ElonMusk
www.wuolah.com/student/ElonMusk

Practica-1-ED.pdf

Práctica 1 ED - Resuelta

2° Estructuras de Datos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

SE BUSCA

FRASE INGENIOSA PARA CAMI MEDIOCRE.

UNA FRASE QUE PUEDAS DECIR TANTO EN LA CAMA

COMO EN CLASE

1 ESCRIBE TU FRASE EN LA CAMI

2 HAZLE FOTO Y SÚBELA A STORIES

3 OJITO, QUE PUEDE SER TUYA

UNA IDEA DE MIERDA POR @TELOPUTODIJE_SHOP

TE LO PUTODIJE

Nota: cuando entregues la práctica no olvides añadir los datos que debes especificar, que son:

- Código fuente
- Hardware usado (CPU, velocidad de reloj, memoria RAM, …)
- Sistema operativo
- Compilador utilizado y opciones de compilación
- Desarrollo completo del cálculo de la eficiencia teórica y gráfica.
- Parámetros usados para el cálculo de la eficiencia empírica y gráfica.
- Ajuste de la curva teórica a la empírica: mostrar resultados del ajuste y gráfica.

Ejercicio 1

Ejercicio 1: ***Ordenación de la burbuja***

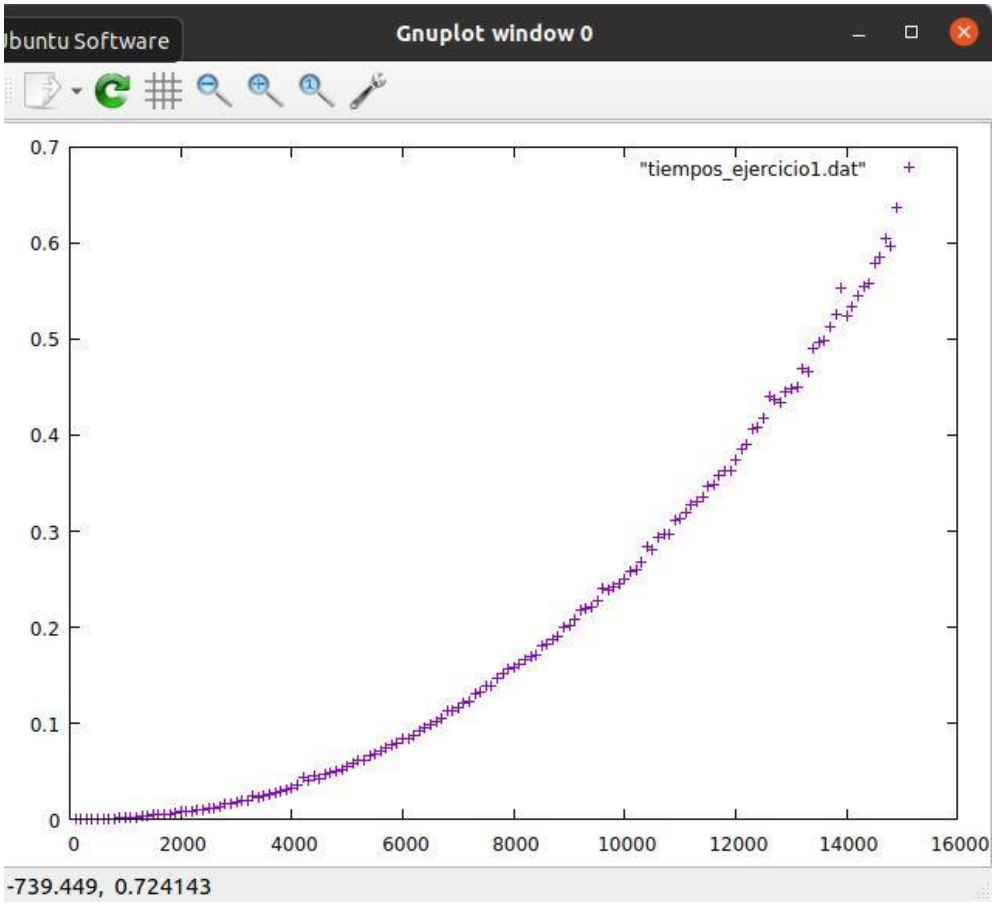
El siguiente código realiza la ordenación mediante el algoritmo de la burbuja:

```
void ordenar(int *v, int n) {
for (int i=0; i<n-1; i++)
for (int j=0; j<n-i-1; j++)
if (v[j]>v[j+1]) {
int aux = v[j];
v[j] = v[j+1];
v[j+1] = aux;
}
}
```

- Código fuente:

```
#include <iostream>

#include <ctime> // Recursos para medir tiempos

#include <cstdlib> // Para generación de números pseudoaleatorios

using namespace std;



void sintaxis() {

 cerr << "Sintaxis:" << endl;

 cerr << " TAM: Tamaño del vector (>0)" << endl;

 cerr << " VMAX: Valor máximo (>0)" << endl;

 cerr << "Genera un vector de TAM números aleatorios en [0,VMAX[" << endl;
```

```
 exit(EXIT_FAILURE);
}

int main(int argc, char * argv[]) {
 if (argc!=3) // Lectura de parámetros
 sintaxis();
 int tam=atoi(argv[1]); // Tamaño del vector
 int vmax=atoi(argv[2]); // Valor máximo
 if (tam<=0 || vmax<=0)
 sintaxis();

 // Generación del vector aleatorio
 int *v=new int[tam]; // Reserva de memoria
 srand(time(0)); // Inicialización generador números pseudoaleatorios
 for (int i=0; i<tam; i++) // Recorrer vector
 v[i] = rand() % vmax; // Generar aleatorio [0,vmax[

 clock_t tini; // Anotamos el tiempo de inicio
 tini=clock();



 ordenar(v,tam);

 clock_t tfin; // Anotamos el tiempo de finalización
 tfin=clock();
 // Mostramos resultados (Tamaño del vector y tiempo de ejecución en seg.)
 cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;

 delete [] v; // Liberamos memoria dinámica
}
```



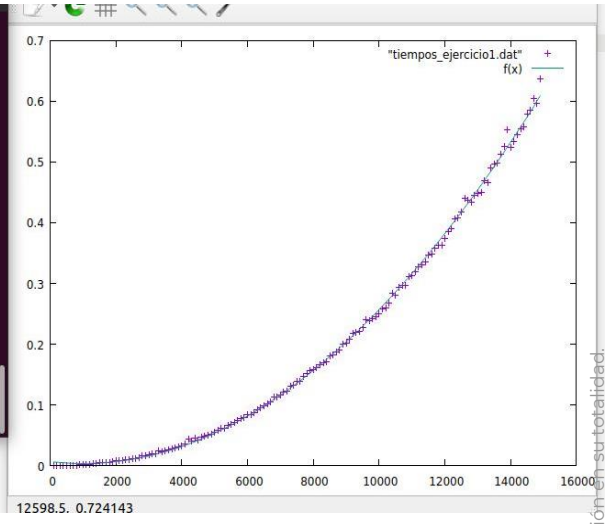
Ejercicio 2

Ejercicio 2: ***Ajuste en la ordenación de la burbuja***

Replique el experimento de ajuste por regresión a los resultados obtenidos en el ejercicio 1 que
calculaba la eficiencia del algoritmo de ordenación de la burbuja. Para ello considere que f(x) es
de la forma ax^2+bx+c

```
iter      chisq       delta/lim  lambda   a             b             c

After 11 iterations the fit converged.
final sum of squares of residuals : 0.00431443
rel. change during last iteration : -2.94083e-08

degrees of freedom    (FIT_NDF)                        : 146
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00543607
variance of residuals (reduced chisquare) = WSSR/ndf   : 2.95509e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 3.1298e-09       +/- 2.692e-11    (0.86%)
b               = -6.30419e-06     +/- 4.168e-07    (6.611%)
c               = 0.00713983       +/- 0.001354     (18.97%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.969  1.000
c               0.750 -0.869  1.000
gnuplot> plot "tiempos_ejercicio1.dat", f(x)
gnuplot>
```

```
 vector aleatorio
m]; // Reserva de memoria
/ Inicialización generador números pseudoaleatorios
```



A: 3.1298e-09 B: -6.30419e-06 C: 0.00713983 (esto sale diferente en cada PC)

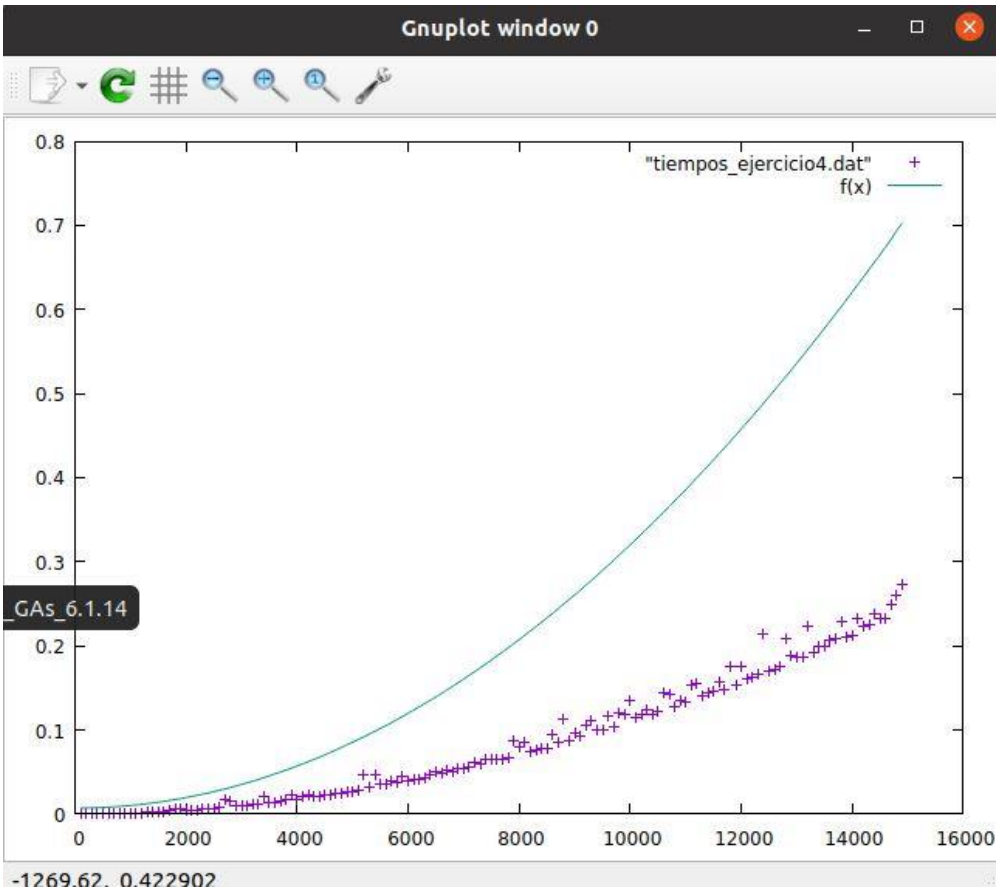
Se observa un muy buen ajuste de la recta ya que coindice casi con todos los puntos mostrados en la gráfica

Ejercicio 4

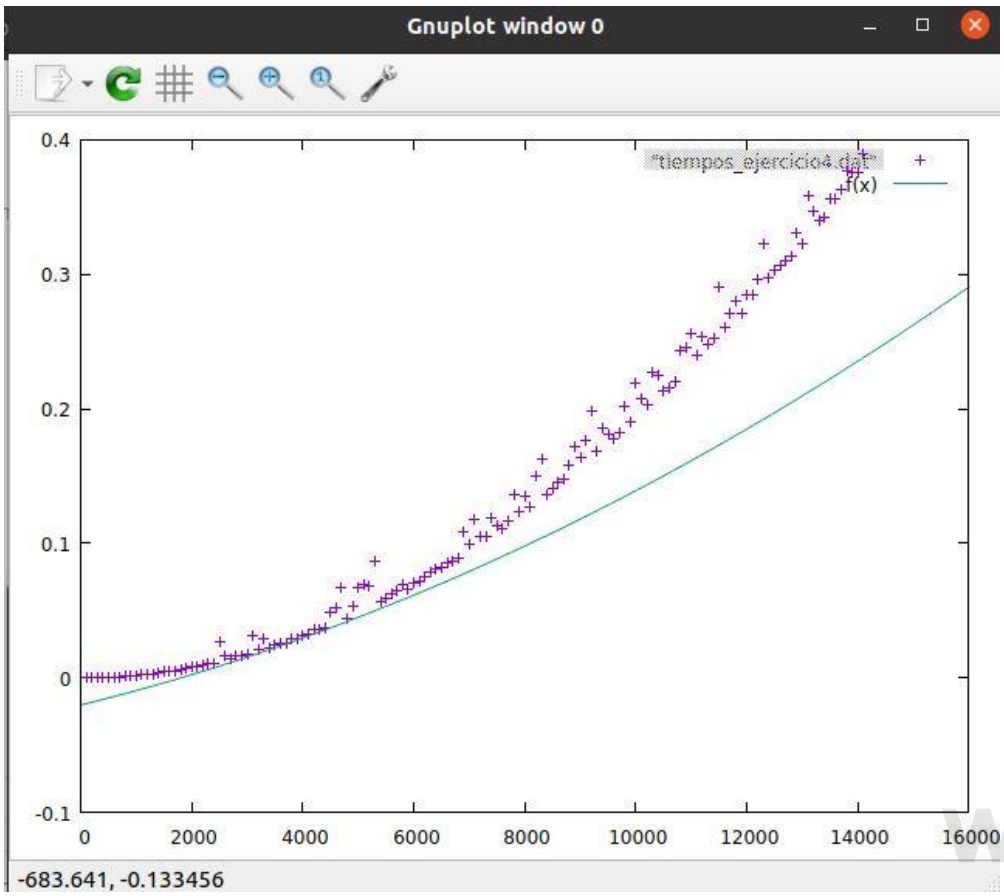
Ejercicio 4: ***Mejor y peor caso***

Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Debe modificar el código
que genera los datos de entrada para situarnos en dos escenarios diferentes:
• El mejor caso posible. Para este algoritmo, si la entrada es un vector que ya está ordenado
el tiempo de cómputo es menor ya que no tiene que intercambiar ningún elemento.
• El peor caso posible. Si la entrada es un vector ordenado en orden inverso estaremos en la
peor situación posible ya que en cada iteración del bucle interno hay que hacer un intercambio.
Calcule la eficiencia empírica en ambos escenarios y compárela con el resultado del ejercicio 1.

Aunque lo más frecuente será preguntar por números que estén en posiciones arbitrarias del vector
(casos promedio).



La función verde representa el ajuste de la función del primer ejercicio, y los puntos son los tiempos de ejecución del mejor caso, que se observan que han consumido menos tiempo (como era de esperar)



En el peor caso se observa que tarda más que en el caso medio, como era de esperar

Ejercicio 5

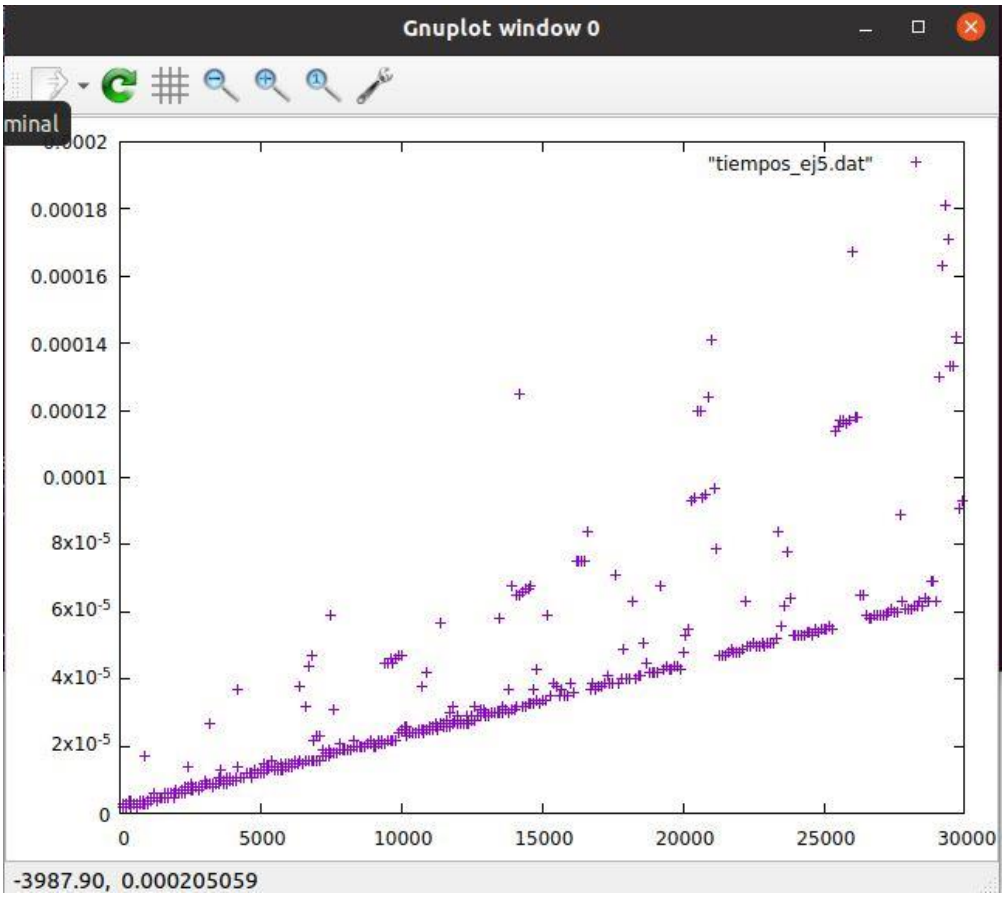
***Ejercicio** 5: Dependencia de la implementación*

Considere esta otra implementación del algoritmo de la burbuja:

```
void ordenar(int *v, int n) {
bool cambio=true;
for (int i=0; i<n-1 && cambio; i++) {
cambio=false;
for (int j=0; j<n-i-1; j++)
if (v[j]>v[j+1]) {
cambio=true;
int aux = v[j];
v[j] = v[j+1];
v[j+1] = aux;
}
}
}
```

En ella se ha introducido una variable que permite saber si, en una de las iteraciones del bucle
externo no se ha modificado el vector. Si esto ocurre significa que ya está ordenado y no hay que
continuar.

Considere ahora la situación del mejor caso posible en la que el vector de entrada ya está
ordenado. ¿Cuál sería la eficiencia teórica en ese mejor caso? Muestre la gráfica con la eficiencia
empírica y compruebe si se ajusta a la previsión.

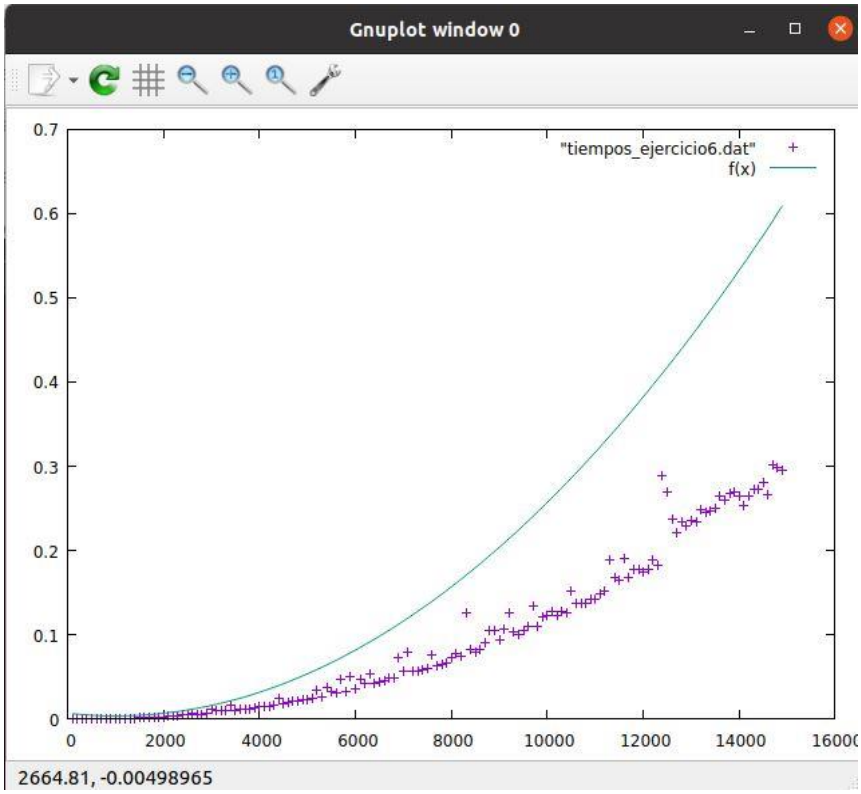


Al encontrarnos en el mejor caso vemos que los tiempos son bastante líneales

Ejercicio 6

Ejercicio 6: ***Influencia del proceso de compilación***

Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Ahora replique dicho
ejercicio pero previamente deberá compilar el programa indicándole al compilador que optimice
el código. Esto se consigue así:
g++ -O3 ordenacion.cpp -o ordenacion_optimizado
Compare las curvas de eficiencia empírica para ver cómo mejora esto la eficiencia del programa.



Al haber introducido la opción de compilación eficiente se observa una diferencia notable de tiempo entre la nube de puntos (compilación eficiente) y la curva verde (no eficiente, interpolación del ejercicio 2)

Ejercicio 7

Ejercicio 7: ***Multiplicación matricial***

Implemente un programa que realice la multiplicación de dos matrices bidimensionales. Realice
un análisis completo de la eficiencia tal y como ha hecho en ejercicios anteriores de este guión.

Código:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

void generarArreglo(int **matriz, int rows, int cols, int valor){
   srand(time(NULL));
   for(int i=0;i<rows;i++){
      for(int j=0;j<cols;j++){
         matriz[i][j]=rand()%valor;
      }
   }
}

void imprimirArreglo(int **matriz, int rows, int cols){
   for(int i=0;i< rows;i++){
      for(int j=0;j< cols;j++){
         cout<<matriz[i][j]<<"\t";
      }
      cout<<endl;
   }
}

void multiplicar (int **m1, int **m2, int rows, int cols) {

          int** producto_mat = new int*[rows];
   for (int i = 0; i < rows; ++i)
      producto_mat[i] = new int[cols];
```

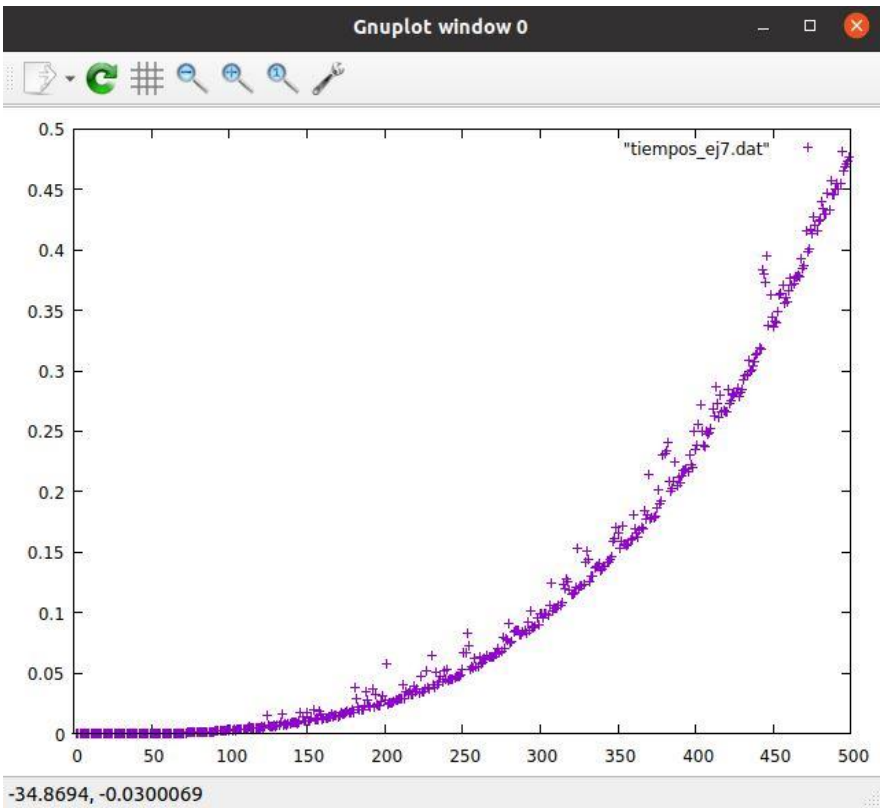
```
        int suma, producto;

        for (int y = 0; y < rows; y++){
                for (int z = 0; z < cols; z++){
                        suma = 0;

                        for (int k = 0; k < rows; k++){
                                producto = m1[y][k] * m2[k][z];
                                suma = suma + producto;
                        }
                        producto_mat[y][z] = suma;
                }
          }
        }

void sintaxis() {
         cerr << "Sintaxis:" << endl;
         cerr << " TAM: Tamaño del vector (>0)" << endl;
         cerr << " VMAX: Valor máximo (>0)" << endl;
         cerr << "Genera un vector de TAM números aleatorios en [0,VMAX[" << endl;
         exit(EXIT_FAILURE);
}

int main(int argc, char * argv[]){
        if (argc!=3) // Lectura de parámetros
        sintaxis();
        int tam=atoi(argv[1]); // Tamaño de las matrices
        int vmax=atoi(argv[2]); // Valor máximo
        if (tam<=0 || vmax<=0)
        sintaxis();
```

```
   int rows , cols;
   rows = cols = tam;

   int** matriz1 = new int*[rows];
   for (int i = 0; i < rows; ++i)
      matriz1[i] = new int[cols];
   generarArreglo(matriz1, rows, cols, vmax);

   int** matriz2 = new int*[rows];
   for (int i = 0; i < rows; ++i)
      matriz2[i] = new int[cols];
   generarArreglo(matriz2, rows, cols, vmax);

    clock_t tini; // Anotamos el tiempo de inicio
          tini=clock();

   multiplicar (matriz1, matriz2, rows, cols);

    clock_t tfin; // Anotamos el tiempo de finalización
           tfin=clock();
           // Mostramos resultados (Tamaño del vector y tiempo de ejecución en seg.)
           cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;


   return 0;
}
```

Script:

```
#!/bin/csh
@ inicio = 2
```

```
@ fin = 500
@ incremento = 1
@ i = $inicio
echo > tiempos.dat

while ($i < $fin)
        echo Ejecución tam = $i
        echo `/home/david/Escritorio/hola/bin/matrices $i 100` >>
        tiempos_ej7.dat
        @ i += $incremento
end
```



En este caso es O(n^3), por lo que aumenta muy rápido conforme avanza el tamaño de los datos