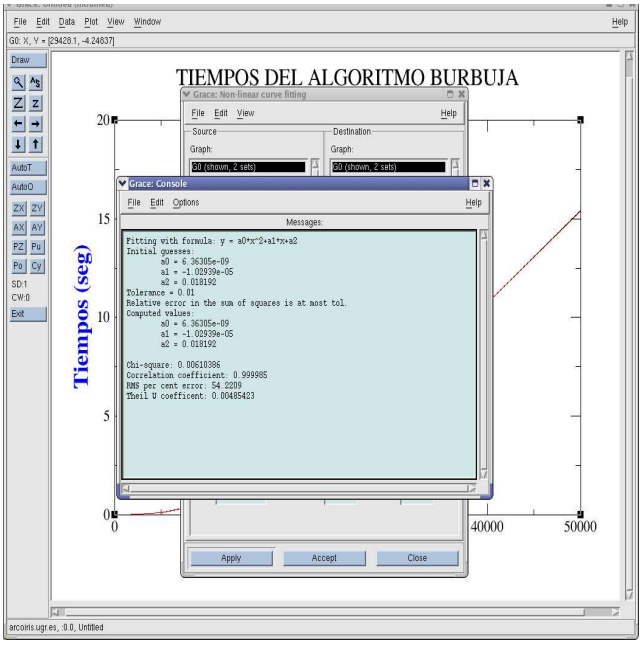

Figura 9: Valores de las constantes ocultas y datos estadísticos de la bondad del ajuste

entre estos estadísticos podemos quedarnos con el coeficiente de correlación. De tal forma que el coeficiente de correlación cuanto mas se aproxime a 1 mejor ajuste representa.

Simultáneamente se representa la gráfica que se obtiene con $T(n)$ ya con los valores que hemos ajustado para las constantes ocultas.