

PRACTICA 1

PABLO RUIZ MINGORANCE

2º C (C 2)



Características del PC utilizado

```
bash: export: `/usr/lib/valgrind': no es un identificador válido
pablo@pablo-VirtualBox:~$ lscpu
Arquitectura: x86_64
modo(s) de operación de las CPUs: 32-bit, 64-bit
Orden de los bytes: Little Endian
CPU(s): 3
Lista de la(s) CPU(s) en línea: 0-2
Hilo(s) de procesamiento por núcleo: 1
Núcleo(s) por «socket»: 3
«Socket(s)»: 1
Modo(s) NUMA: 1
ID de fabricante: GenuineIntel
Familia de CPU: 6
Modelo: 158
Nombre del modelo: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Revisión: 9
CPU MHz: 2808.000
BogoMIPS: 5616.00
Fabricante del hipervisor: KVM
Tipo de virtualización: lleno
Caché L1d: 32K
Caché L1i: 32K
Caché L2: 256K
Caché L3: 6144K
CPU(s) del nodo NUMA 0: 0-2
Indicadores: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pnpi p
clmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm 3dnowprefetch
invpcid_single pti fsgsbase avx2 invpcid rdseed clflushopt flush_l1d
pablo@pablo-VirtualBox:~$
```

Compilador: GNU project C and C++ compiler
S.O.: Ubuntu 18.04.2 (en Máquina Virtual)

Ejercicio 1

Código:

```
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]) {  
                //swap(v[j],v[j+1]);  
                //alternativa al swap  
                int aux = v[j];//incluir algorithm  
                v[j] = v[j+1];  
                v[j+1] = aux;  
            }  
}
```

Eficiencia de forma teórica

Número de operaciones elementales que realiza:

- Línea 2: 4 OE (declaración, asignación, decremento, comparación).
- Línea 3: 5 OE (declaración, asignación, 2 decrementos, comparación).
- Línea 4: 4 OE (incremento, 2 accesos al elemento, comparación).
- Línea 5: 3 OE (declaración, acceso al elemento, asignación).
- Línea 6: 4 OE (incremento, 2 accesos al elemento, asignación).
- Línea 7: 3 OE (incremento, acceso al elemento, asignación).

Ecuación:

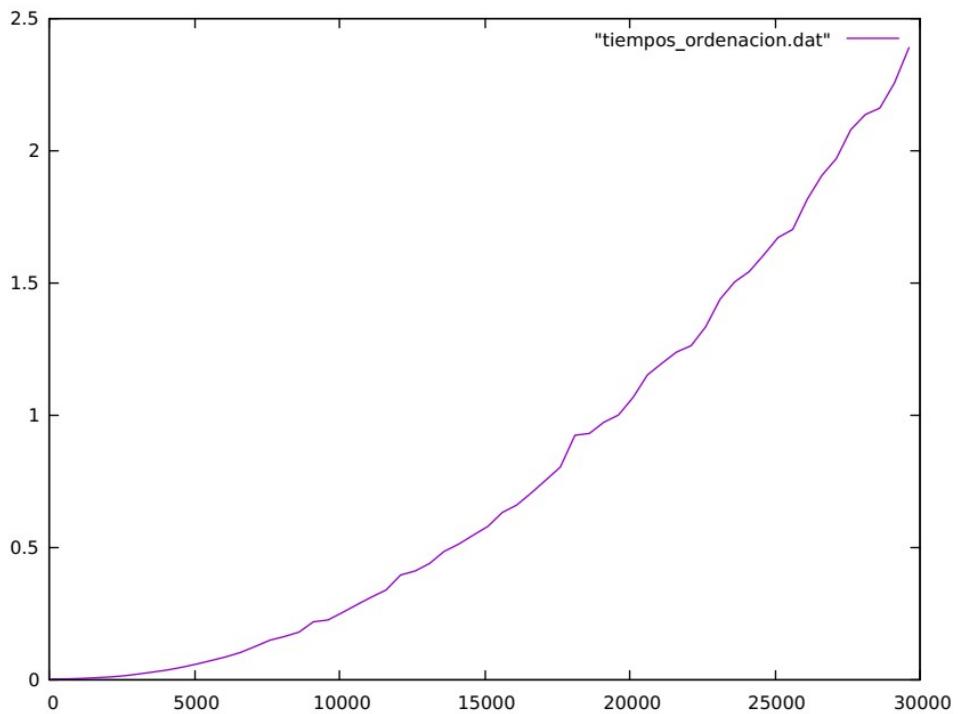
$$\begin{aligned} 4 + \sum_{i=0}^{n-2} \left(5 + \sum_{j=0}^{n-i-2} (4+3+4+3+2) + 2 \right) &= 4 + \sum_{i=0}^{n-2} \left(7 + \sum_{j=0}^{n-i-2} 16 \right) = \\ &= 4 + \sum_{i=0}^{n-2} 7 + \sum_{i=0}^{n-2} \sum_{j=0}^{n-i-2} 16 = 4 + 7 \sum_{i=0}^{n-2} 1 + \sum_{i=0}^{n-2} \left(16 \sum_{j=0}^{n-i-2} 1 \right) = 4 + 7(n-1) + \sum_{i=0}^{n-2} 16(n-i-1) = \\ &= 4 + 7(n-1) + 16 \sum_{i=0}^{n-2} (n-i-1) = 4 + 7(n-1) + 16 \left(\sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1 \right) = \\ &= 4 + 7(n-1) + 16n(n-1) - 8(n-1)(n-2) - 16(n-1) = \\ &= 4 + 7n - 7 + 16n^2 - 16n - 8n^2 + 24n - 16 - 16n + 16 = 8n^2 - n - 3 \end{aligned}$$

Pertenece a $O(n^2)$

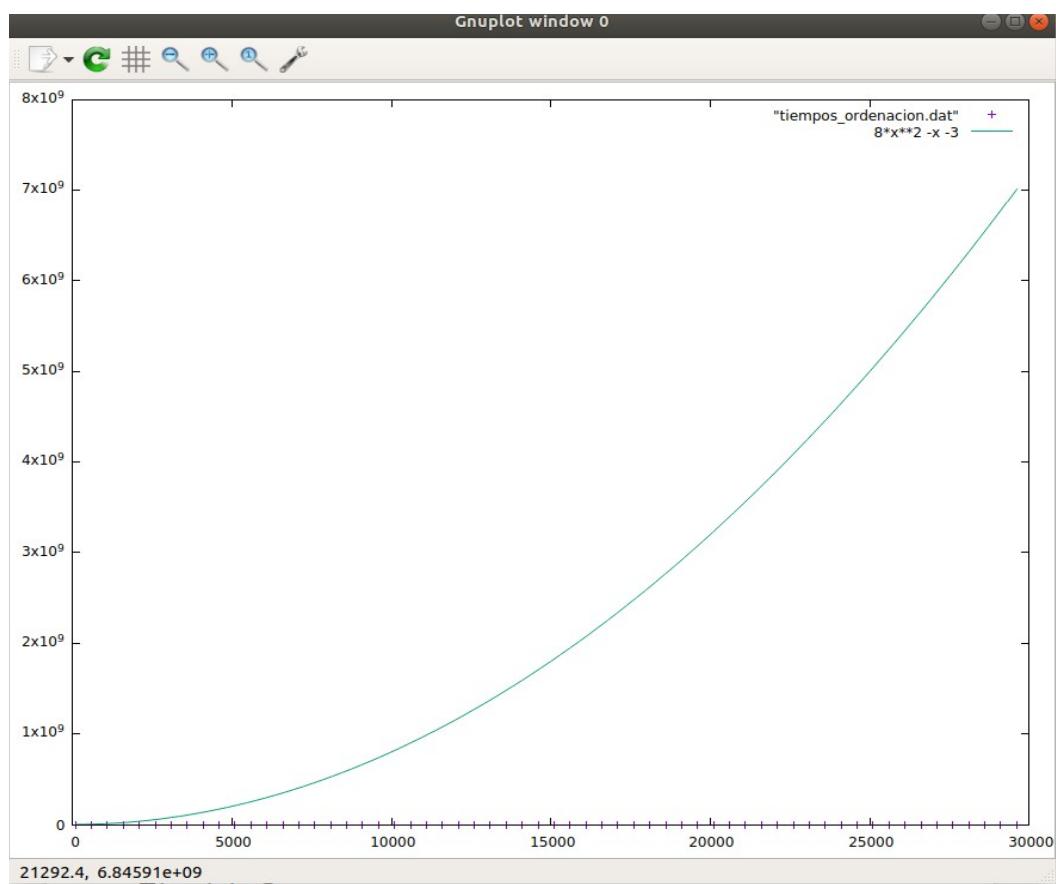
Tiempos:

Estudiaré el comportamiento del algoritmo de forma experimental, usando ejecuciones_ordenacion.csh. Tamaño inicial= 100, Tamaño máximo= 30000, incremento= 500, muestras totales=60 .

N	Segundos	N	Segundos
100	2E-05	15100	0.577343
600	0.000759	15600	0.62999
1100	0.002207	16100	0.658421
1600	0.004914	16600	0.704022
2100	0.008322	17100	0.752329
2600	0.012555	17600	0.801901
3100	0.019357	18100	0.921861
3600	0.027282	18600	0.928793
4100	0.0352	19100	0.97143
4600	0.045438	19600	0.998589
5100	0.057275	20100	1.06439
5600	0.070709	20600	1.15053
6100	0.084752	21100	1.19429
6600	0.101039	21600	1.23623
7100	0.123516	22100	1.26048
7600	0.147104	22600	1.33167
8100	0.160874	23100	1.43646
8600	0.177363	23600	1.50214
9100	0.216744	24100	1.54055
9600	0.223346	24600	1.60377
10100	0.251976	25100	1.66988
10600	0.281397	25600	1.70042
11100	0.310244	26100	1.81496
11600	0.336873	26600	1.90406
12100	0.393856	27100	1.96807
12600	0.408889	27600	2.07734
13100	0.437288	28100	2.13479
13600	0.483156	28600	2.15908
14100	0.510488	29100	2.25415
14600	0.544198	29600	2.38819



EMPÍRICA



La función de puntos es la empírica y la continua la teórica, se observa que la teórica crece mas rápido. Esto se debe a que en la forma teórica tomamos que cada OE vale un segundo.
Dificultad=5

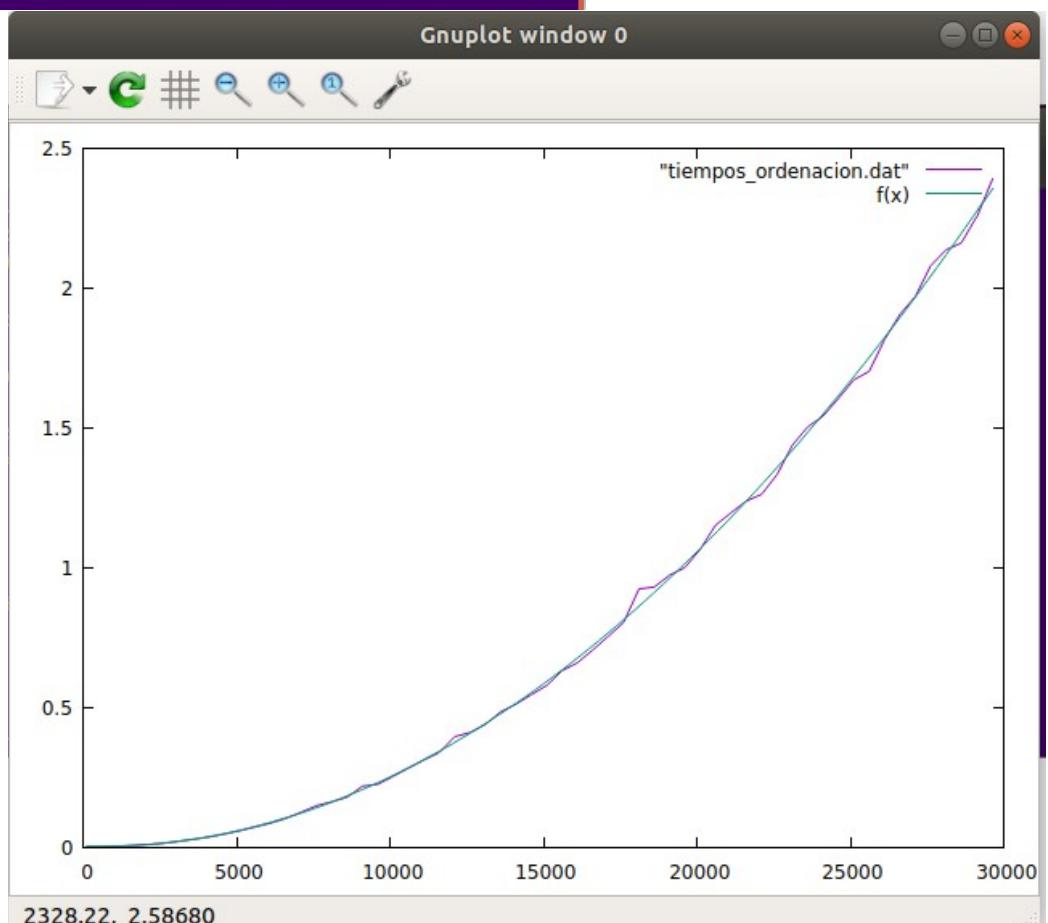
Ejercicio 2

En las siguientes capturas, se muestra las constantes de ajustar $f(x)$ a los resultados obtenidos en el ej 1. (Los valores experimentales son los mismos que el ejercicio anterior)

```
pablo@pablo-VirtualBox: ~/Escritorio/ED/material/creado/ej1
Archivo Editar Ver Buscar Terminal Ayuda
gnuplot> fit f(x) "tiempos_ordenacion.dat" via a,b,c
iter      chisq      delta/lim   lambda   a           b           c
0 9.47799534e+18  0.00e+00  2.29e+08  1.000000e+00  1.000000e+00  1.000000e+00
1 2.8930837525e+14 -3.28e+09  2.29e+07  5.483216e-03  9.999583e-01  1.000000e+00
2 1.1088372106e+09 -2.61e+10  2.29e+06 -4.157040e-05  9.999560e-01  1.000000e+00
3 1.1074783274e+09 -1.23e+02  2.29e+05 -4.186852e-05  9.997456e-01  1.000000e+00
4 1.0623078298e+09 -4.25e+03  2.29e+04 -4.100565e-05  9.791424e-01  9.999972e-01
5 1.1026270672e+08 -8.63e+05  2.29e+03 -1.320649e-05  3.153605e-01  9.999082e-01
6 2.4743851165e+03 -4.46e+09  2.29e+02 -5.598375e-08  1.355686e-03  9.998648e-01
7 6.9213163695e+00 -3.57e+07  2.29e+01  6.517512e-09 -1.366983e-04  9.997332e-01
8 6.7430367077e+00 -2.64e+03  2.29e+00  6.471952e-09 -1.350311e-04  9.867644e-01
9 1.2705920232e+00 -4.31e+05  2.29e-01  4.377099e-09 -6.000379e-05  4.269611e-01
10 1.7521262352e-02 -7.15e+06  2.29e-02  2.798074e-09 -3.450851e-06  5.001155e-03
11 1.7450063704e-02 -4.08e+02  2.29e-03  2.786082e-09 -3.021370e-06  1.796658e-03
12 1.7450063704e-02 -2.35e-06  2.29e-04  2.786081e-09 -3.021337e-06  1.796415e-03
iter      chisq      delta/lim   lambda   a           b           c
After 12 iterations the fit converged.
final sum of squares of residuals : 0.0174501
rel. change during last iteration : -2.35313e-11

degrees of freedom      (FIT_NDF)                      : 57
rms of residuals        (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0174969
variance of residuals  (reduced chisquare) = WSSR/ndf   : 0.000306141

Final set of parameters            Asymptotic Standard Error
=====
a      = 2.78608e-09      +/- 3.37e-11      (1.209%)
b      = -3.02134e-06     +/- 1.034e-06     (34.23%)
c      = 0.00179641       +/- 0.006645      (369.9%)
correlation matrix of the fit parameters:
      a      b      c
a      1.000
b      -0.968  1.000
c      0.738 -0.861  1.000
gnuplot>
```



Ejercicio 3

Es la Búsqueda binaria, sirve para encontrar un valor pasado por parámetro en un vector ordenado.

The screenshot shows a C++ code editor window titled "ejercicio3.cpp" located at "/Escritorio/ED/material/creado/ej3". The code implements a binary search algorithm. It includes headers for iostream, cstdlib, and chrono. It uses namespaces std and std::chrono. The function "operacion" takes four parameters: a pointer to an integer array "v", an integer "n", an integer "x", and two integers "inf" and "sup" representing the search range. The function initializes a boolean "enc" to false and a variable "med" to the midpoint of the range. It then enters a while loop where it checks if "inf" is less than "sup" and if "enc" is false. Inside the loop, it calculates "med" as the average of "inf" and "sup". It then compares the value at "v[med]" with "x": if they are equal, "enc" is set to true; if "v[med]" is less than "x", "inf" is set to "med+1"; otherwise, "sup" is set to "med-1". After the loop, if "enc" is true, it returns "med"; otherwise, it returns -1. The code editor interface includes buttons for "Abrir" (Open), "Guardar" (Save), and standard window controls. At the bottom, there are tabs for "C++" and "INS", and status bars for "Anchura del tabulador: 8" (Tab width: 8) and "Ln 1, Col 1" (Line 1, Column 1).

```
#include <iostream>
#include <cstdlib> // Para generación de números pseudoaleatorios
#include <chrono> // Recursos para medir tiempos
using namespace std;
using namespace std::chrono;

int operacion(int *v, int n, int x, int inf, int sup) {
    int med;
    bool enc=false;
    while ((inf<sup) && (!enc)) {
        med = (inf+sup)/2;
        if (v[med]==x)
            enc = true;
        else if (v[med] < x)
            inf = med+1;
        else
            sup = med-1;
    }
    if (enc)
        return med;
    else
        return -1;
}
```

Eficiencia de forma teórica //Línea 01 la cabecera de la función

- Línea 02: 1 OE (declaración).
- Línea 03: 2 OE (declaración, asignación).
- Línea 04: 3 OE (comparación, negación, aplica operador lógico).
- Línea 05: 3 OE (suma, división, asignación).
- Línea 06: 2 OE (acceso, comparación).
- Línea 07: 1 OE (asignación).
- Línea 08: 2 OE (acceso, comparación).
- Línea 09: 2 OE (incremento, asignación).
- Línea 11: 2 OE (decremento, asignación).
- Línea 13: 1 OE (comprobación).
- Línea 14 y 16: 1 OE (return).

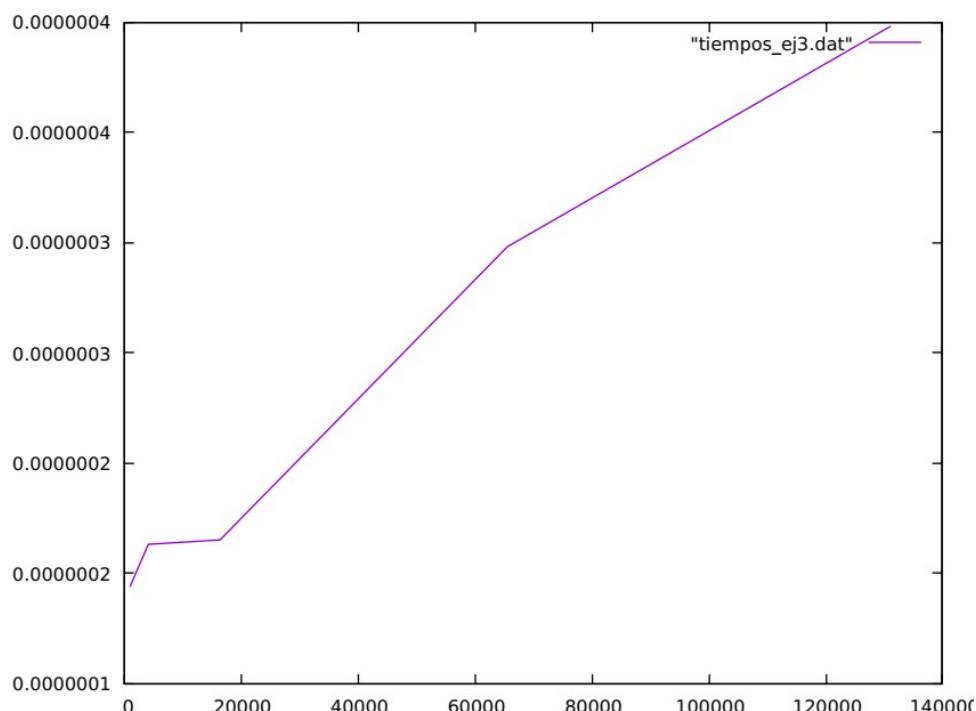
$$1 + 2 + 3 + \sum_{i=0}^{\log(n)} (3 + 4) + 1 + 1 = 8 + \sum_{i=0}^{\log(n)} 7 = 8 + 7 \sum_{i=0}^{\log(n)} 1 = 8 + 7(\log(n) + 1) = 7\log(n) + 15$$

Pertenece a $O(\log(n))$

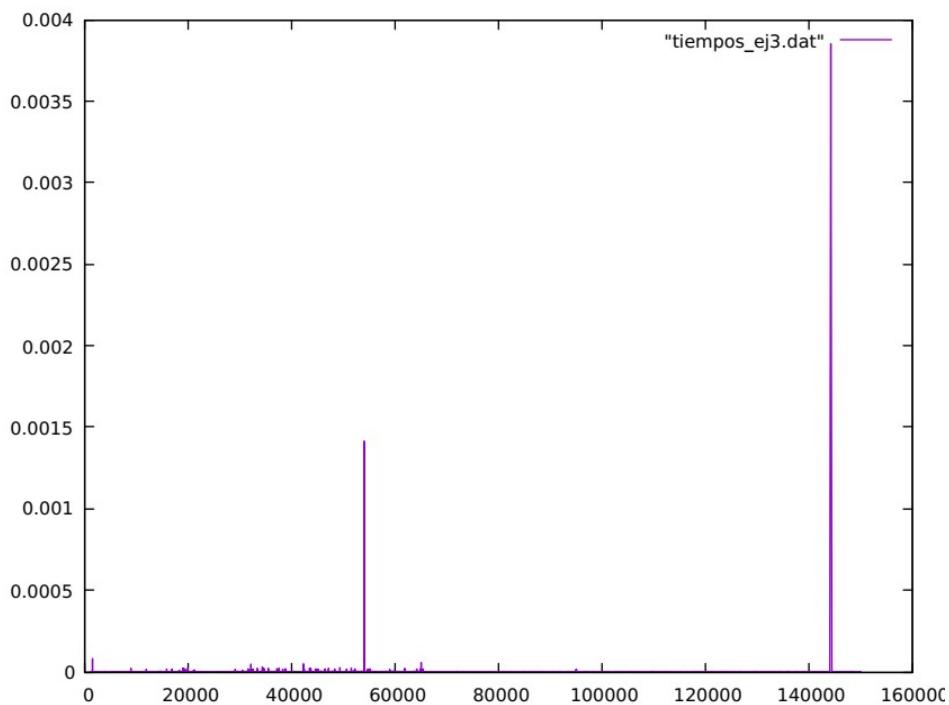
Tiempos:

Para estudiar el comportamiento del algoritmo de forma empírica he utilizado el script ejecuciones_ejercicio3.csh . Este script no realiza los incrementos lineales, sino en potencias de dos. He obtenido 5 muestras de la ejecución del algoritmo.

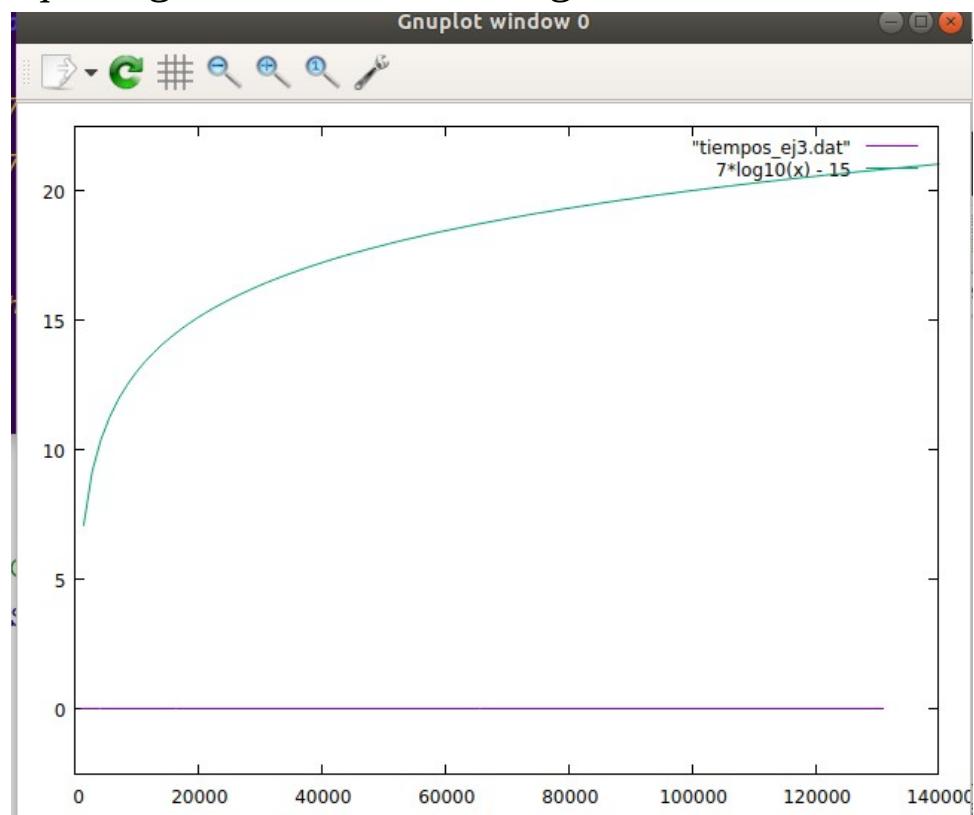
N	Segundos
1024	1.94e-07
4096	2.13e-07
16384	2.15e-07
65536	3.48e-07
131072	4.48e-07



Al cambiar los incrementos de forma lineal:



Se puede apreciar que los puntos más altos unidos pueden formar la curva que seguiría una función logarítmica.



Dificultad=6

Ejercicio 4

Código:

```
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;  
                swap (v[j],v[j+1]); //incluir algorithm  
            }  
    }  
}
```

Eficiencia de forma teórica (mejor caso)

En el mejor caso no se tendría que ejecutar a partir del ultimo if (sin incluirlo) y el primer bucle solo se ejecutaría una vez.

- Línea 02: 2 OE (declaración,asignación).
- Línea 03: 6 OE (declaración, asignación, decremento comparación,operador lógico).
- Línea 04: 1 OE (asignación).
- Línea 05: 5 OE (declaración, asignación, 2 decremento, comparación).
- Línea 06: 2 OE (2 acceso, comparación).

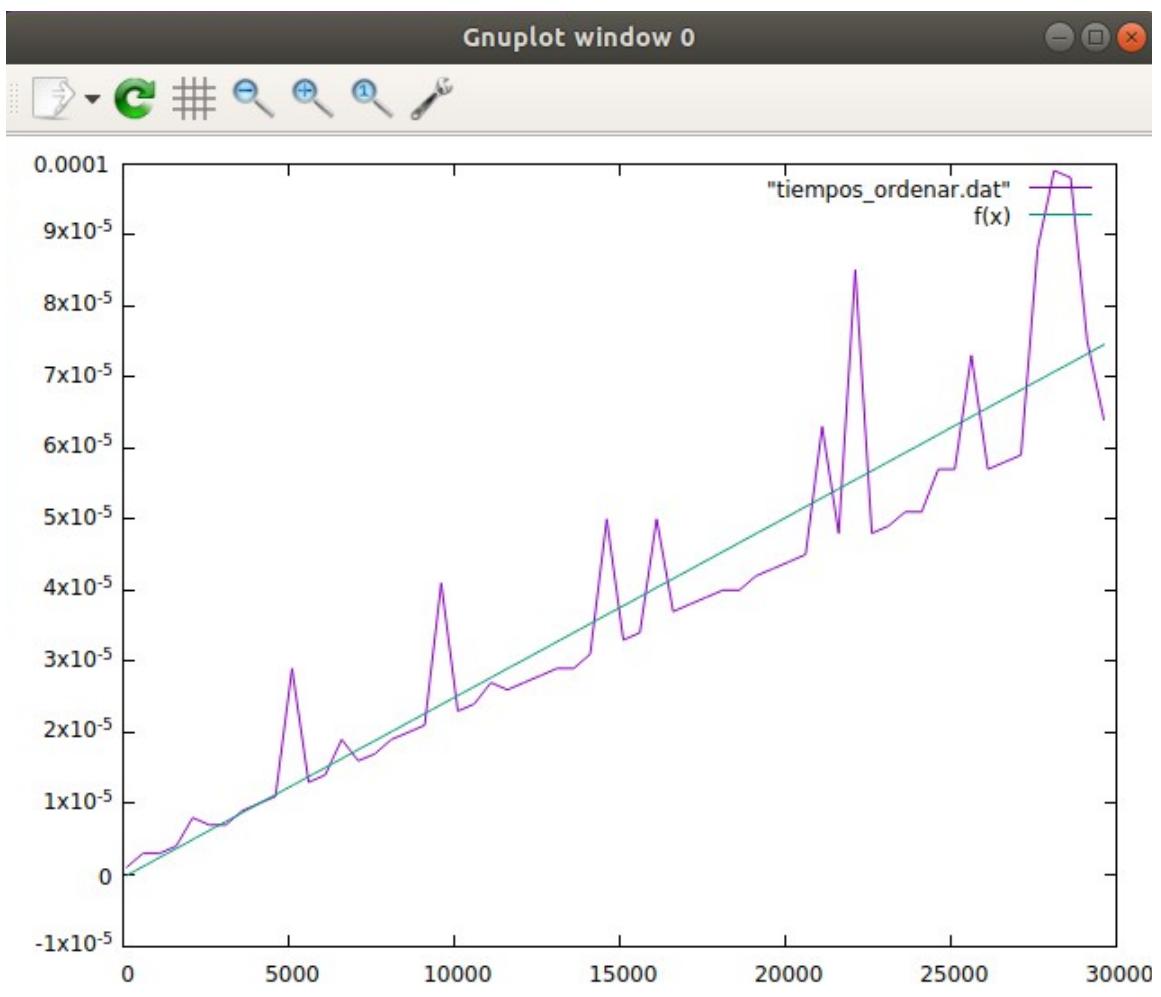
$$2 + 6 + 1 + 5 + \sum_{j=0}^{n-2} (4 + 2) = 14 + 6 \sum_{j=0}^{n-2} 1 = 14 + 6(n - 1) = 6n + 8$$

Tamaño inicial= 100, Tamaño máximo= 30000, incremento= 500, muestras totales=60 .

```
pablo@pablo-VirtualBox: ~/Escritorio/ED/material/creado/ej4
Archivo Editar Ver Buscar Terminal Ayuda
gnuplot> fit f(x) "tiempos_ordenar.dat" via a,b
iter      chisq      delta/lim      lambda      a          b
  0  1.7731881971e+10  0.00e+00  1.22e+04  1.000000e+00  1.000000e+00
  1  1.2111266211e+06 -1.46e+09  1.22e+03  8.214630e-03  9.999501e-01
  2  1.5230684083e+01 -7.95e+09  1.22e+02 -4.955928e-05  9.999393e-01
  3  1.5190953622e+01 -2.62e+02  1.22e+01 -5.019624e-05  9.989101e-01
  4  1.2485422316e+01 -2.17e+04  1.22e+00 -4.550698e-05  9.055980e-01
  5  9.7711574224e-02 -1.27e+07  1.22e-01 -4.023468e-06  8.011336e-02
  6  9.6821789441e-08 -1.01e+11  1.22e-02 -1.373929e-09  7.729517e-05
  7  4.9674962483e-09 -1.85e+06  1.22e-03  2.529498e-09 -3.795222e-07
  8  4.9674962397e-09 -1.74e-04  1.22e-04  2.529536e-09 -3.802760e-07
iter      chisq      delta/lim      lambda      a          b
After 8 iterations the fit converged.
final sum of squares of residuals : 4.9675e-09
rel. change during last iteration : -1.7416e-09

degrees of freedom      (FIT_NDF) : 58
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf) : 9.25454e-06
variance of residuals (reduced chisquare) = WSSR/ndf : 8.56465e-11

Final set of parameters                      Asymptotic Standard Error
=====
a            = 2.52954e-09      +/- 1.38e-10      (5.455%)
b            = -3.80276e-07     +/- 2.372e-06     (623.7%)
correlation matrix of the fit parameters:
      a      b
a    1.000
b   -0.864  1.000
gnuplot>
```



En esta gráfica se observa la representación de los resultados de la forma empírica (morado) y a la función ajustada ($an+b$) (verde). No es el mejor ajuste posible pero tampoco el peor.
Dificultad=4

Ejercicio 5

Código:

```
#include <iostream>
#include <ctime> // Recursos para medir tiempos
#include <cstdlib> // Para generación de números pseudoaleatorios

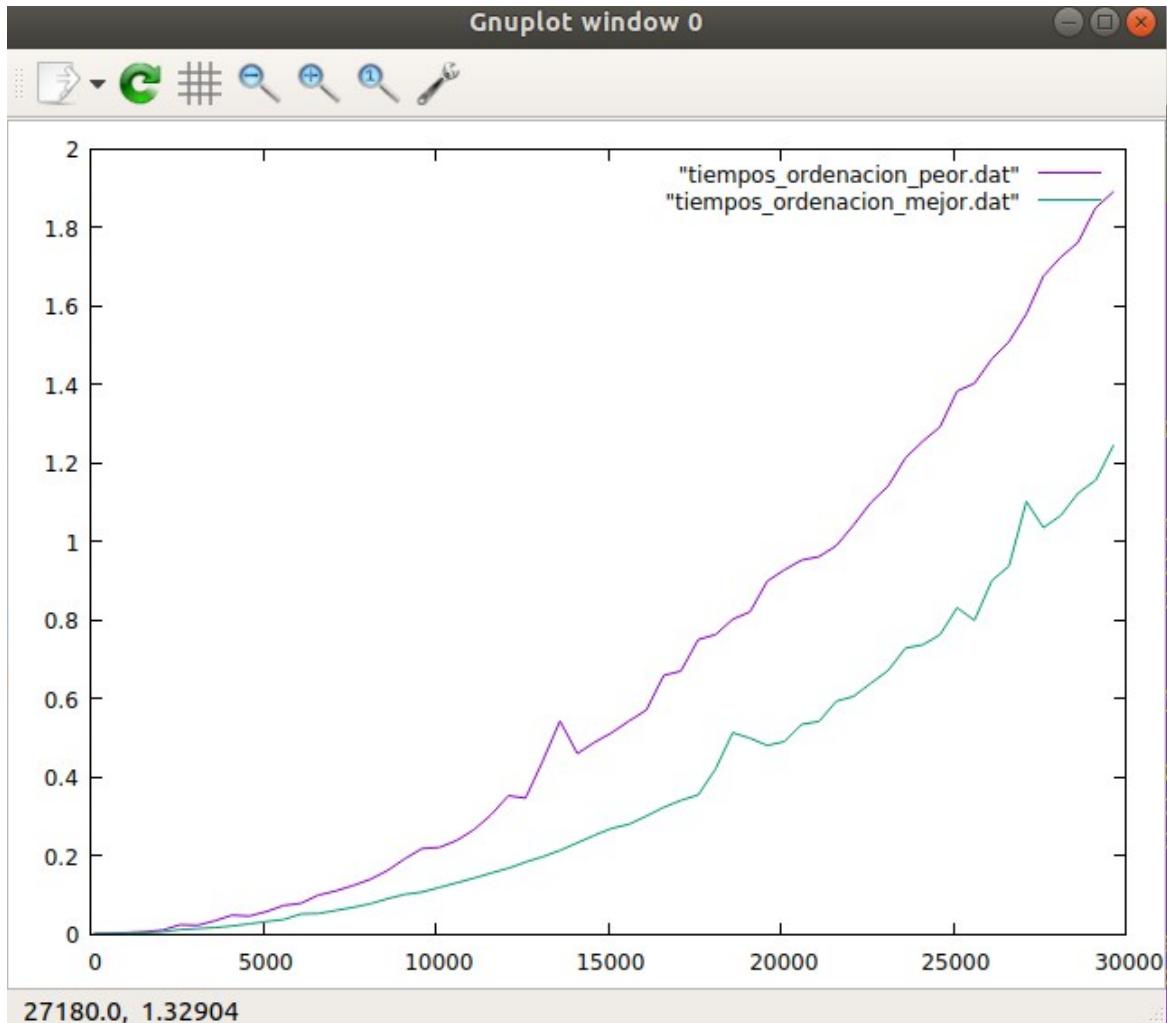
using namespace std;

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;
            }
        }
}
```

Para el mejor caso, el vector tiene que estar ordenado de menor a mayor y por lo tanto no ejecutaría las instrucciones por debajo del if.

Para el peor caso, el vector tiene que estar ordenado de mayor a menor.

Tamaño inicial= 100, Tamaño máximo= 30000, incremento= 500, muestras totales=60 .



Aquí podemos observar los resultados de la forma empírica, el verde representa el mejor caso y el morado el peor y como es lógico el peor caso tarda mas en finalizar la ejecución.

Para obtener el mejor y el peor caso he usado una variable int llamada “cont” en donde, en el mejor caso, lo igualaba a 0 fuera del bucle y dentro de este iba añadiendo una unidad más en sucesivas posiciones. Para el peor es el mismo proceso a la inversa, igualando “cont” al tamaño y decrementando el valor.

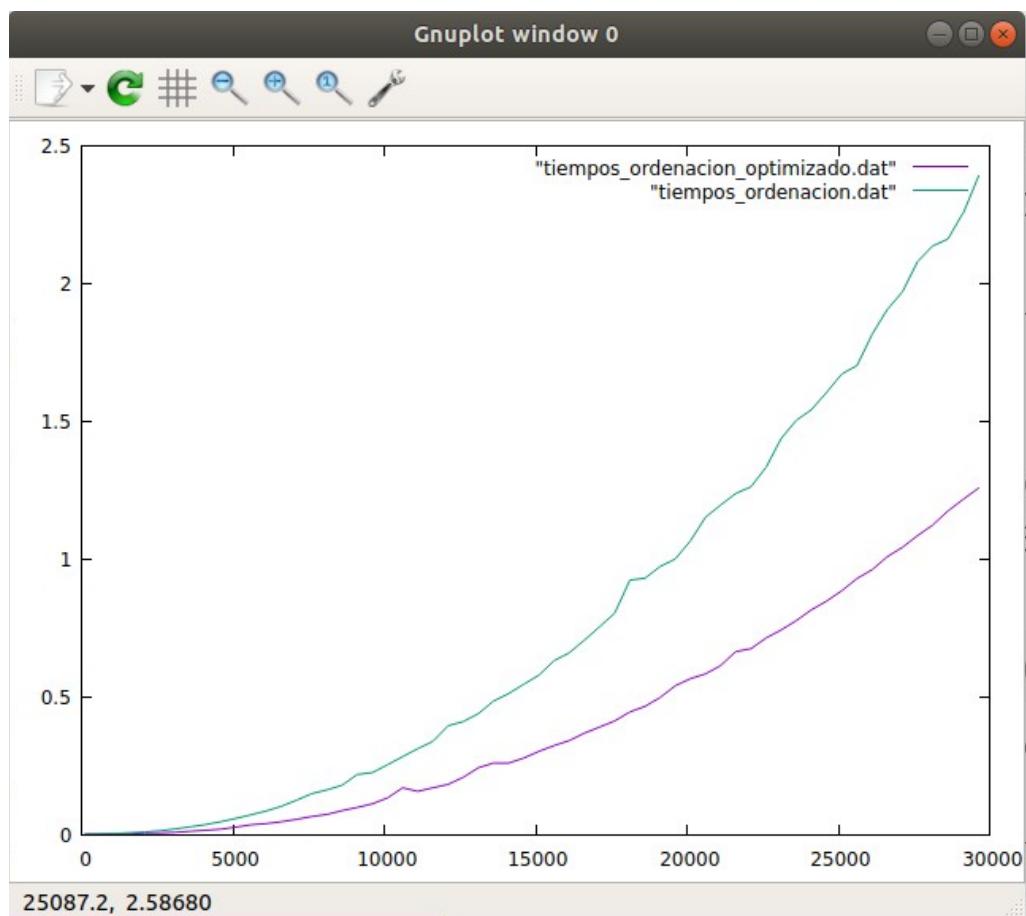
Dificultad=5

Ejercicio 6

Para este ejercicio voy a comparar de manera empírica la diferencia de eficiencia de ejecutar el algoritmo de ordenación burbuja previamente mostrado, indicándole al compilador que optimice el código frente a no indicárselo.

Para ambos:

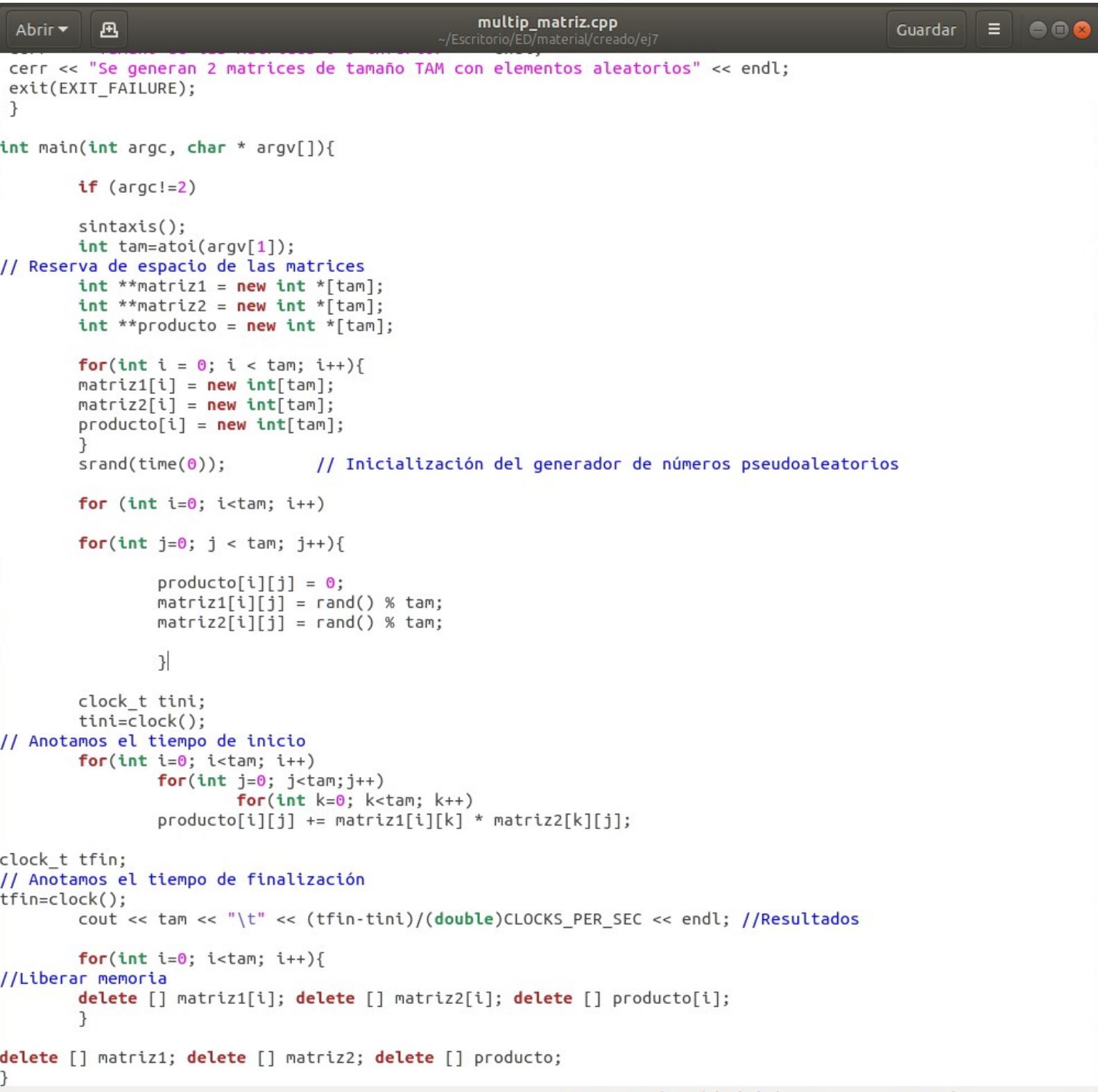
Tamaño inicial= 100, Tamaño máximo= 30000, incremento= 500, muestras totales=60 .



Se observa claramente que el código optimizado (morado) es mas eficiente
Dificultad=4

Ejercicio 7

Código empleado:



```
multip_matriz.cpp
~/Escritorio/ED/material/creado/ej7
Guarda ▾
Abrir ▾
cerr << "Se generan 2 matrices de tamaño TAM con elementos aleatorios" << endl;
exit(EXIT_FAILURE);
}

int main(int argc, char * argv[]){
    if (argc!=2)

        sintaxis();
        int tam=atoi(argv[1]);
    // Reserva de espacio de las matrices
        int **matriz1 = new int *[tam];
        int **matriz2 = new int *[tam];
        int **producto = new int *[tam];

    for(int i = 0; i < tam; i++){
        matriz1[i] = new int[tam];
        matriz2[i] = new int[tam];
        producto[i] = new int[tam];
    }
    srand(time(0));           // Inicialización del generador de números pseudoaleatorios

    for (int i=0; i<tam; i++)
        for(int j=0; j < tam; j++){

            producto[i][j] = 0;
            matriz1[i][j] = rand() % tam;
            matriz2[i][j] = rand() % tam;

        }

    clock_t tini;
    tini=clock();
    // Anotamos el tiempo de inicio
    for(int i=0; i<tam; i++)
        for(int j=0; j<tam;j++)
            for(int k=0; k<tam; k++)
                producto[i][j] += matriz1[i][k] * matriz2[k][j];

    clock_t tfin;
    // Anotamos el tiempo de finalización
    tfin=clock();
    cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl; //Resultados

    for(int i=0; i<tam; i++)
    //Liberar memoria
        delete [] matriz1[i]; delete [] matriz2[i]; delete [] producto[i];
    }

    delete [] matriz1; delete [] matriz2; delete [] producto;
}
```

C++ ▾ Anchura del tabulador: 8 ▾

Ln 40, Col 18 ▾

INS

```

for(int i=0; i<tam; i++)
    for(int j=0; j<tam; j++)
        for(int k=0; k<tam; k++)
            producto[i][j] += matriz1[i][k] * matriz2[k][j];

```

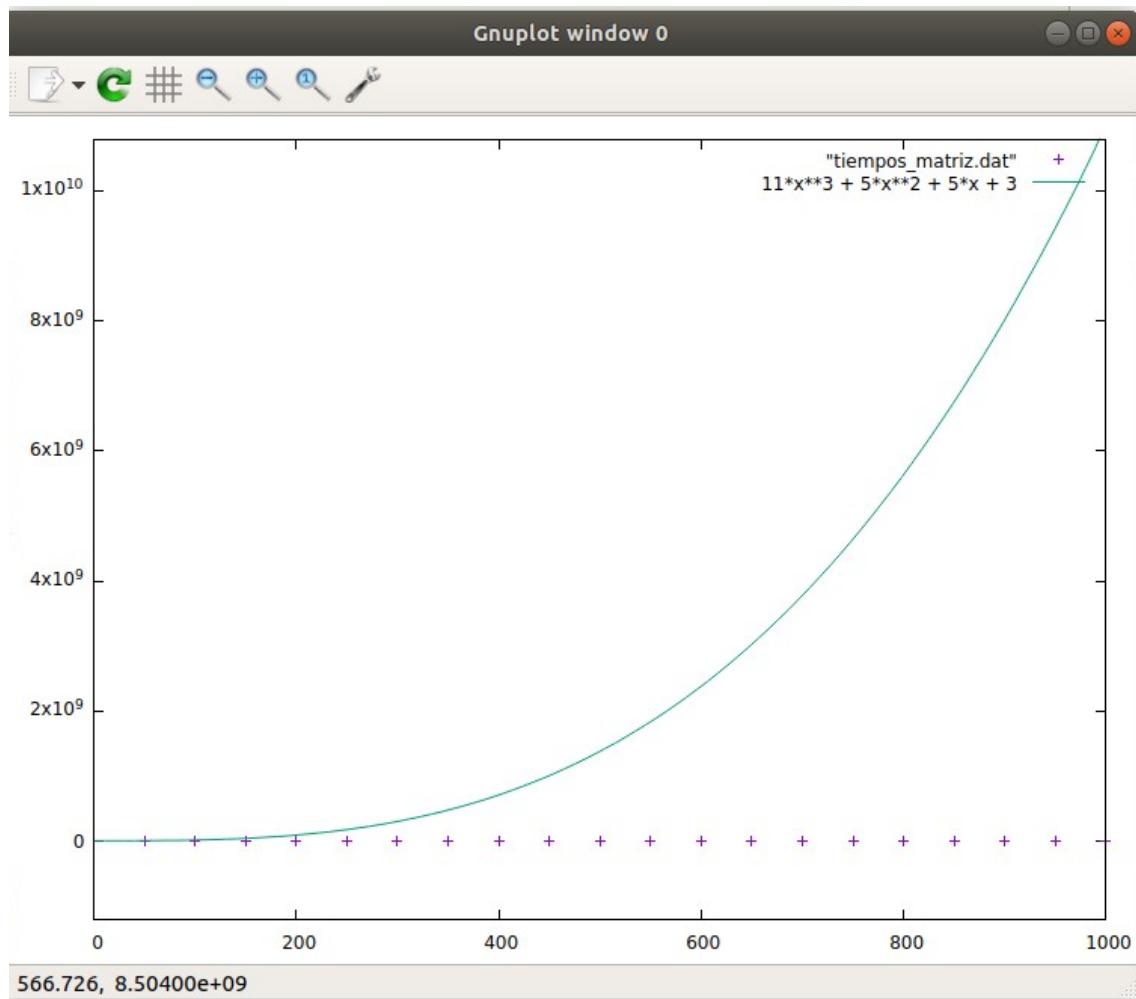
Este es el código para la multiplicación de matrices.

Eficiencia de forma teórica:

- Línea 1: 3 OE (declaración, asignación, comparación)
- Línea 2: 3 OE (declaración, asignación, comparación)
- Línea 3: 3 OE (declaración, asignación, comparación)
- Línea 4: 11 OE (8 accesos, multiplicación, suma, asignación)

$$\begin{aligned}
& 3 + \sum_{i=0}^{n-1} \left(5 + \sum_{j=0}^{n-1} \left(5 + \sum_{k=0}^{n-1} 11 \right) \right) = 3 + \sum_{i=0}^{n-1} \left(5 + \sum_{j=0}^{n-1} (5 + 11n) \right) = \\
& = 3 + \sum_{i=0}^{n-1} \left(5 + \sum_{j=0}^{n-1} 5 + \sum_{j=0}^{n-1} 11n \right) = 3 + \sum_{i=0}^{n-1} \left(5 + 5 \sum_{j=0}^{n-1} 1 + 11n \sum_{j=0}^{n-1} 1 \right) = \\
& = 3 + \sum_{j=0}^{n-1} (5 + 5n + 11n^2) = 3 + \sum_{j=0}^{n-1} 5 + \sum_{j=0}^{n-1} 5n + \sum_{j=0}^{n-1} 11n^2 = 3 + 5n + 5n^2 + 11n^3
\end{aligned}$$

Tamaño inicial= 50, Tamaño máximo= 1000, incremento= 50, muestras totales=20 .



Como es de esperar la función teórica crece más debido a que las OE tardan un segundo.

Archivo Editar Ver Buscar Terminal Ayuda

```
gnuplot> fit f(x) "tiempos_matriz.dat" via a,b,c,d
iter      chisq      delta/lim   lambda   a          b          c          d
 0  3.3898438656e+18  0.00e+00  2.06e+08  1.000000e+00  1.000000e+00  1.000000e+00
 1  5.1679079032e+14 -6.56e+08  2.06e+07  1.122015e-02  9.988703e-01  9.999987e-01  1.000000e+00
 2  1.2497451658e+11 -4.13e+08  2.06e+06  -1.137713e-03  9.985610e-01  9.999978e-01  1.000000e+00
 3  1.1791400673e+11 -5.99e+03  2.06e+05  -1.106586e-03  9.698911e-01  9.999137e-01  9.999998e-01
 4  7.5686455477e+09 -1.46e+06  2.06e+04  -2.788493e-04  2.435333e-01  9.977814e-01  9.999946e-01
 5  1.5821197082e+05 -4.78e+09  2.06e+03  1.067595e-06  -2.100014e-03  9.968919e-01  9.999916e-01
 6  6.9302566386e+04 -1.28e+05  2.06e+02  1.984018e-06  -2.884721e-03  9.802707e-01  9.998704e-01
 7  9.6146551330e+03 -6.21e+05  2.06e+01  7.416447e-07  -1.069141e-03  3.613081e-01  9.953408e-01
 8  1.7342657015e+00 -5.54e+08  2.06e+00  8.656383e-09  2.017024e-06  -3.856616e-03  9.903201e-01
 9  1.0384111336e+00 -6.70e+04  2.06e-01  5.897800e-09  5.577600e-06  -4.588636e-03  7.931560e-01
10  2.1134892529e-01 -3.91e+05  2.06e-02  1.354055e-08  -7.851826e-06  2.303522e-03  -1.536283e-01
11  2.0943706128e-01 -9.13e+02  2.06e-03  1.392592e-08  -8.528973e-06  2.651040e-03  -2.013663e-01
12  2.0943706079e-01 -2.32e-04  2.06e-04  1.392611e-08  -8.529314e-06  2.651215e-03  -2.013904e-01
```

After 12 iterations the fit converged.

final sum of squares of residuals : 0.209437

rel. change during last iteration : -2.32076e-09

```
degrees of freedom      (FIT_NDF)           : 16
rms of residuals        (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.114411
variance of residuals   (reduced chisquare) = WSSR/ndf   : 0.0130898
```

Final set of parameters		Asymptotic Standard Error	
a	= 1.39261e-08	+/- 1.378e-09	(9.894%)
b	= -8.52931e-06	+/- 2.197e-06	(25.76%)
c	= 0.00265121	+/- 0.001006	(37.93%)
d	= -0.20139	+/- 0.125	(62.06%)

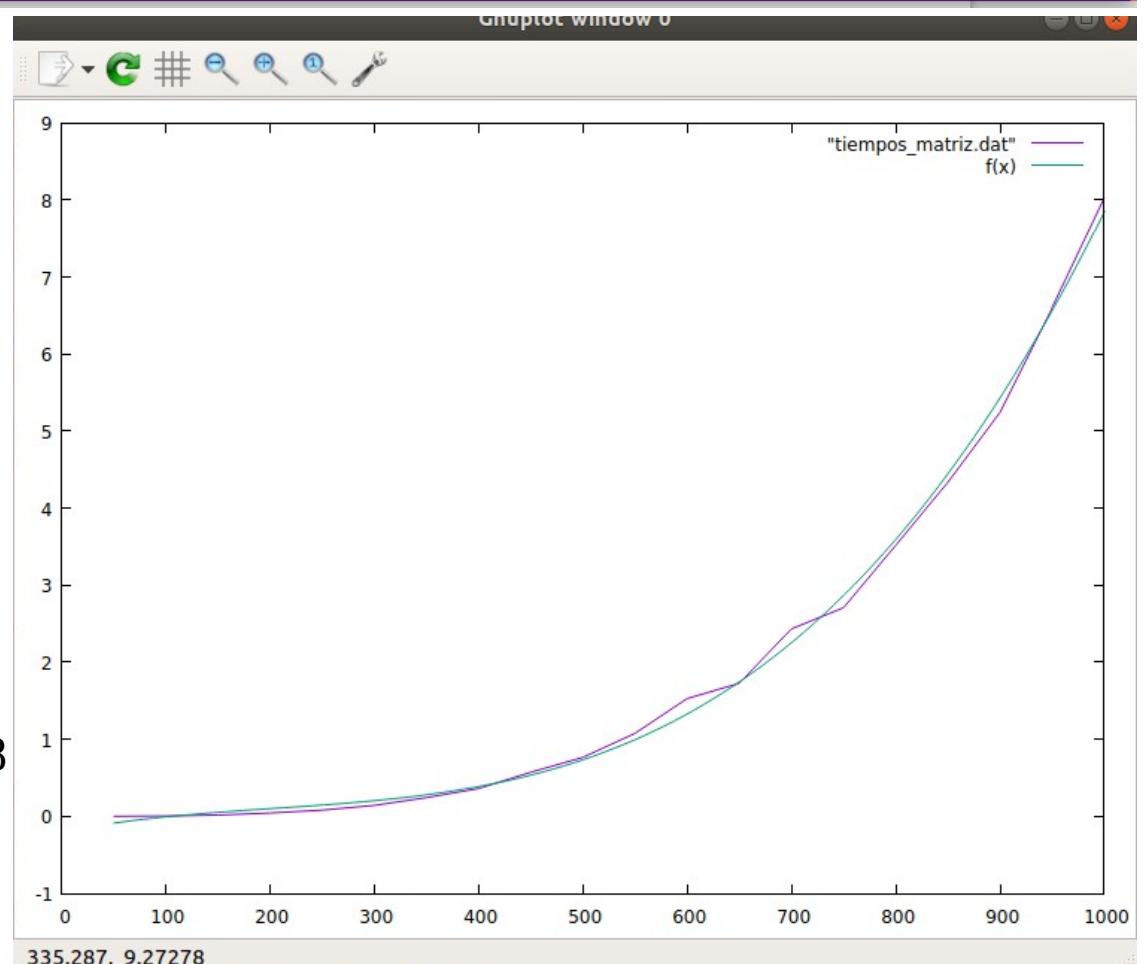
correlation matrix of the fit parameters:

	a	b	c	d
a	1.000			
b	-0.988	1.000		
c	0.929	-0.974	1.000	
d	-0.732	0.807	-0.905	1.000

gnuplot> □

El ajuste
como de los
datos es
bastante
bueno.

Dificultad=8



Ejercicio 8

Código de Mergesort:

```
'const int UMBRAL_MS = 100;

void mergesort(int T[], int num_elem)
{
    mergesort_lims(T, 0, num_elem);
}

static void mergesort_lims(int T[], int inicial, int final)
{
    if (final - inicial < UMBRAL_MS)
    {
        insercion_lims(T, inicial, final);
    } else {
        int k = (final - inicial)/2;

        int * U = new int [k - inicial + 1];
        assert(U);
        int l, l2;
        for (l = 0, l2 = inicial; l < k; l++, l2++)
            U[l] = T[l2];
        U[l] = INT_MAX;

        int * V = new int [final - k + 1];
        assert(V);
        for (l = 0, l2 = k; l < final - k; l++, l2++)
            V[l] = T[l2];
        V[l] = INT_MAX;

        mergesort_lims(U, 0, k);
        mergesort_lims(V, 0, final - k);
        fusion(T, inicial, final, U, V);
        delete [] U;
        delete [] V;
    };
}
```

Eficiencia teórica:

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n; \quad \text{cambio: } n = 2^m \rightarrow m = \log_2(n)$$

$$T(2^m) = 2T\left(\frac{2^m}{2}\right) + 2 * 2^m = 2T(2^{(m-1)}) + 1 * 2^{(m+1)}$$

$$T(2^m) = 2T(2^{(m-1)}) + 1 * 2^{(m+1)}$$

$$T(2^{(m-1)}) = 2T(2^{(m-2)}) + 2^m$$

$$T(2^m) = 2[2T(2^{(m-2)}) + 2^m] + 1 * 2^{(m+1)} = 2^2T(2^{(m-2)}) + 2^{(m+1)} + 2^{(m+1)}$$

$$T(2^m) = 2^2T(2^{(m-2)}) + 2 * 2^{(m+1)}$$

$$T(2^{(m-2)}) = 2T(2^{(m-3)}) + 2 * 2^{(m-2)}$$

$$T(2^m) = 2^2[2T(2^{(m-3)}) + 2 * 2^{(m-2)}] + 2 * 2^{(m+1)}$$

$$T(2^m) = 2^3T(2^{(m-3)}) + 2^3 * 2^{(m-2)} + 2 * 2^{(m+1)}$$

$$T(2^m) = 2^3T(2^{(m-3)}) + 2^{(m+1)} + 2 * 2^{(m+1)}$$

$$T(2^m) = 2^3T(2^{(m-3)}) + 3 * 2^{(m+1)}$$

$$T(2^m) = 2^kT(2^{(m-k)}) + k * 2^{(m+1)}$$

$$\text{Para } k = m \quad T(2^m) = 2^mT(1) + m * 2^{(m+1)}$$

$$T(n) = n + 2n\log_2(n)$$

Pertenece a $O(n * \log_2(n))$

Código de Inserción:

```
static void insercion_lims(int T[], int inicial, int final)
{
    int i, j;
    int aux;
    for (i = inicial + 1; i < final; i++) {
        j = i;
        while ((T[j] < T[j-1]) && (j > 0)) {
            aux = T[j];
            T[j] = T[j-1];
            T[j-1] = aux;
            j--;
        };
    };
}
```

Eficiencia teórica:

$$\begin{aligned} 3 + 3 + \sum_{i=0}^n \left(1 + 6 + \sum_{i=0}^n (2 + 4 + 3 + 2) + 2 \right) &= 6 + \sum_{i=0}^n \left(9 + \sum_{i=0}^n 11 \right) = 6 + \sum_{i=0}^n (9 + 11n + 11) = \\ &= 6 + \sum_{i=0}^n (11n + 20) = 6 + 11n(n+1) + 20(n+1) = 6 + 11n^2 + 11n + 20n + 20 = \\ &= 11n^2 + 31n + 26 \end{aligned}$$

Pertenece a $O(n^2)$

Tamaño inicial= 100, Tamaño máximo=2400, incremento= 100, muestras totales=24 .

Inserción prueba de eficiencia de forma empírica:

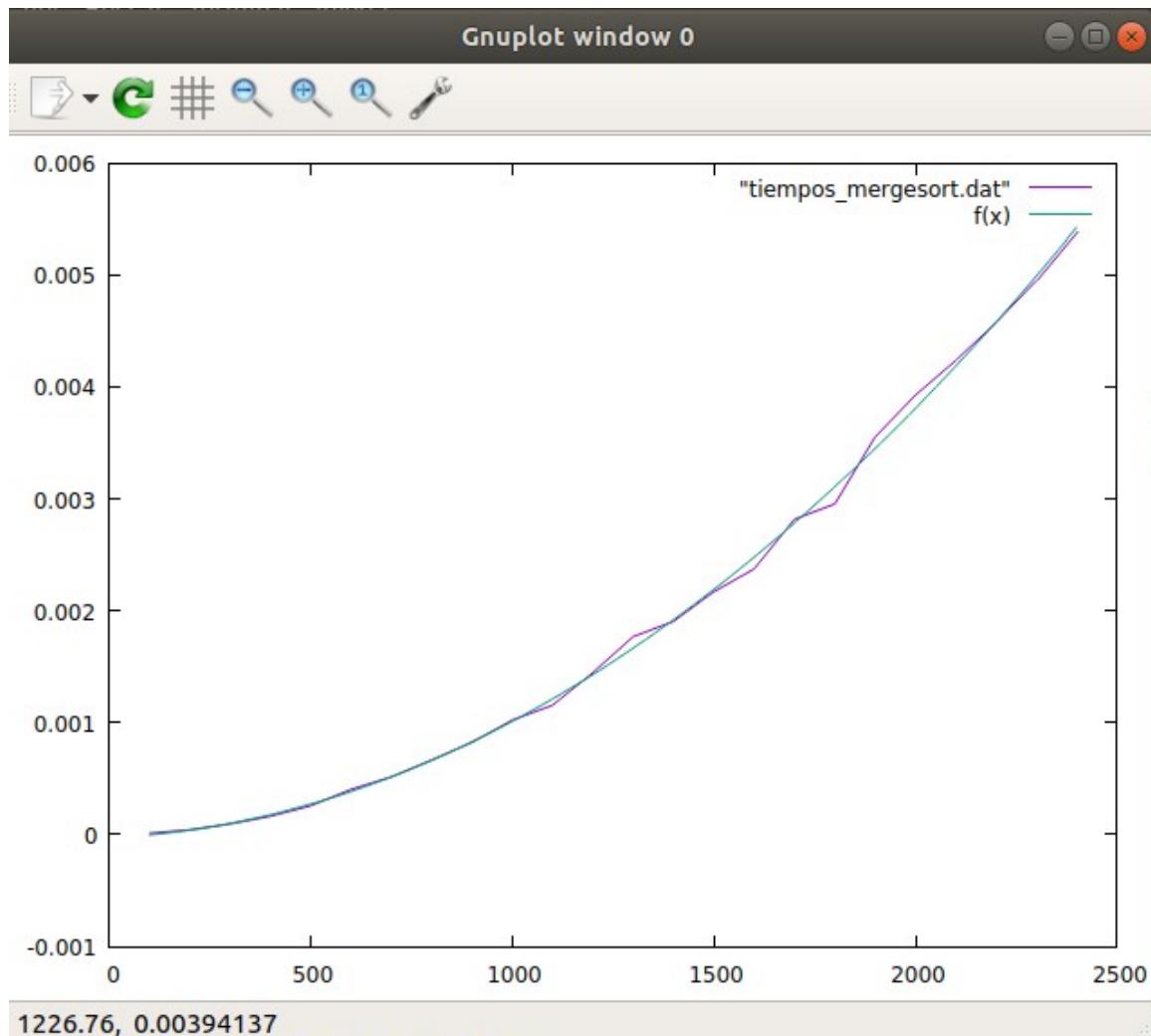
```
pablo@pablo-VirtualBox: ~/Escritorio/ED/material/creado/ej8

Archivo Editar Ver Buscar Terminal Ayuda
gnuplot> fit f(x) "tiempos_mergesort.dat" via a,b,c
iter      chisq      delta/lim      lambda      a          b          c
 0 1.7648214672e+14  0.00e+00  1.56e+06  1.000000e+00  1.000000e+00  1.000000e+00
 1 3.3120373839e+10 -5.33e+08  1.56e+05  1.319512e-02  9.994950e-01  9.999997e-01
 2 3.0628306591e+06 -1.08e+09  1.56e+04  -5.085362e-04  9.993631e-01  9.999995e-01
 3 2.9871619691e+06 -2.53e+03  1.56e+03  -5.041405e-04  9.870238e-01  9.999794e-01
 4 5.9106303667e+05 -4.05e+05  1.56e+02  -2.239450e-04  4.381463e-01  9.990830e-01
 5 3.9581901136e+01 -1.49e+09  1.56e+01  -1.229577e-06  1.866991e-03  9.982796e-01
 6 2.1982545464e+00 -1.70e+06  1.56e+00  5.496981e-07  -1.613507e-03  9.891987e-01
 7 5.9793174194e-01 -2.68e+05  1.56e-01  2.871845e-07  -8.415771e-04  5.158920e-01
 8 6.9602409864e-05 -8.59e+08  1.56e-02  3.966851e-09  -8.917587e-06  5.534434e-03
 9 8.1498257364e-08 -8.53e+07  1.56e-03  8.800172e-10  1.577011e-07  -2.803371e-05
10 8.1497431513e-08 -1.01e+00  1.56e-04  8.796808e-10  1.586904e-07  -2.864003e-05
11 8.1497431513e-08 -8.12e-11  1.56e-05  8.796808e-10  1.586904e-07  -2.864003e-05
iter      chisq      delta/lim      lambda      a          b          c
After 11 iterations the fit converged.
final sum of squares of residuals : 8.14974e-08
rel. change during last iteration : -8.11982e-16

degrees of freedom      (FIT_NDF)                  : 21
rms of residuals        (FIT_STDFIT) = sqrt(WSSR/ndf)   : 6.22963e-05
variance of residuals   (reduced chisquare) = WSSR/ndf   : 3.88083e-09

Final set of parameters            Asymptotic Standard Error
=====
a      = 8.79681e-10      +/- 2.975e-11      (3.382%)
b      = 1.5869e-07      +/- 7.661e-08      (48.27%)
c      = -2.864e-05      +/- 4.156e-05      (145.1%)

correlation matrix of the fit parameters:
      a      b      c
a      1.000
b      -0.971  1.000
c      0.775 -0.885  1.000
gnuplot> □
```



Es un ajuste muy bueno donde los datos están representados con la curva morada y la curva verde representa una función de segundo grado ajustada

Tamaño inicial= 100, Tamaño máximo=10000, incremento= 100, muestras totales=100 .

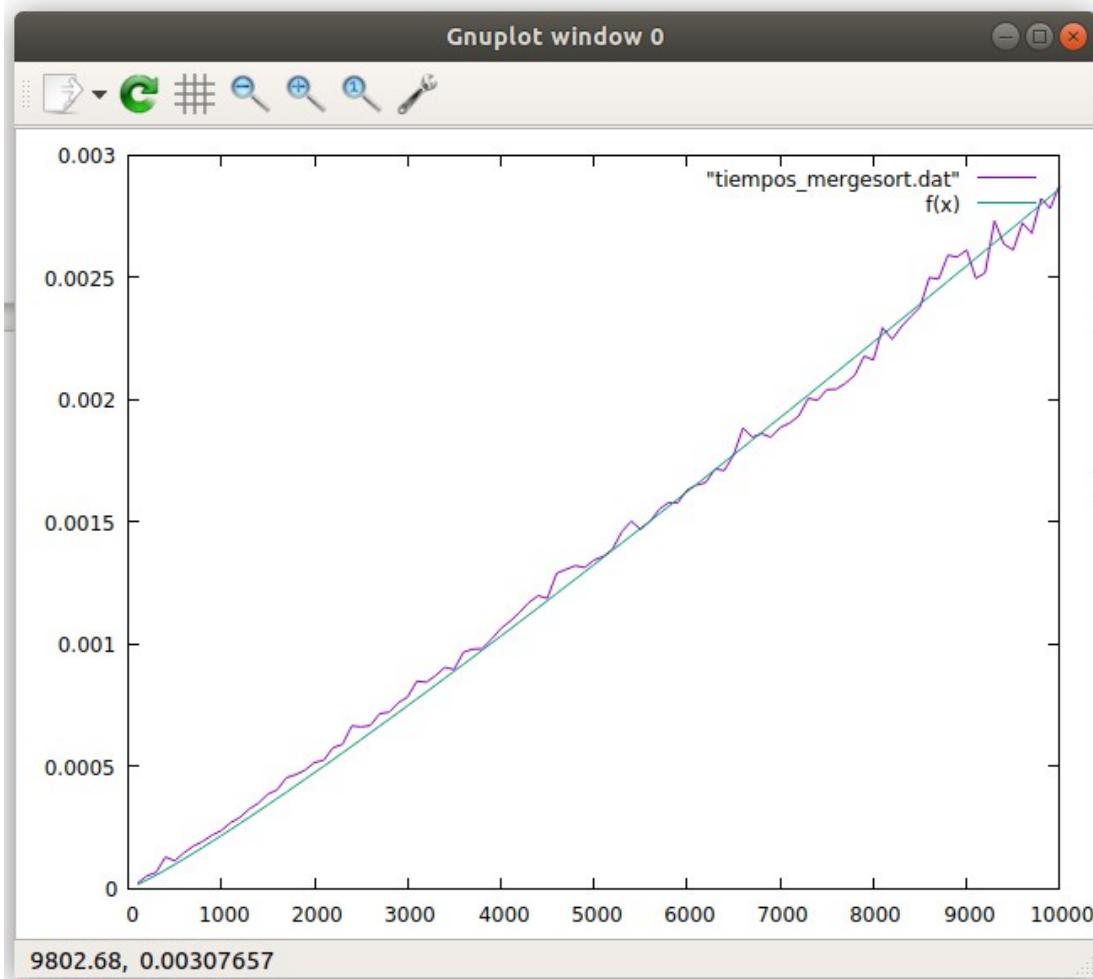
Mergesort prueba de eficiencia de forma empírica:

```
pablo@pablo-VirtualBox: ~/Escritorio/ED/material/creado/ej8
Archivo Editar Ver Buscar Terminal Ayuda
gnuplot> f(x)= a*x*(log(x)/log(2))
gnuplot> fit f(x) "tiempos_mergesort.dat" via a
iter      chisq      delta/lim   lambda   a
0 5.5634659737e+11  0.00e+00  7.46e+04  1.000000e+00
1 5.4538437148e+07 -1.02e+09  7.46e+03  9.901011e-03
2 5.4527551865e-01 -1.00e+13  7.46e+02  1.011532e-06
3 2.0853715765e-07 -2.61e+11  7.46e+01  2.153296e-08
4 2.0853661238e-07 -2.61e-01  7.46e+00  2.153197e-08
iter      chisq      delta/lim   lambda   a

After 4 iterations the fit converged.
final sum of squares of residuals : 2.08537e-07
rel. change during last iteration : -2.61476e-06

degrees of freedom      (FIT_NDF) : 99
rms of residuals        (FIT_STDFIT) = sqrt(WSSR/ndf) : 4.58959e-05
variance of residuals   (reduced chisquare) = WSSR/ndf : 2.10643e-09

Final set of parameters          Asymptotic Standard Error
=====
a      = 2.1532e-08      +/- 6.153e-11      (0.2858%)
gnuplot>
```

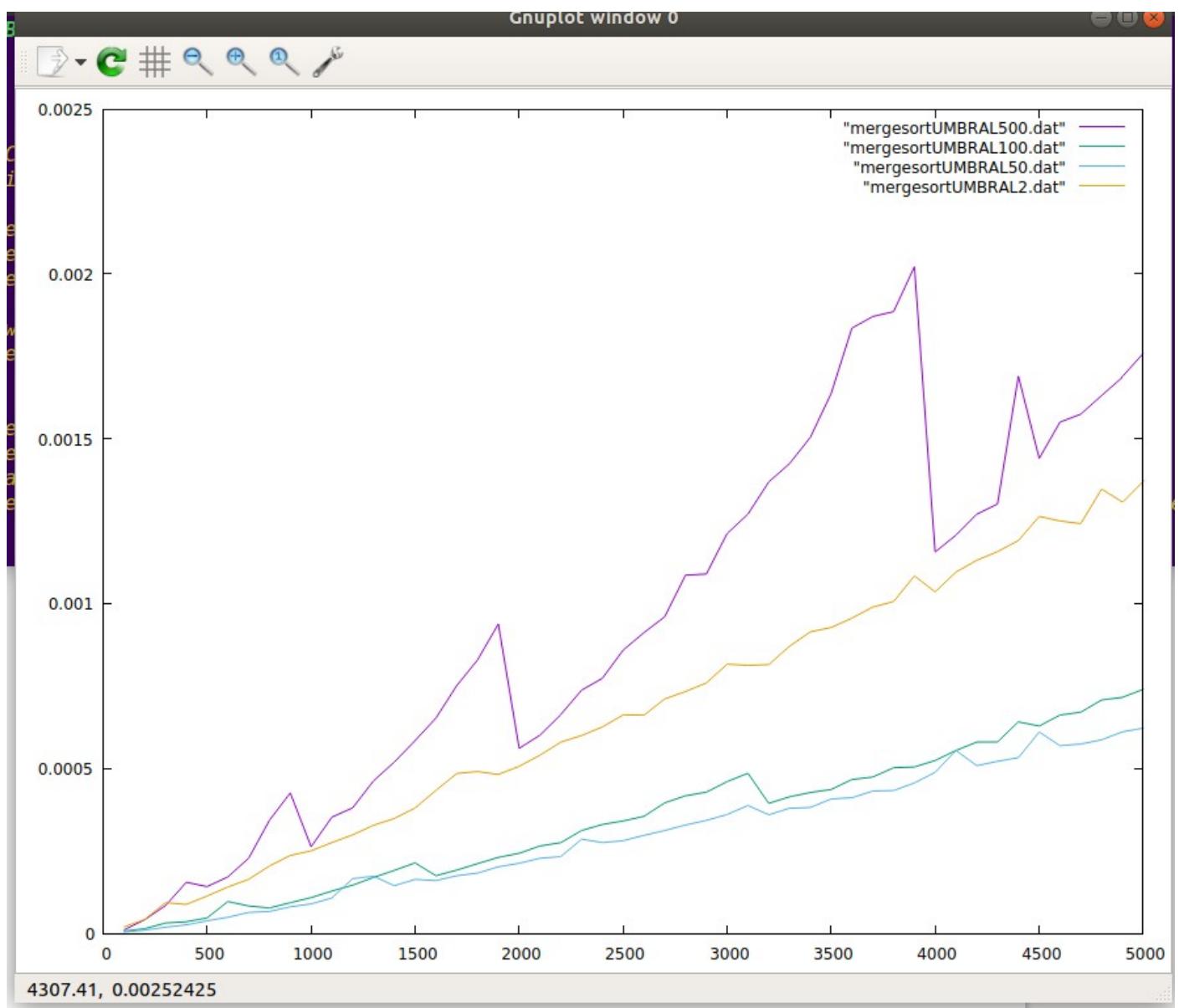


El ajuste en este caso también es bastante bueno.
Se aprecia la eficiencia de este código ($O(n \log_2 n)$) frente al de inserción ($O(n^2)$).

Ahora haré algunas pruebas con UMBRAL_MS para observar como reacciona a su incremento o decrementos

UMBRAL_MS=500, 100, 50, 2

Tamaño inicial= 100, Tamaño máximo=5000, incremento= 100,
muestras totales=100 .



Al parecer es mas eficiente cuando UMBRAL_MS vale 50 o 100, eso puede deberse a que insercion es mas eficiente para tamaños más pequeños por eso la mezcla de ambos algoritmos ha causado que UMBRAL_MS 50 y 100 sean los mas eficientes.

Dificultad=10

FIN