

# PRACTICA 1. ESTRUCTURA DE DATOS

Ángel Solano Corral

Javier Ramirez Pulido

## CARACTERISTICAS DEL ORDENADOR

SO: Ubuntu 18.04

Procesador: Intel Core i7 – 8750H 2.2GHz

Memoria RAM: 16 gb

Sección1: Ejercicio 1: Ordenación de la burbuja. Valoración: 5

Eficiencia Teórica

$$\sum_{i=0}^{n-2} \sum_{j=i}^{n-2-i} 8 = \sum_{i=0}^{n-2} 8(n-i-1) = 8n \sum_{i=0}^{n-2} 1 + 8 \sum_{i=0}^{n-2} 1 = 8 \sum_{i=0}^{n-2} i =$$
$$= 8n(n-1) + 8(n-1) - 8 \left( \frac{n^2-n}{2} \right) = 4n^2 + 4n - 8 \in O(n^2)$$

Pass	Time (s)
100	2.3e-05
600	0.000623
1100	0.001096
1600	0.003889
2100	0.006849
2600	0.010819
3100	0.015677
3600	0.021412
4100	0.028732
4600	0.037032
5100	0.046967
5600	0.058804
6100	0.069807
6600	0.083493
7100	0.098057
7600	0.114932
8100	0.132016
8600	0.150632
9100	0.170846
9600	0.198548
10100	0.216068
10600	0.241201
11100	0.263487
11600	0.296759
12100	0.326195
12600	0.34868
13100	0.378254
13600	0.409091
14100	0.450003
14600	0.47898
15100	0.514135
15600	0.55039
16100	0.586879
16600	0.63109
17100	0.664887
17600	0.723693
18100	0.751716
18600	0.796497
19100	0.845107
19600	0.887568
20100	0.944513
20600	0.982876
21100	1.03572
21600	1.08436
22100	1.14027
22600	1.19628
23100	1.2641

Actividades Editor de textos

sab 20/20

tiempos\_ordenacion.dat

Guardar

```
6100 0.959887
6400 0.983493
7100 0.998957
7600 0.114932
8100 0.132916
8600 0.158632
9100 0.172846
9600 0.198548
10100 0.21668
10600 0.241201
11100 0.263487
11600 0.296759
12100 0.326195
12600 0.34868
13100 0.378254
13600 0.409091
14100 0.439063
14600 0.47898
15100 0.514135
15600 0.55839
16100 0.586879
16600 0.63189
17100 0.664887
17600 0.723693
18100 0.751716
18600 0.796497
19100 0.845187
19600 0.887568
20100 0.944513
20600 0.982876
21100 1.03572
21600 1.08436
22100 1.14827
22600 1.19628
23100 1.26411
23600 1.38579
24100 1.3776
24600 1.43668
25100 1.48936
25600 1.54786
26100 1.62133
26600 1.69245
27100 1.73895
27600 1.82156
28100 1.90072
28600 1.95888
29100 2.0299
29600 2.09218
```

Actividades Geany

sab 20/22

ordenacion.cpp - /home/angel/Escritorio/javipruebas - Geany

```
1 #include <iostream>
2 #include <stdlib.h>
3 #include <string>
4 #include <string>
5 using namespace std;
6 void ordenar(int *v, int n) {
7     for(int i=0; i < n-1; i++){
8         for(int j=0; j < n-i-1; j++){
9             if(v[j]>v[j+1]){
10                 int aux = v[j];
11                 v[j] = v[j+1];
12                 v[j+1] = aux;
13             }
14         }
15     }
16 }
17
18 int main (int argc, char *argv[]) {
19     if (argc != 2)
20         exit(0);
21
22     clock_t t_ini, t_fin, t_final;
23
24     int util = atoi(argv[1]);
25     int *v = new int[util];
26     int x;
27     for (int i = 0; i < util; i++){
28         v[i] = rand() % util;
29     }
30 }
```

Estado 20:21:46: Esto es Geany 1.32.

Compilador 20:21:46: Archivo /home/angel/Descargas/scd-pl-fuentes/prodcons-plantilla.cpp abierto(1)

Mensajes 20:21:46: Archivo /home/angel/Escritorio/javipruebas/ordenacion.cpp abierto(2)

Borrador

Terminal

línea: 1 / 49 col: 0 sel: 0 INS TAB mode: LF codificación: UTF-8 tipo de archivo: C++ ámbito: desconocido

Actividades Geany

sab 20/22

ordenacion.cpp - /home/angel/Escritorio/javipruebas - Geany

```
14 int aux = v[j];
15 v[j] = v[j+1];
16 v[j+1] = aux;
17 }
18 }
19
20 int main (int argc, char *argv[]) {
21     if (argc != 2)
22         exit(0);
23
24     clock_t t_ini, t_fin, t_final;
25
26     int util = atoi(argv[1]);
27     int *v = new int[util];
28     int x;
29     for (int i = 0; i < util; i++){
30         v[i] = rand() % util;
31     }
32     t_ini = clock();
33     ordenar(v, util);
34     t_fin = clock();
35     cout << util << "\t" << (t_fin-t_ini)/(double)CLOCKS_PER_SEC << endl;
36 }
37
38
39
40
41
42
43
44
45
46
47
48
49 }
```

Estado 20:21:46: Esto es Geany 1.32.

Compilador 20:21:46: Archivo /home/angel/Descargas/scd-pl-fuentes/prodcons-plantilla.cpp abierto(1)

Mensajes 20:21:46: Archivo /home/angel/Escritorio/javipruebas/ordenacion.cpp abierto(2)

Borrador

Terminal

línea: 1 / 49 col: 0 sel: 0 INS TAB mode: LF codificación: UTF-8 tipo de archivo: C++ ámbito: desconocido

Actividades Editor de textos sab 2022

ejecuciones\_ordenacion.ch

Guardar

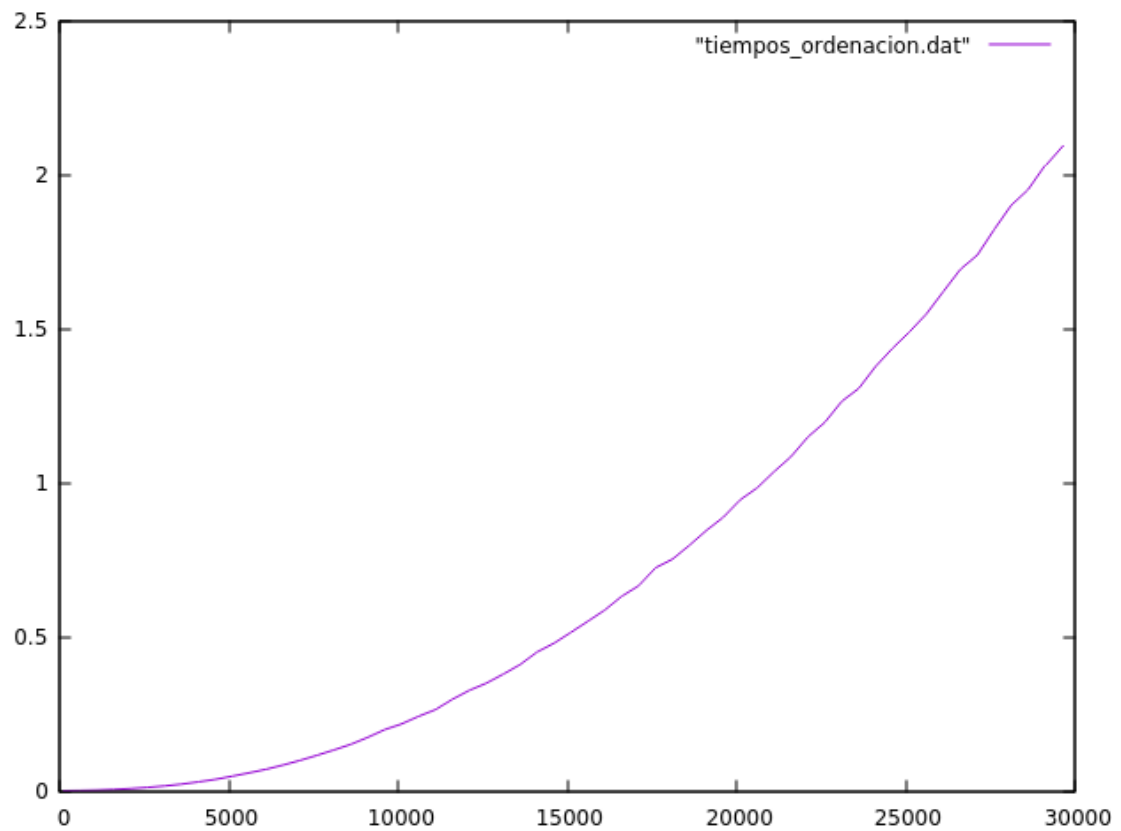
tiempos\_ordenacion.dat

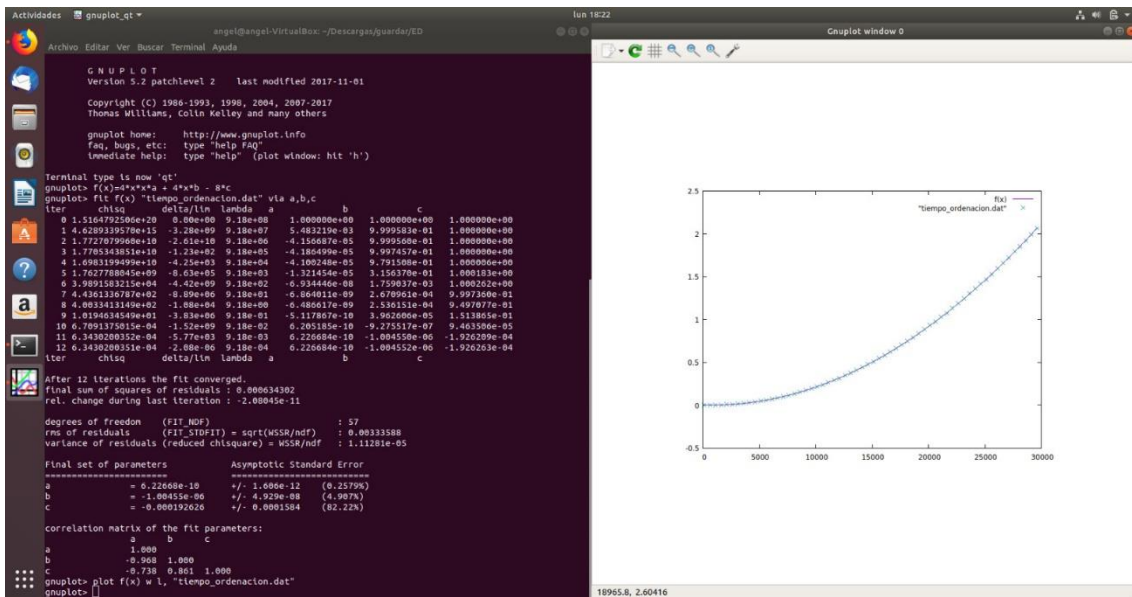
ejecuciones\_ordenacion.ch

```
#!/bin/csh
@ inicio = 100
@ fin = 30000
@ incremento = 500
set ejecutable = ordenacion
set salida = tiempos_ordenacion.dat

@ i = $inicio
echo > $salida
while ( $i <= $fin )
    echo Ejecución tan = $i
    echo ./{Ejecutable} $i' >> $salida
    @ i += $incremento
end
```

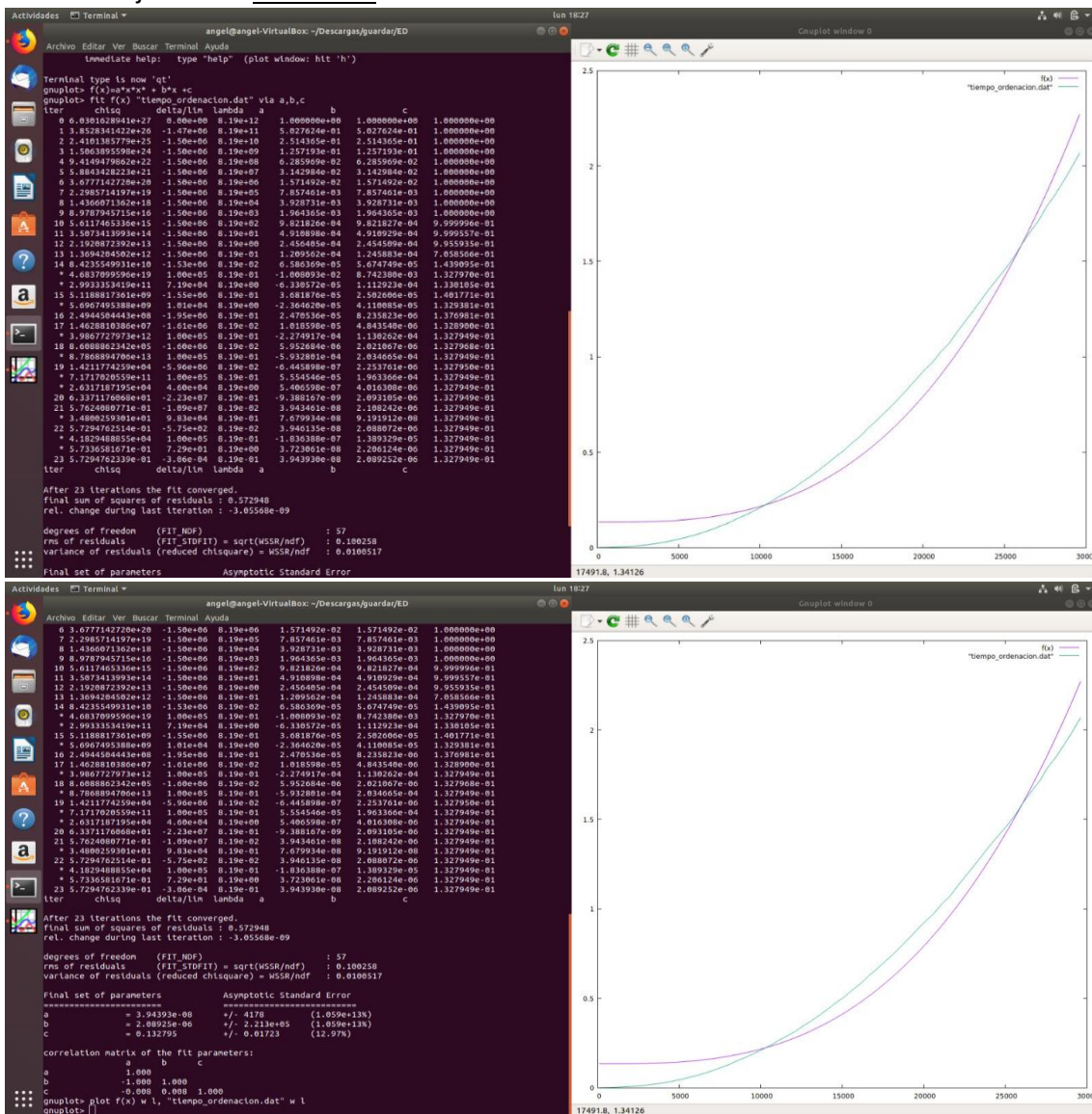
sh Anchura del tabulador: 8 Ln 10, Col 21 INS





Se ajusta bastante a la eficiencia teórica calculada.

## Sección2: Ejercicio 2. Valoración: 4

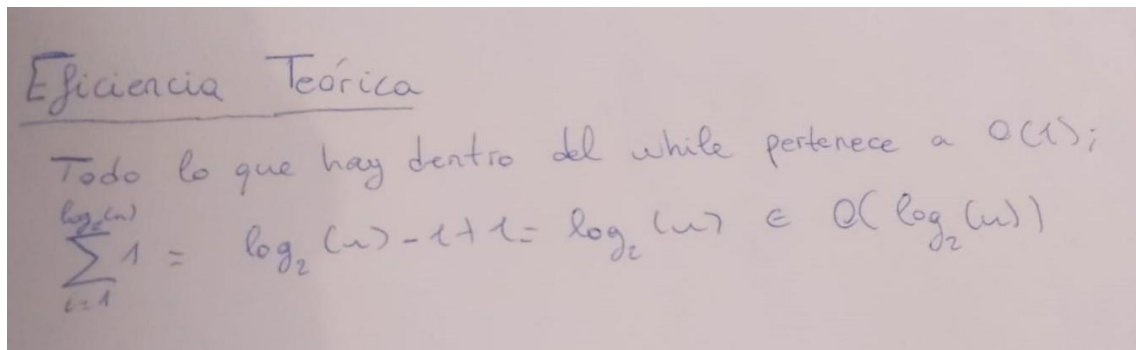


### Sección3. Ejercicio 3: Problemas de precisión. Valoración: 6

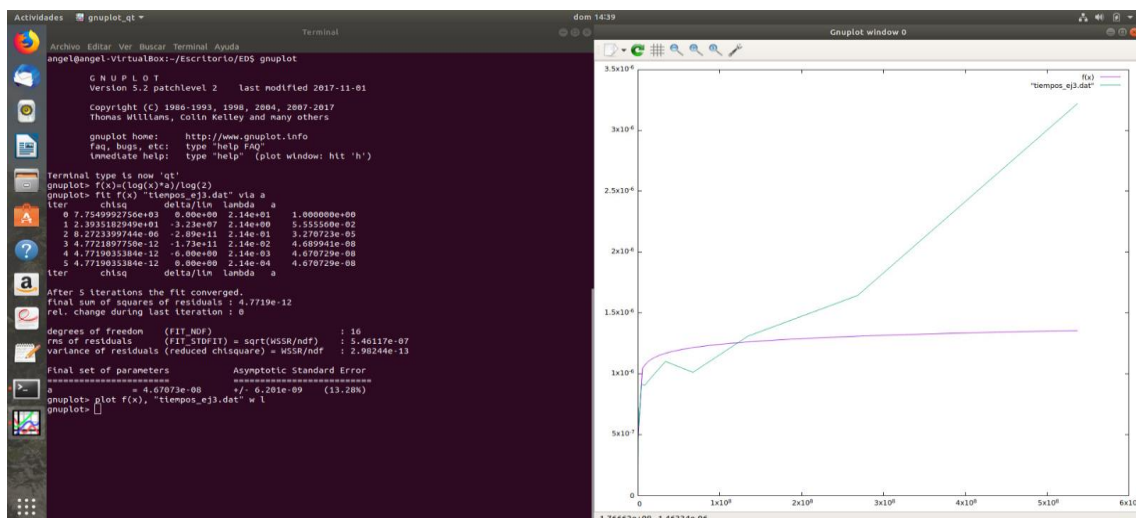
#### ¿Qué hace este algoritmo?

Este algoritmo genera un vector dinámico aleatorio. Cuenta con una función llamada **operacion** (**int \*v, int n, int x, int inf, int sup**) que busca el elemento 'x' entre las posiciones 'inf' y 'sup' del vector (realiza una búsqueda binaria). Si se encuentra el elemento 'x', se devuelve la posición en la que se ha encontrado; en caso contrario, se devuelve un -1. Dentro del **main** nos indica que hay que pasarle 3 parámetros; el primero el ejecutable, segundo el tamaño del vector y tercero el valor máximo que pueden tomar los elementos del vector. Generará un vector aleatorio con el tamaño indicado y con valores  $\leq$  que el valor máximo, y se aplicará la función operacion (de la cual se medirá el tiempo transcurrido). Por último, nos mostrará el tiempo que tarda en realizarse la función respecto al tamaño vector. En caso de que el número de argumentos pasados no sea correcto, se llamará a una función **sintaxis ()** que, tras mostrar un texto de ayuda, hará que el programa finalice.

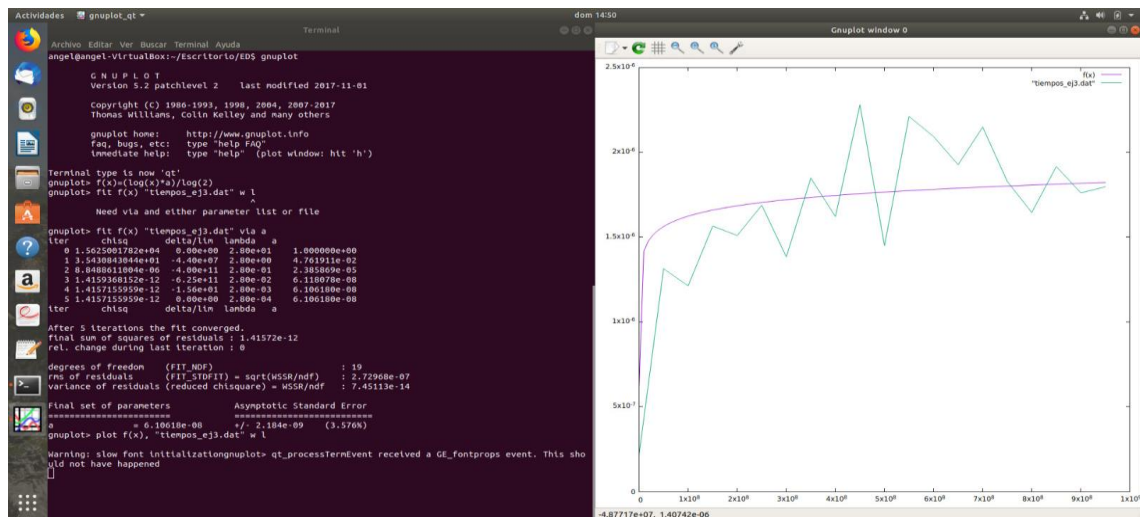
#### Eficiencia teórica



#### Eficiencia empírica



#### Forma lineal



Con los incrementos de potencia de 2, cuanto mayor tamaño de entrada, mayor es el tiempo de ejecución y con los incrementos lineales, esto no se cumple. El tiempo de ejecución va incrementando, pero no de forma constante.

#### Sección4. Ejercicio 4: Dependencia de la implementación. Valoración: 5

##### Programa que ejecute el algoritmo de la burbuja

Para hacer el mejor caso posible lo que hemos hecho es ordenar previamente y calcular el tiempo de la función ordenación con el vector ya ordenado.

Actividades | Geany

dom 15:06

ordenacion2.cpp - /home/angel/Escritorio/ED - Geany

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <chrono>
5  #include <cstdlib>
6
7  using namespace std;
8
9  // Función para ordenar un vector de enteros
10 void ordenar(vector<int> &v) {
11     int n = v.size();
12     for (int i = 0; i < n-1; i++) {
13         for (int j = i+1; j < n; j++) {
14             if (v[i] > v[j]) {
15                 swap(v[i], v[j]);
16             }
17         }
18     }
19 }
20
21 int main(int argc, char *argv[]) {
22     if (argc != 2) {
23         exit(0);
24     }
25
26     clock_t t_ini, t_fin, t_final;
27     int util = atoi(argv[1]);
28     int *v = new int[util];
29     int i;
30
31     for (int i = 0; i < util; i++) {
32         v[i] = rand() % util;
33     }
34
35     ordenar(v, util);
36     t_ini = clock();
37     ordenar(v, util);
38     t_fin = clock();
39
40     cout << util << "x" << (t_fin-t_ini)/(double)CLOCKS_PER_SEC << endl;
41 }
  
```

Estado 15:06:17: Esto es Geany 1.32.

Compilador 15:06:17: Archivo /home/angel/Descargas/scd-g1-fuentes/prodcons-plantilla.cpp abierto(1)

Mensajes 15:06:17: Archivo /home/angel/Escritorio/javipruebas/ordenacion.cpp abierto(2)

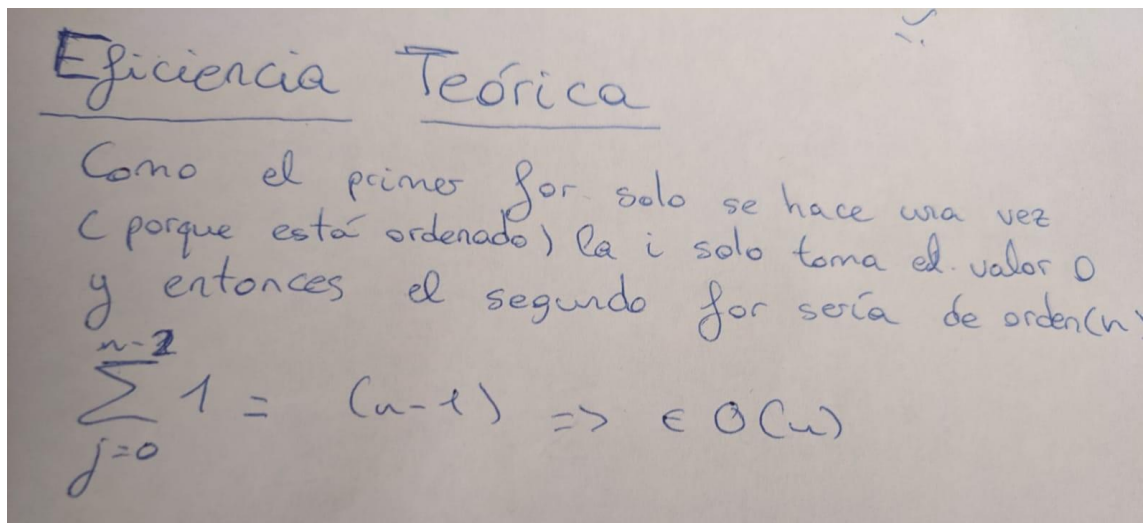
Borrador 15:06:17: Archivo /home/angel/Escritorio/ED/ordenacion2.cpp abierto(3)

Terminal

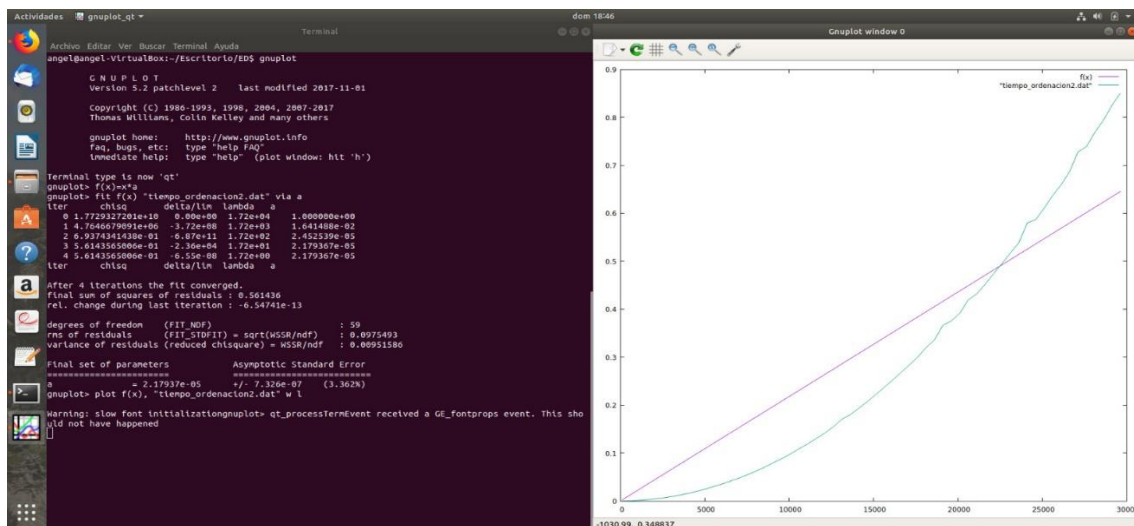
Esto es Geany 1.32.



## Eficiencia teórica mejor caso



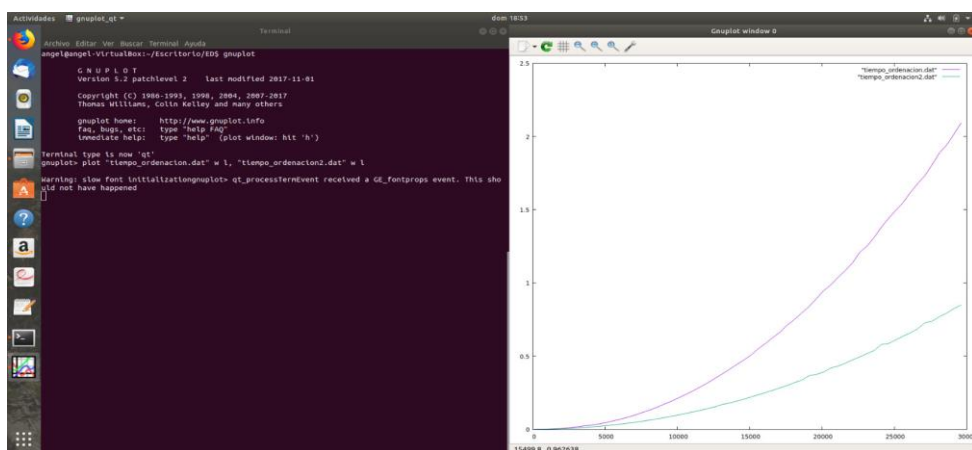
## Eficiencia empírica mejor caso



Vemos que no se ajusta mucho a la previsión.

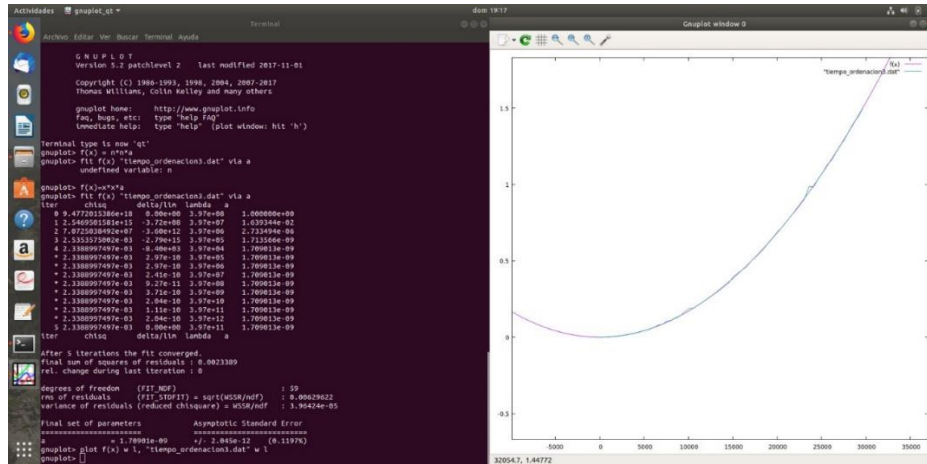
## Sección 5. Ejercicio 5: Mejor y peor caso. Valoración: 5

### Comparación del mejor caso con el resultado del ejercicio 1



Para el peor caso, lo que hemos hecho ha sido ordenar el vector en orden decreciente y luego hemos calculado el tiempo que tarda en ejecutarse la función ordenación con el vector ordenado al revés.

### Eficiencia empírica peor caso

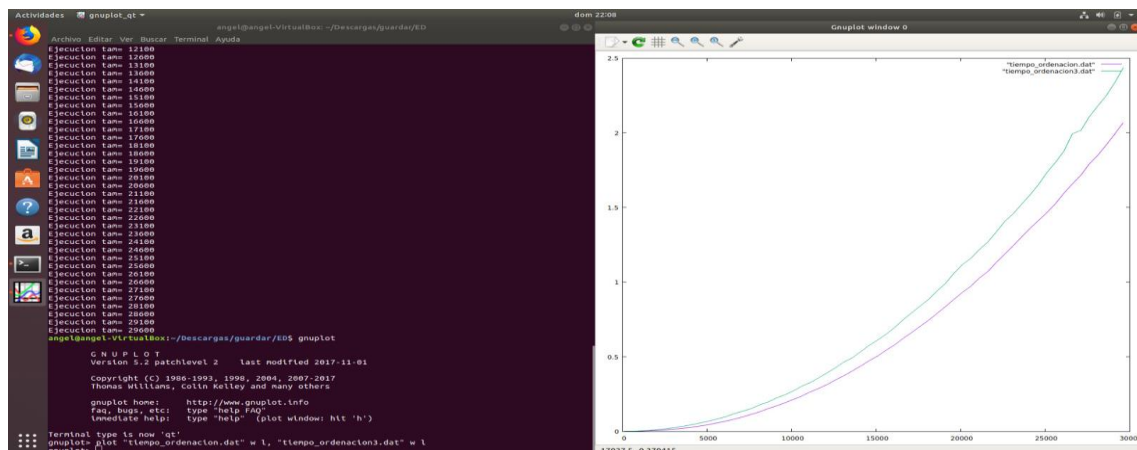


### Eficiencia teórica peor caso

Eficiencia Teórica

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

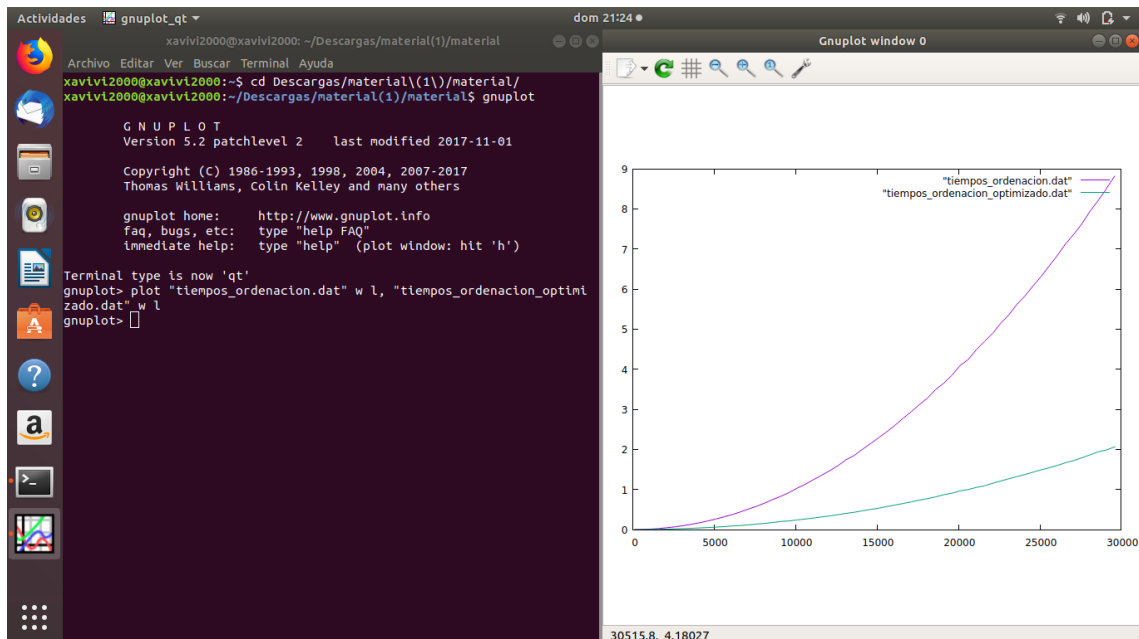
### Comparación del peor caso con el resultado del ejercicio 1





## Sección6. Ejercicio 6: Influencia del proceso de compilación. Valoración: 4

Vemos claramente que, con el código optimizado, los tiempos de ejecución se reducen considerablemente.

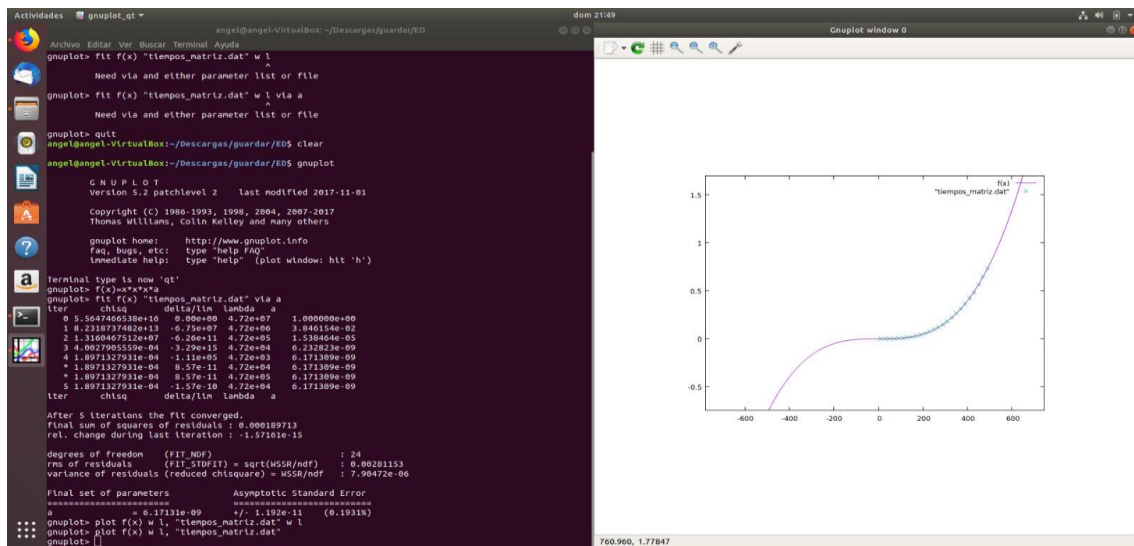


## Sección7. Ejercicio 7: Multiplicación matricial. Valoración: 8

Eficiencia teórica

$$\text{Eficiencia Teórica}$$
$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = n \cdot n \cdot n = n^3 \in O(n^3)$$

## Eficiencia empírica



## Programa de matriz

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

using namespace std;

class Matriz{
private:
    int num_filas;
    int num_columnas;
    int **matriz;
public:
    Matriz(){
        num_filas = 0;
        num_columnas = 0;
        matriz = new int*[0];
        matriz[0] = new int[0];
    }
    Matriz(int filas, int columnas){
        if(filas == columnas){
            num_filas = filas;
            num_columnas = columnas;
            matriz = new int*[num_filas];
            for (int i = 0; i < num_filas; i++){
                matriz[i] = new int[num_columnas];
            }
        }
        else{
            num_filas = 0;
            num_columnas = 0;
            matriz = new int*[0];
            matriz[0] = new int[0];
        }
    }
    void hacer_matriz(){
        for (int i = 0; i < num_filas; i++){
            for (int j = 0; j < num_columnas; j++){
                matriz[i][j] = rand() % 6 + 1;
            }
        }
    }
    int getFilas(){
        return num_filas;
    }
    int getCols(){
        return num_columnas;
    }
    int getPos(int fila, int columna){
        return matriz[fila][columna];
    }
    int setPos(int fila, int col, int valor){
        matriz[fila][col] = valor;
    }
    Matriz multiplicar(Matriz m2){
        int aux = 0;
        Matriz resultado(num_filas, num_columnas);
        int n = num_filas;
        for (int i = 0; i < n; i++){
            for (int j = 0; j < n; j++){
                aux = 0;
                for (int k = 0; k < n; k++){
                    aux += matriz[i][k]*m2.getPos(k,j);
                }
                resultado.setPos(i,j,aux);
            }
        }
        return resultado;
    }
}
```



```
dom 21:53
*matrices.cpp
Guardar

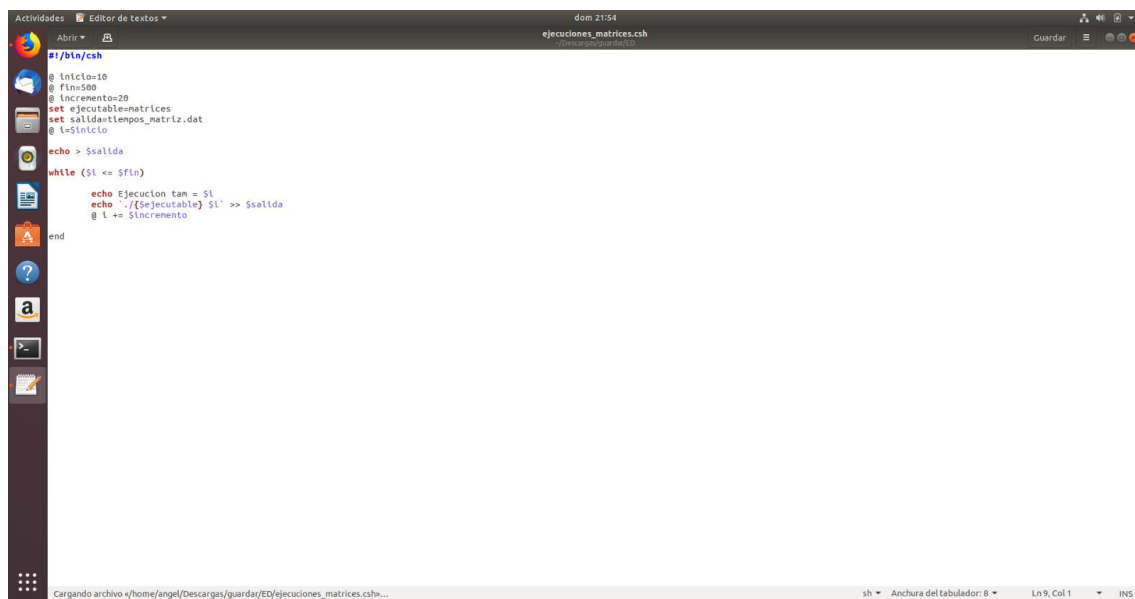
// ...
    }
    resultado.setPos(i,j,aux);
}
return resultado;
}

int main (int argc, char * argv[]) {
    if (argc != 2)
        exit(0);
    int num = atoi(argv[1]);
    Matriz m1(num,num), m2(num,num), resultado;

    clock_t t_ini, t_fin, t_resultado;
    m1.hacer_matriz();
    m2.hacer_matriz();
    t_ini = clock();
    resultado = m1.multiplicar(m2);
    t_fin = clock();
    cout << "    " << num << "    " << (t_fin-t_ini)/(double)CLOCKS_PER_SEC << endl;
}
```

C++ Anchura del tabulador: 8 Ln 96, Col 3 INS

## Script de matriz



```
dom 21:54
ejecuciones_matrices.csh
Guardar

#!/bin/csh
@ inicio=10
@ fin=500
@ incremento=20
set ejecutable=matrices
set salida=tiempos_matriz.dat
@ i=$inicio

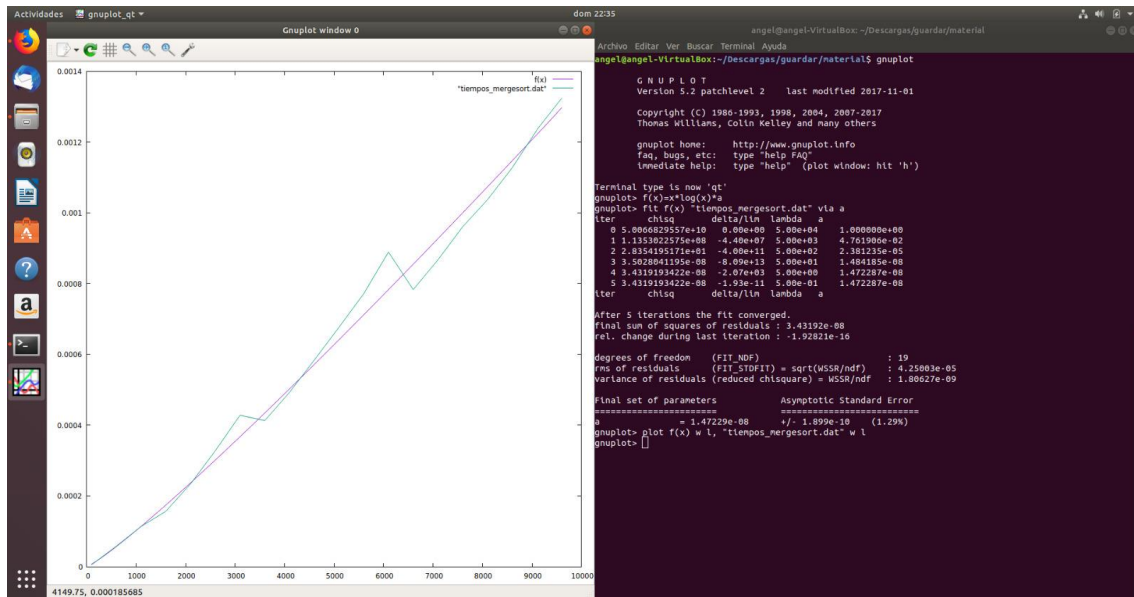
echo > $salida
while ($i <= $fin)
    echo Ejecucion tam = $i
    echo -f{ejecutable} $i' >> $salida
    @ i += $incremento
end

Cargando archivo «/home/angel/Descargas/guarda/ED/ejecuciones_matrices.csh»...
sh Anchura del tabulador: 8 Ln 9, Col 1 INS
```

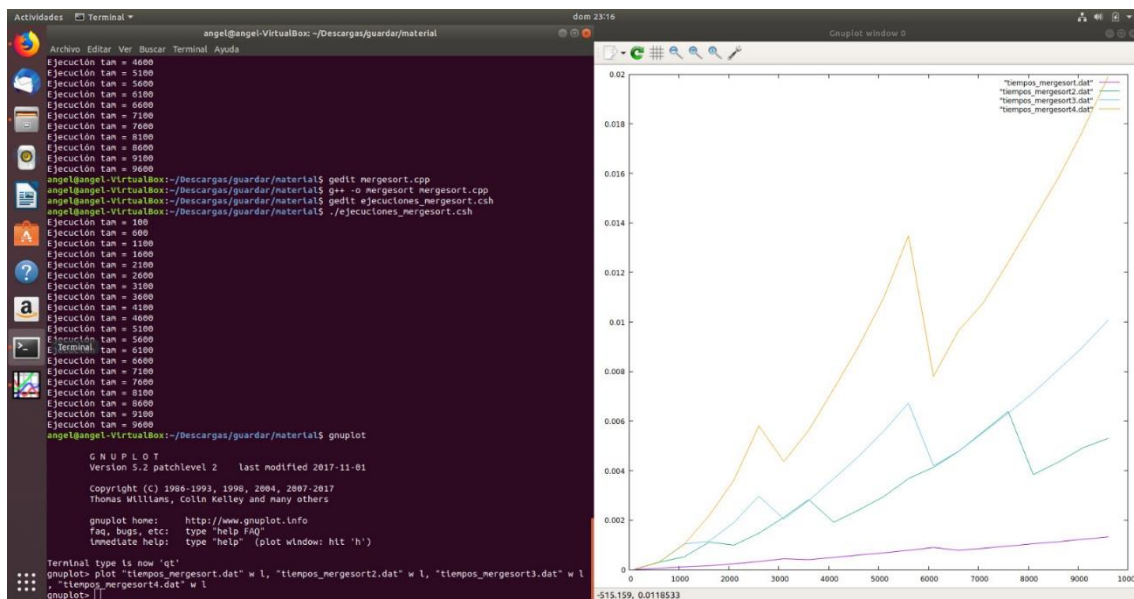
Vemos que la eficiencia empírica se ajusta bastante a la teórica.

## Sección8. Ejercicio 8: Ordenación por Mezcla. Valoración: 7

### Eficiencia empírica



### Modificación de UMBRAL\_MS



A mayor valor de UMBRAL\_MS, más tiempo tarda en ejecutar el programa.

