

# EJERCICIO 4 : Mejor y peor caso

Autor : Jesús Ruiz Castellano, 76439001-L

## 1.- Código fuente : [ordenacion\\_peorcaso.cpp](#)

```
1 #include <iostream>
2 #include <ctime>      // Recursos para medir tiempos
3 #include <cstdlib>    // Para generación de números pseudoaleatorios
4
5 using namespace std;
6
7 void ordenar(int *v, int n) {           // Algoritmo ordenacion por burbuja
8     for (int i=0; i < n-1; i++)
9         for (int j=0; j < n-i-1; j++)
10             if (v[j] > v[j+1]) {
11                 int aux = v[j];
12                 v[j] = v[j+1];
13                 v[j+1] = aux;
14             }
15 }
16
17
18 void sintaxis()
19 {
20     cerr << "Sintaxis:" << endl;
21     cerr << " TAM: Tamaño del vector (>0)" << endl;
22     cerr << " VMAX: Valor máximo (>0)" << endl;
23     cerr << "Se genera un vector de tamaño TAM con elementos aleatorios en [0,VMAX]" << endl;
24     exit(EXIT_FAILURE);
25 }
26
27 int main(int argc, char * argv[])
28 {
29     // Lectura de parámetros
30     if (argc!=3)
31         sintaxis();
32     int tam=atoi(argv[1]);    // Tamaño del vector
33     int vmax=atoi(argv[2]);  // Valor máximo
34     if (tam<=0 || vmax<=0)
35         sintaxis();
36
37     // Generación del vector aleatorio
38     int *v=new int[tam];       // Reserva de memoria
39     srand(time(0));            // Inicialización del generador de números pseudoaleatorios
40     for (int i=0; i<tam; i++)  // Recorrer vector
41         v[i] = rand() % vmax;  // Generar aleatorio [0,vmax[
42
43
44     for (int i=0; i < tam-1; i++)
45         for (int j=0; j < tam-i-1; j++)
46             if (v[j] < v[j+1]) { // ordenamos el vector de manera contraria
47                 int aux = v[j];
48                 v[j] = v[j+1];
49                 v[j+1] = aux;
50             }
51
52     clock_t tini;    // Anotamos el tiempo de inicio
53     tini=clock();
54
55     for (int i = 0 ; i < 1000 ; i++)
56         ordenar(v,tam);    // v esta ordenado de mayor a menor, que es el peor caso
57
58     clock_t tfin;    // Anotamos el tiempo de finalización
59     tfin=clock();
60
61     // Mostramos resultados reduciendo el error en 1000
62     cout << tam << "\t" << ((tfin-tini)/(double)CLOCKS_PER_SEC)/1000.0 << endl;
63
64     delete [] v;    // Liberamos memoria dinámica
65 }
```

## EJERCICIO 4 : Mejor y peor caso

### 2.- Código fuente : [ordenacion\\_mejorcaso.cpp](#)

La única diferencia en el código es que, en el main(), se ordena el vector de menor a mayor antes de llamar a la función ordenar.

```
37 // Generación del vector aleatorio
38 int *v=new int[tam]; // Reserva de memoria
39 srand(time(0)); // Inicialización del generador de números pseudoaleatorios
40 for (int i=0; i<tam; i++) // Recorrer vector
41     v[i] = rand() % vmax; // Generar aleatorio [0,vmax[
42
43
44 for (int i=0; i < tam-1; i++)
45     for (int j=0; j < tam-i-1; j++)
46         if (v[j] > v[j+1]) { // ordenamos el vector de menor a mayor
47             int aux = v[j];
48             v[j] = v[j+1];
49             v[j+1] = aux;
50         }
51
52 clock_t tini; // Anotamos el tiempo de inicio
53 tini=clock();
54
55 for (int i = 0 ; i < 1000 ; i++)
56     ordenar(v,tam); // v esta ordenado de menor a mayor, que es el mejor caso
57
58 clock_t tfin; // Anotamos el tiempo de finalización
59 tfin=clock();
60
61 // Mostramos resultados reduciendo el error en 1000
62 cout << tam << "\t" << ((tfin-tini)/(double)CLOCKS_PER_SEC)/1000.0 << endl;
63
64 delete [] v; // Liberamos memoria dinámica
65 }
```

### 3.- Hardware usado:

#### 3.1- CPU

**vendor\_id** : GenuineIntel  
**model name** : Intel(R) Core(TM) i3 CPU M 330 @ 2.13GHz  
**cpu MHz** : 933.000

#### 3.2- Velocidad de Reloj

**Versión** : hwclock de util-linux 2.20.1  
mie 12 oct 2016 00:10:40 CEST -0.563168 segundos

#### 3.3- Memoria RAM

**MemTotal** : 3907668 kB  
**SwapTotal** : 4049916 kB

## EJERCICIO 4 : Mejor y peor caso

### 4.- Sistema Operativo

Ubuntu 14.04.3 LTS

Arquitectura : x86\_64 (64 bits)

### 5.- Compilador usado y opciones de compilación

gcc - GNU project C and C++ compiler

Opción de compilación : `g++ -o <nombre_ejecutable> <ejecutable.cpp>`

`g++ -o ordenacionMejor ordenacion_mejorcaso.cpp`

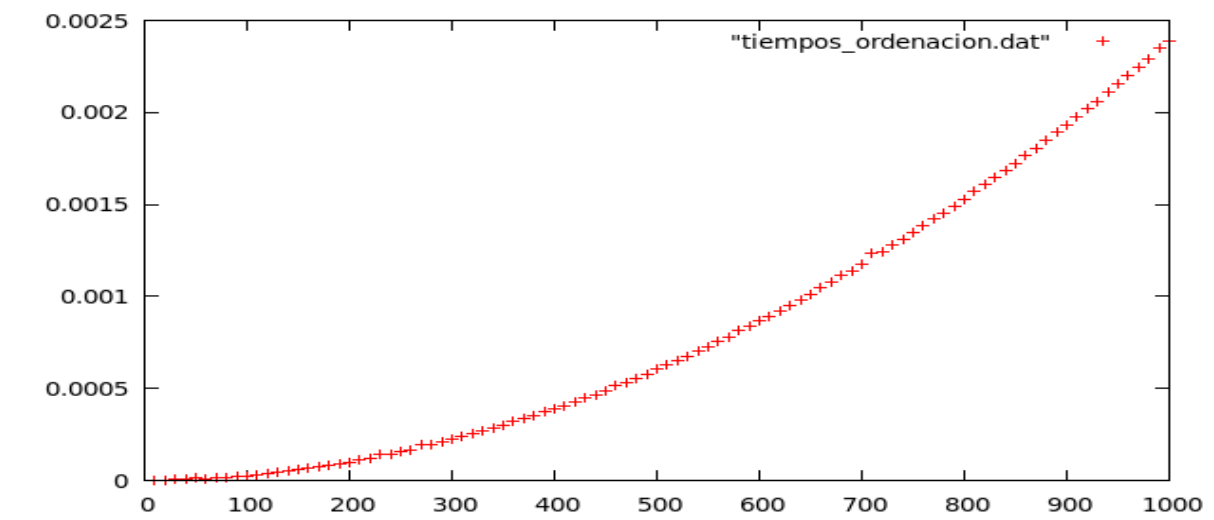
`g++ -o ordenacionPeor ordenacion_peorcaso.cpp`

### 6.- Parámetros usados para el cálculo de la eficiencia empírica y gráfica

#### PEOR CASO

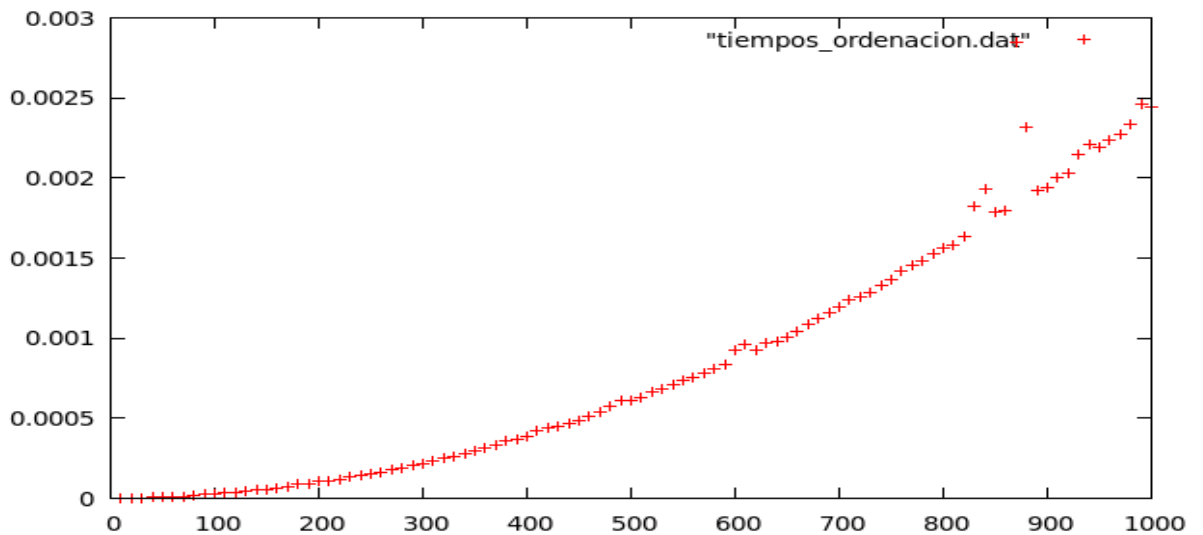
Para ésta parte he ejecutado el programa con los siguientes valores para tamaño máximo del vector : 600 y 6000.

#### 6.1- Gráfica de tiempos para la ejecución del programa con tamaño de vector = 600



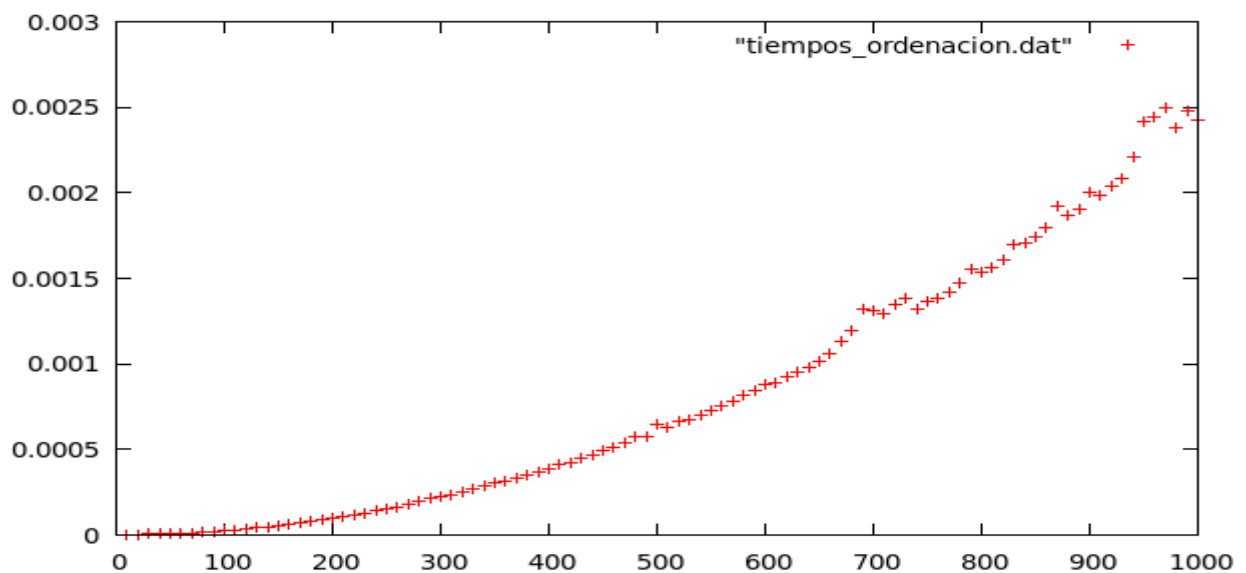
## EJERCICIO 4 : Mejor y peor caso

### 6.2- Gráfica de tiempos para la ejecución del programa con **tamaño de vector = 60000**



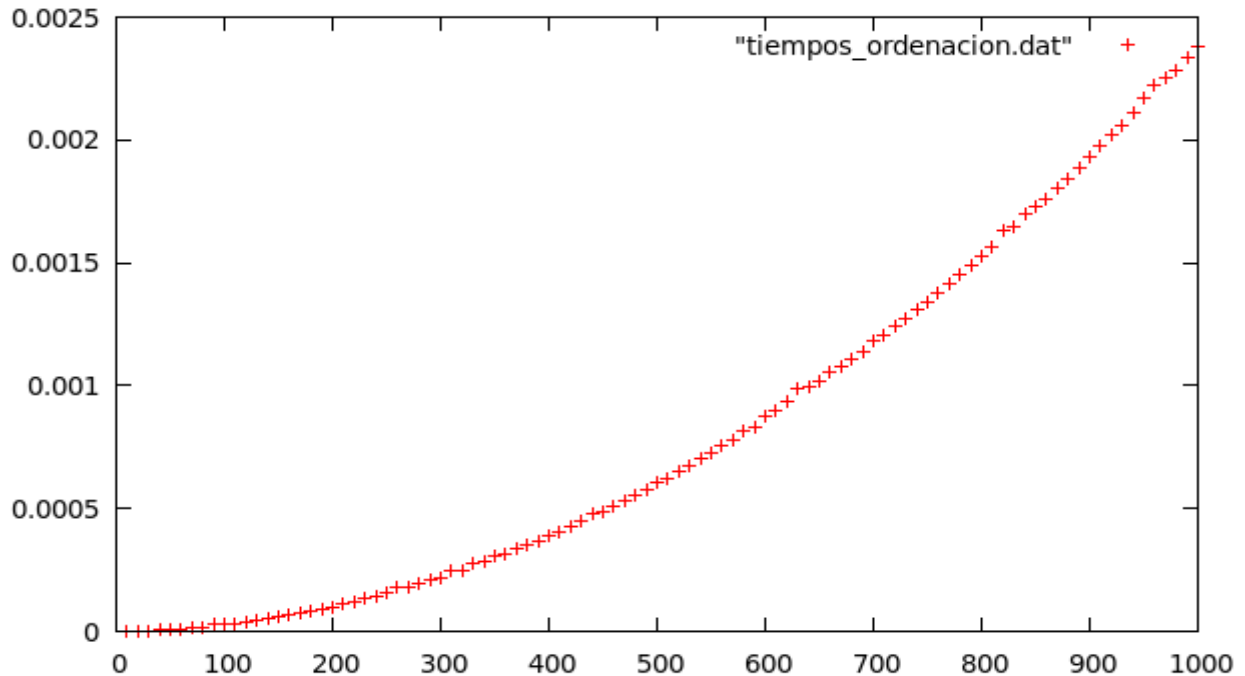
### MEJOR CASO

### 6.3- Gráfica de tiempos para la ejecución del programa con **tamaño de vector = 600**



## EJERCICIO 4 : Mejor y peor caso

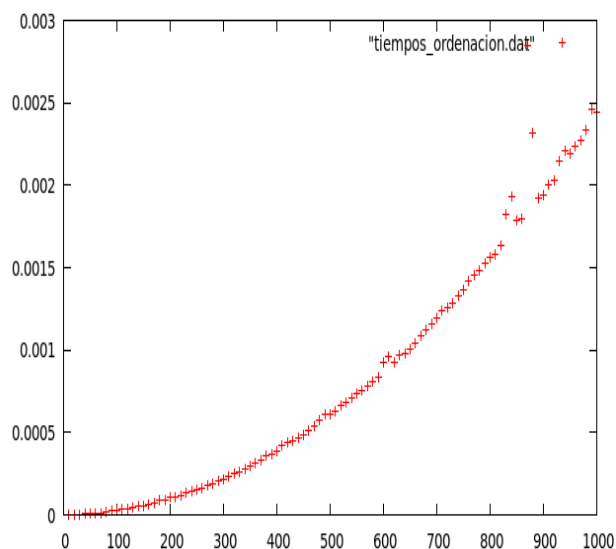
### 6.4- Gráfica de tiempos para la ejecución del programa con **tamaño de vector = 6000**



**\*Con tamaño del vector = 10000 en adelante mi ordenador tarda demasiado en ejecutarlo. Por eso he probado hasta 6000**

**\*Como se puede apreciar, las gráficas tienen la misma eficiencia en ambos casos, pero vemos cómo la curva en el mejor caso es más pronunciada. Ascende más rápido.**

#### PEOR CASO



#### MEJOR CASO

