Estructura de datos (2019-2020)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 1. Eficiencia

Patricia Maldonado Mancilla   Jesús Pérez Terrón

Índice

1. Eficiencia calculada de los siguientes algoritmos

Hemos calculado la eficiencia de algoritmos aprendidos en la asignatura de algorítmica que ya hemos cursado, además de otros extra.

1.1. Ejercicio2

- **Código fuente**

```cpp
#include <iostream>

#include <ctime>

using namespace std;

#
define MAXIMO 30000000
int M[MAXIMO];

void crear_matriz(int * M) {
  int i;

  for (i = 0; i < MAXIMO; i++)
    M[i] = i;
}
int opera(int * M, int n, int x, int inf, int sup) {
  int enc, med;

  enc = 0;
  while ((inf < sup) && (!enc)) {
    med = (inf + sup) / 2;
    if (M[med] == x)
      return med;
    else if (M[med] < x)
      inf = med + 1;
    else sup = med - 1;
  }
  if (enc)
    return med;
  else
    return -1;
}

int alg2_1(int * M, int n, int x) { // o(n)
  int i;

  for (i = 0; i < n; i++)
    opera(M, n, x, 0, n - 1);
}

main() {
  clock_t ti, tf;
  int n_datos;

  crear_matriz(M);

  for (n_datos = 1000000; n_datos < MAXIMO; n_datos += 500000) {
    ti = clock();
    alg2_1(M, n_datos, -1);
    tf = clock();
    cout << n_datos << "\t" << tf - ti << endl;

  }

}
```

Figura 1.1: Ejercicio 2 código fuente

- **Eficiencia teórica**
  O(log2(n))

- **Eficiencia empírica**

  Para las variables a,b obtenemos el siguiente resultado:
  gnuplot>f(x)=a*x*log(x) + b
  gnuplot>fit f(x)"tiempos.dat"via a,b
  plot "tiempos.dat", f(x)

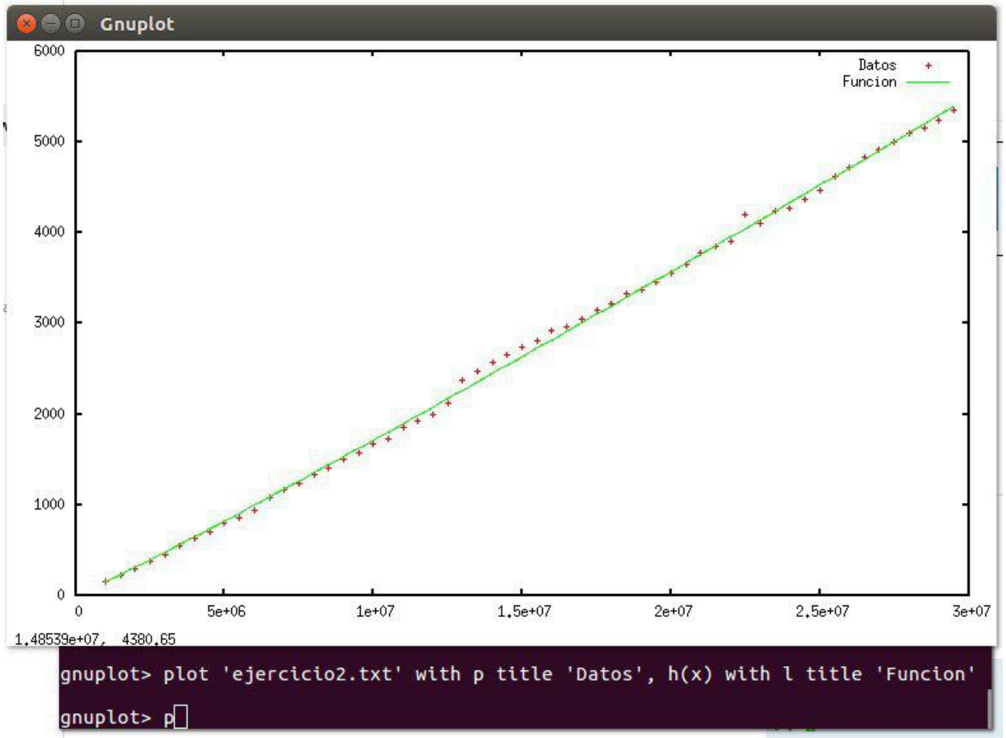


Figura 1.2: Ejercicio2 gráfica caso peor

1.2. Burbuja

- **Código fuente**

```cpp
#include <iostream>
#include <vector>
#include <chrono>    // Recursos para medir tiempos
#include <cstdlib>

using namespace std;

using namespace std::chrono;

void burbuja(vector<int> V,int n){
    int aux;
    aux=0;//1
    for (int i=0; i<n-1; i++) // sumatoria i=0, hasta n-2 --
    {
       for (int j=i+1; j<n; j++)//sumatoria j=i+1,hasta n1 -- 3
      {
        if(V[i]>V[j]) //3
        {
         aux = V[i];//2
         V[i] = V[j];//3
         V[j] = aux;//2
        }
      }
    }
}


int main(int argc, char * argv[]){

    vector<int> V;
    int tam = atoi(argv[1]);
    //int vmax = atoi(argv[2]);

    for (int i = tam-1; i>=0; i--)
    {
        V.push_back(i);
    }

  high_resolution_clock::time_point start,end;
  duration<double> tiempo_transcurrido;

  start = high_resolution_clock::now(); // tiempo de inicio

    burbuja(V,tam);

  end = high_resolution_clock::now(); // tiempo fin

  tiempo_transcurrido = duration_cast<duration<double> > (end-start);

  cout << tam  << "\t" << tiempo_transcurrido.count() << endl;


}
```

Figura 1.3: Burbuja código fuente caso peor

- **Eficiencia teórica**

$$1+3\left(\sum_{i=0}^{n-2} 3 + \left(\sum_{j=i+1}^{n-1} 12\right) + 3\right) =$$

$$4 + \left(\sum_{i=0}^{n-2} 6 + \sum_{j=i+1}^{n-1} 12\right) = 4 + 6(n-1) + 12 \sum_{i=0}^{n-2} n - i - 1 =$$

$$4 + 6n - 6 + 12n(n-1) - 12(n-1)\left(\frac{n-2}{2}\right) - 12(n-1) =$$

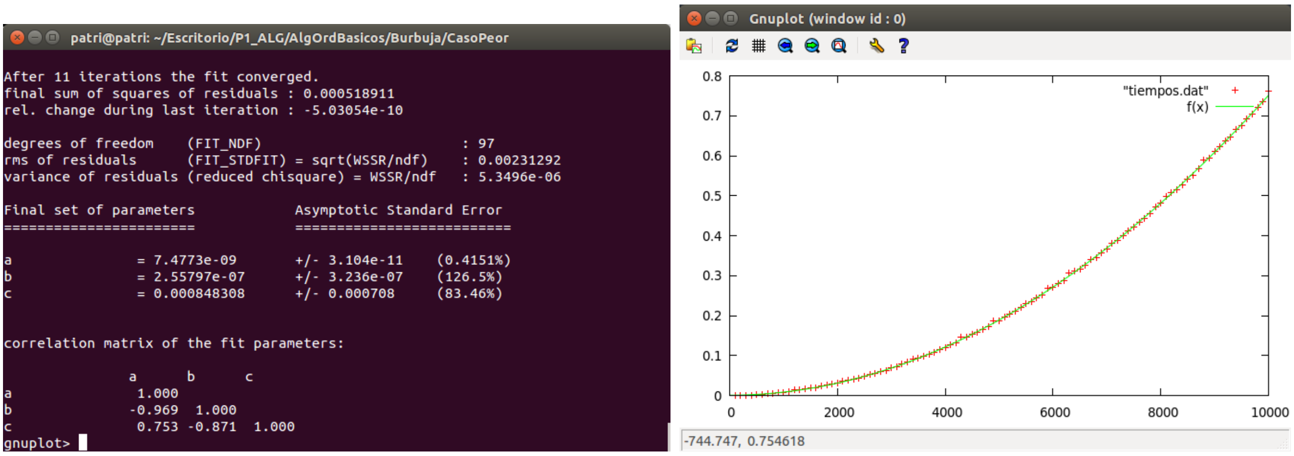
$$4 + 6n - 6 + 12n^2 - 12n - 6n^2 + 18n + 12 - 12n + 12 =$$

$$6n^2 + 22 \in O(n^2)$$

Figura 1.4: Burbuja eficiencia teórica caso peor

- **Eficiencia empírica**

Para las variables a,b y c obtenemos el siguiente resultado:
gnuplot>f(x)=(a*x*x)+(b*x)+c
gnuplot>fit f(x)"tiempos.dat"via a,b,c



(a) Ejecución (b) Gráfica

Figura 1.5: Burbuja ejecución y gráfica caso peor

1.3. Inserción

- **Código fuente**

```
using namespace std;

using namespace std::chrono;


 void inserccion(vector<int> V,int n)
{

  // int n = V.size();
   for (int i = 1; i < n; i++)
   {
       int v = V[i];

       int j = i - 1;

       while (j >= 0 && V[j] > v)
          {

              V[j + 1] = V[j];
              j--;
          }
       V[j + 1] = v;
   }

}

int main(int argc, char * argv[]){

    vector<int> V;
    int tam = atoi(argv[1]);
    //int vmax = atoi(argv[2]);

    for (int i = tam-1; i>=0; i--)
    {
        V.push_back(i);
    }

  high_resolution_clock::time_point start,end;
  duration<double> tiempo_transcurrido;

  start = high_resolution_clock::now();    // Anotamos el tiempo de inicio

    inserccion(V,tam);

  end = high_resolution_clock::now();

tiempo_transcurrido = duration_cast<duration<double> > (end-start);

   cout << tam << "\t" << tiempo_transcurrido.count() << endl;


}
```

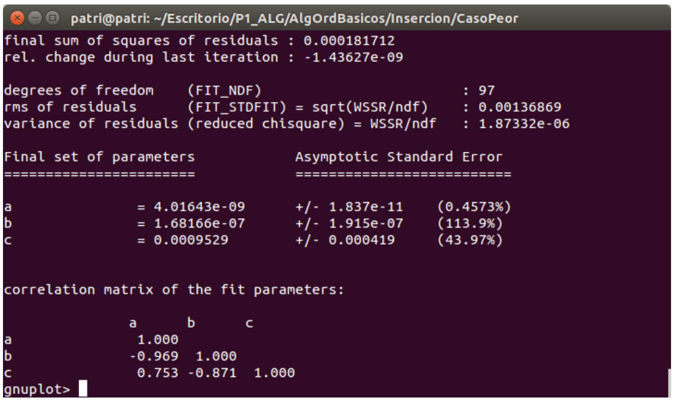
Figura 1.6: Inserción código fuente caso peor

- **Eficiencia teórica**

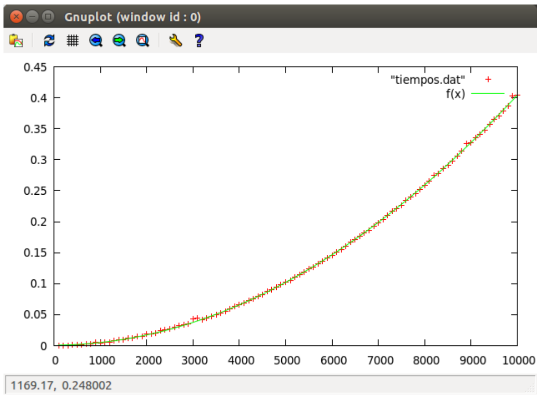
$$2 + \sum_{i=1}^{n-1} 4 + 4 \sum_{j=0}^{i-1} 4 + 2 + 4) + 3 + 3 =$$
$$2 + \sum_{i=1}^{n-1} 14 + \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 10 = 2 + 14(n-1) + \sum_{i=1}^{n-1} 10\, i =$$
$$2 + 14(n-1) + 10\,(1+2 \ldots + n-1) =$$
$$2 + 14(n-1) + 10(n(\frac{n-1}{2}) =$$
$$2 + 14\,n - 14 + 5n^2 - 5n$$
$$9n - 12 + 5n^2 \in O(n^2)$$

Figura 1.7: Inserción eficiencia teórica caso peor

- **Eficiencia empírica** Para las variables a,b obtenemos el siguiente resultado: gnuplot>f(x)=(a*x*x)+(b*x)+c



(a) Ejecución



(b) Gráfica

Figura 1.8: Inserción ejecución y gráfica caso peor

1.4. Selección

- **Código fuente**

```
void seleccion(vector<int> V,int n)
{
    int i, j, m, mi;
    for (i = 0; i < n - 1; i++)
    {
        /* find the minimum */
        mi = i;
        for (j = i + 1; j < n; j++)
            if (V[j] < V[mi])
                mi = j;

        m = V[mi];

        /* move elements to the right */
        for (j = mi; j > i; j--)
            V[j] = V[j-1];

        V[i] = m;
    }
}
int main(int argc, char * argv[]){

    vector<int> V;
    int tam = atoi(argv[1]);
    //int vmax = atoi(argv[2]);

    for (int i = tam-1; i>=0; i--)
    {
        V.push_back(i);
    }

  high_resolution_clock::time_point start,end;
  duration<double> tiempo_transcurrido;

  start = high_resolution_clock::now();    // Anotamos el tiempo de inicio


    seleccion(V,tam);

  end = high_resolution_clock::now();

tiempo_transcurrido = duration_cast<duration<double> > (end-start);

  cout << tam << "\t" << tiempo_transcurrido.count() << endl;


}
```

Figura 1.9: Seleccion código fuente caso peor

- **Eficiencia teórica**

$$1 + 3 + (\sum_{i=0}^{n-2} 1 + 3(\sum_{j=i+1}^{n-1} 3 + 1 + 3) + 2 + 3 + 2 + 3) =$$

$$4 + (\sum_{i=0}^{n-2} 14(\sum_{j=i+1}^{n-1} 7)) = 4 + 14n - 14 + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 7 =$$

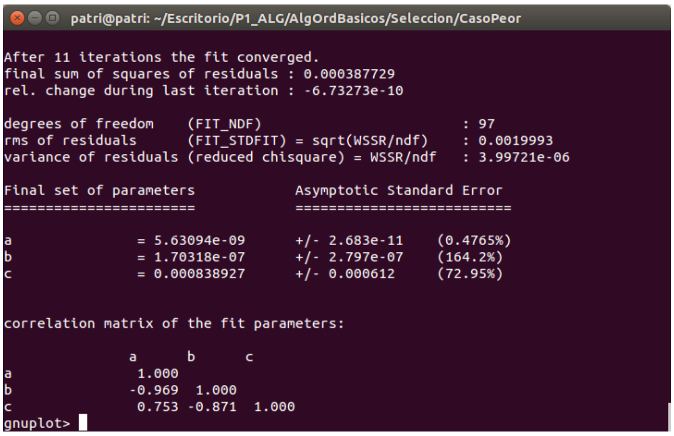
$$4 + 14n - 14 + 7\sum_{i=0}^{n-2} n - i - 1 = 14n - 14 + 7(\sum_{i=0}^{n-2} n \sum_{i=0}^{n-2} i \sum_{i=0}^{n-2} 1) =$$

$$14n - 14 + 7n^2 - 7n - (7n + 7(\frac{n-2}{2})) - 7n + 7 =$$

$$14n - 14 + 7n^2 - 7n - \frac{7}{2}n^2 + \frac{21}{2} = 7n^2 + \frac{21}{2}n - 14 \in O(n^2)$$

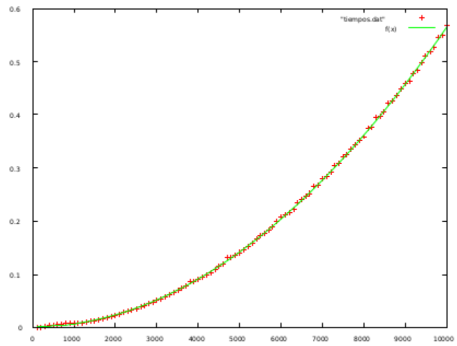
Figura 1.10: Selección eficiencia teórica caso peor

- **Eficiencia empírica** Para las variables a,b obtenemos el siguiente resultado:
  gnuplot>f(x)=(a*x*x)+(b*x)+c
  gnuplot>fit f(x)"tiempos.dat"via a,b,c



```
patri@patri: ~/Escritorio/P1_ALG/AlgOrdBasicos/Seleccion/CasoPeor

After 11 iterations the fit converged.
final sum of squares of residuals : 0.000387729
rel. change during last iteration : -6.73273e-10

degrees of freedom    (FIT_NDF)                        : 97
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0019993
variance of residuals (reduced chisquare) = WSSR/ndf   : 3.99721e-06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 5.63094e-09      +/- 2.683e-11    (0.4765%)
b               = 1.70318e-07      +/- 2.797e-07    (164.2%)
c               = 0.000838927      +/- 0.000612     (72.95%)


correlation matrix of the fit parameters:

                a      b      c
a               1.000
b              -0.969  1.000
c               0.753 -0.871  1.000
gnuplot>
```

(a) Ejecución



(b) Gráfica

Figura 1.11: Selección ejecución y gráfica caso peor

1.5. ABB

- **Código fuente**

```
using namespace std;
using namespace std::chrono;


void ListarAbb(ABB<int> &ab_bus){
  ABB<int>::nodo n;
 cout<<endl<<"Elementos ordenados: ";

 for (n=ab_bus.begin();n!=ab_bus.end();++n){
       cout<<*n<<" ";
 }
}

int main(int argc, char * argv[]){

    int tam = atoi(argv[1]);
     vector<int>v, k;

     high_resolution_clock::time_point start,end;
     duration<double> tiempo_transcurrido;

     // Inicio tiempo
    start = high_resolution_clock::now();

    for(int i = 0; i<tam; i++){
        k.push_back(i);
    }

     ABB<int>ab_bus;

     for (int i=0;i<tam;i++){
           ab_bus.Insertar(i);

     }

    ABB<int>::nodo n;
    for (n=ab_bus.begin();n!=ab_bus.end();++n){
        v.push_back(*n);
    }

// fin tiempo
 end = high_resolution_clock::now();
tiempo_transcurrido = duration_cast<duration<double> > (end-start);

  cout << tam << "\t" << tiempo_transcurrido.count() << endl;

}
```

Figura 1.12: ABB código fuente caso peor

- **Eficiencia teórica**

$$T(n) \begin{cases} 1 & n = 1 \text{ ó } n = 2 \\ T(n-1) + 1 & n \geq 2 \end{cases}$$

$$T(n) - T(n-1) = 1$$

$$(x-1)(x-1) = 0 \qquad \begin{cases} b = 1 \\ p(n) = 1 \end{cases} \quad d = 0$$

$$T(n) = c1 \cdot 1^n + c2 \cdot n \cdot 1^n$$

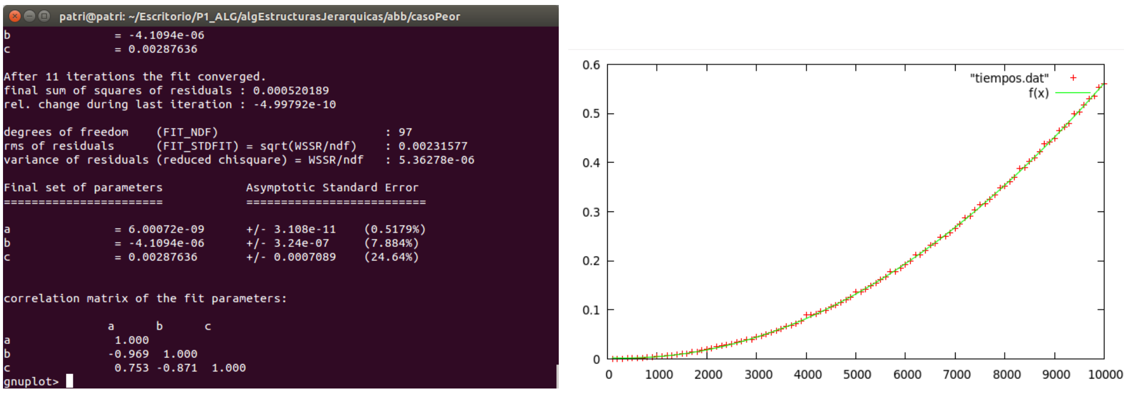
$$T(n) \in O(n)$$

$$\boldsymbol{T(n) * n \; etiquetas \in O(n^2)}$$

Figura 1.13: ABB eficiencia teórica caso peor

- **Eficiencia empírica**

  Para las variables a,b obtenemos el siguiente resultado:
  gnuplot>f(x)=(a*x*x)+(b*x)+c
  gnuplot>fit f(x)"tiempos.dat"via a,b,c
  plot "tiempos.dat", f(x)



(a) Ejecución (b) Gráfica

Figura 1.14: ABB ejecución y gráfica caso peor

1.6. APO

- **Código fuente**

```cpp
#include "apo.h"
#include <chrono>    // Recursos para medir tiempos
#include <vector>

using namespace std;
using namespace std::chrono;

int main(int argc, char * argv[]){
int tam = atoi(argv[1]);
 vector<int>v;
APO<int>ap_int;
      //cout<<"Introduce un APO:";

        high_resolution_clock::time_point start,end;
      duration<double> tiempo_transcurrido;
      start = high_resolution_clock::now();
        //vector ordenado == peor caso
        for(int i = 0; i<tam; i++){
            ap_int.insertar(i);
        }
        while(!ap_int.vacio()){
            v.push_back(ap_int.minimo());
            ap_int.borrar_minimo();
        }
        //cin>>ap_int;
      //cout<<"APO introducido:"<<ap_int;

        end = high_resolution_clock::now();
      tiempo_transcurrido = duration_cast<duration<double> > (end-start);

        cout <<tam << "\t" << tiempo_transcurrido.count() << endl;

}
```

Figura 1.15: APO código fuente caso peor

- **Eficiencia teórica**

$$T(n)\begin{cases}1 & n=0 \text{ ó } n=1\\ T(n/2)+1 & n\geq 2\end{cases}$$

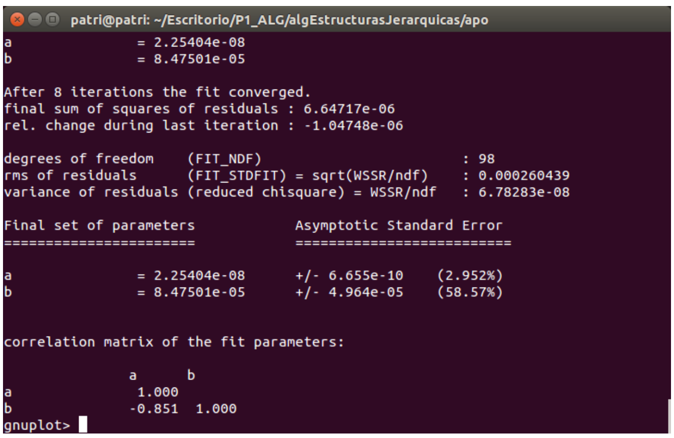
$$T(n)=T(n/2)+1$$
$$n=2^m \quad m=\log_2 n$$
$$T(2^m)=T(2^{m-1})+1$$
$$T(2^m)-T(2^{m-1})=1$$
$$(x-1)(x-1)=0 \qquad \begin{cases} b=1\\ p(n)=1\end{cases} \quad d=0$$

Figura 1.16: APO eficiencia teórica caso peor

$$T(2^m)=c1\cdot 1^m+c2\cdot m\cdot 1^m$$
$$T(2^m)=c1+\log_2 n\cdot c2$$
$$T(1)=1=C1$$
$$T(2)=2=C1+2\cdot C2 \;=>\; C2=\frac{1}{2}$$
$$T(n)\in O(\log_2(n))$$

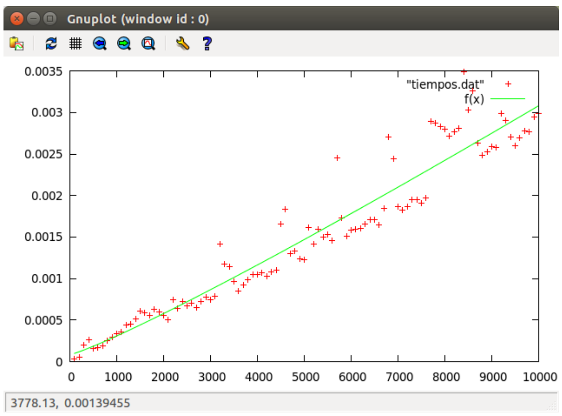
Figura 1.17: APO eficiencia teórica caso peor

- **Eficiencia empírica** Para las variables a,b obtenemos el siguiente resultado:
gnuplot>f(x) = (a*x*log(x)/log(2)) +b
gnuplot>fit f(x)"tiempos.dat"via a,b
plot "tiempos.dat", f(x)



```
patri@patri: ~/Escritorio/P1_ALG/algEstructurasJerarquicas/apo
a               = 2.25404e-08
b               = 8.47501e-05

After 8 iterations the fit converged.
final sum of squares of residuals : 6.64717e-06
rel. change during last iteration : -1.04748e-06

degrees of freedom    (FIT_NDF)                        : 98
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.000260439
variance of residuals (reduced chisquare) = WSSR/ndf   : 6.78283e-08

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 2.25404e-08      +/- 6.655e-10    (2.952%)
b               = 8.47501e-05      +/- 4.964e-05    (58.57%)


correlation matrix of the fit parameters:

                a      b
a               1.000
b              -0.851  1.000
gnuplot>
```

(a) Ejecución



(b) Gráfica

Figura 1.18: APO ejecución y gráfica caso peor