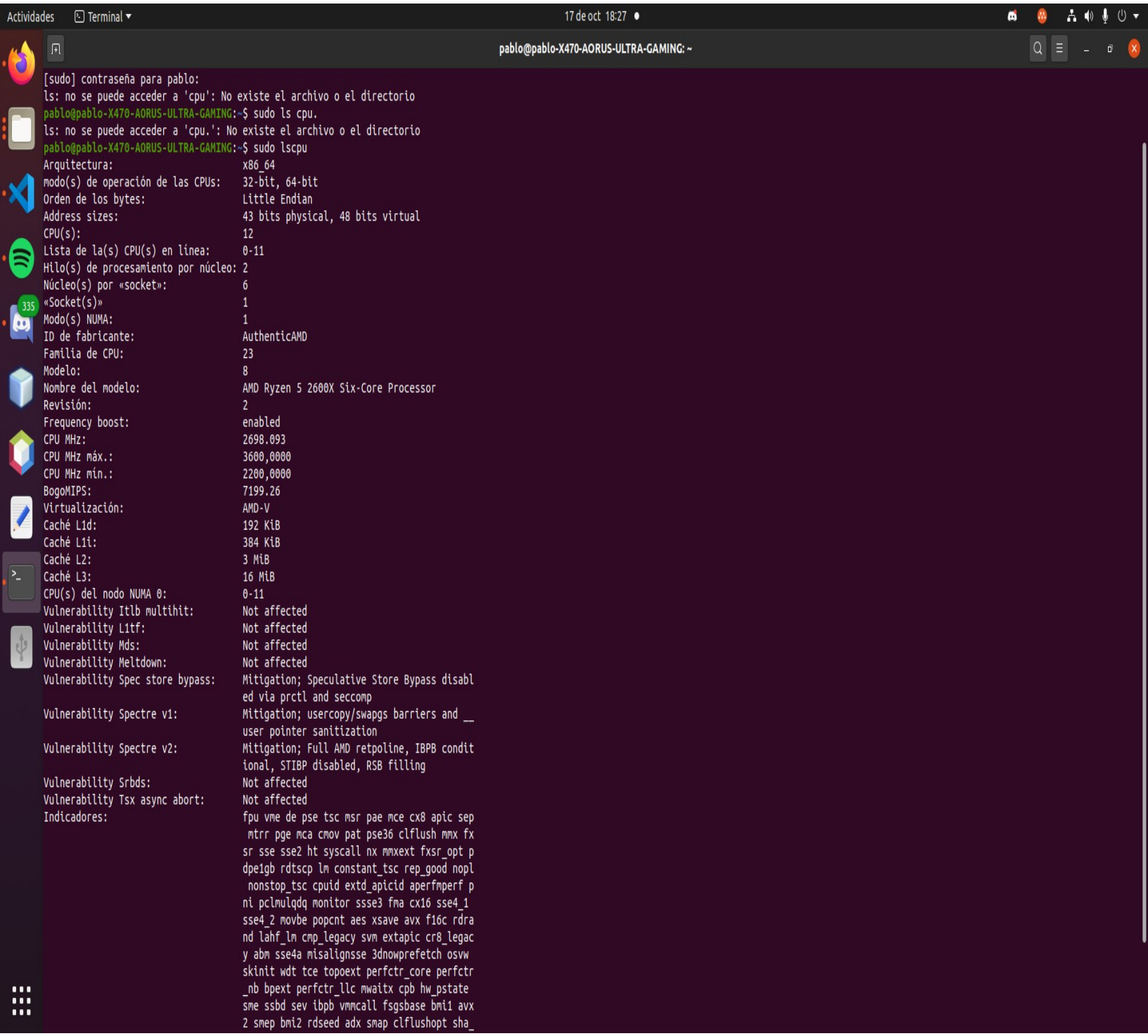
E.D Practicas Eficiencia

Pablo Borrego Megías A3

Hardware usado:

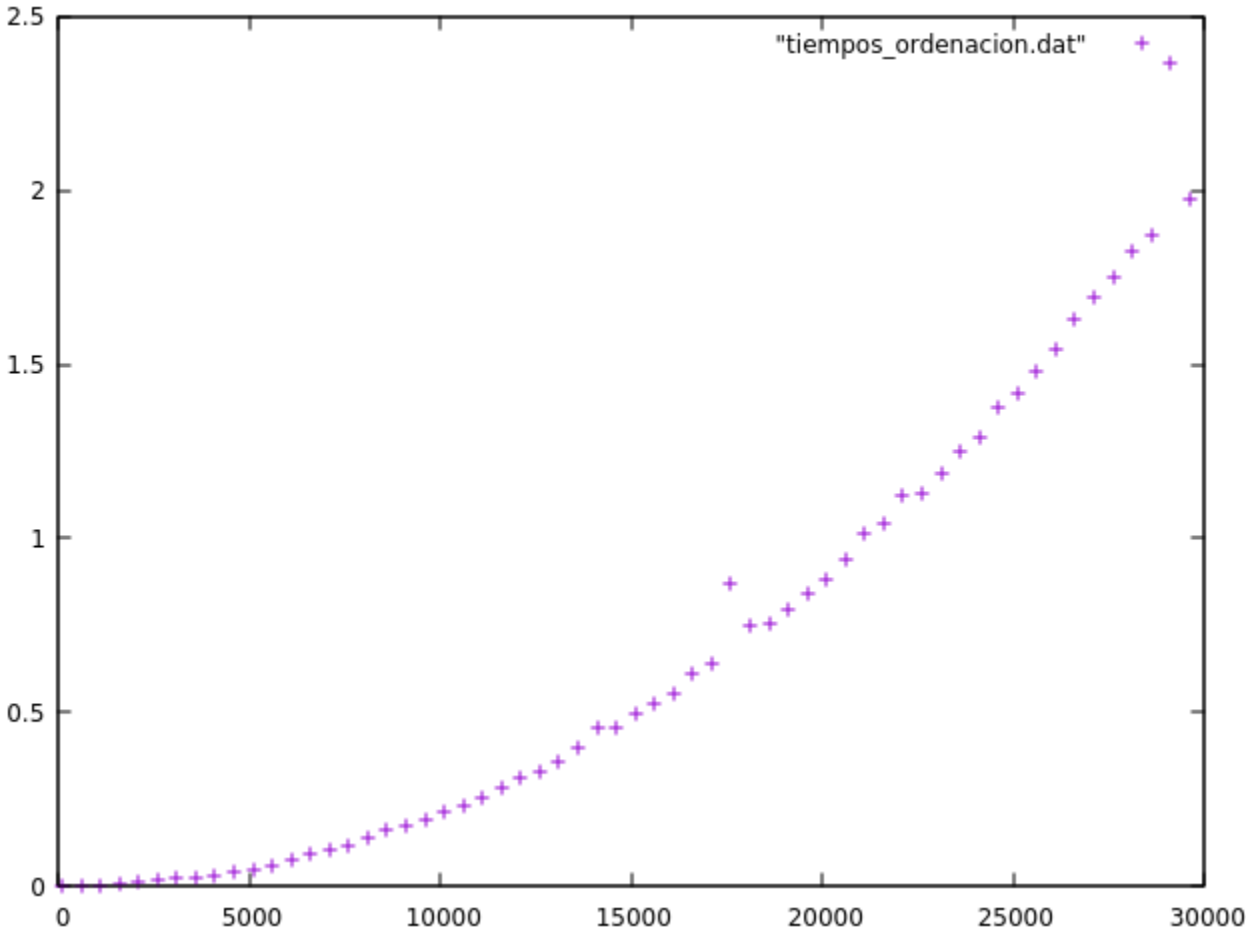


Ejercicio 1.

Código fuente:

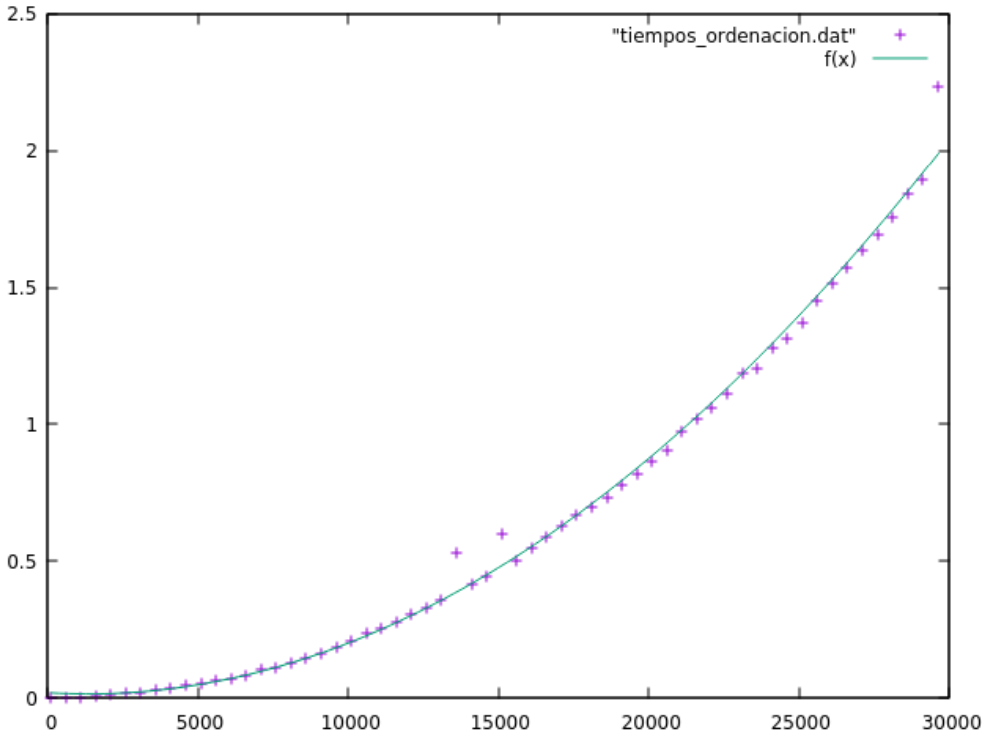
```
#include <iostream>
#include <ctime>    // Recursos para medir tiempos
#include <cstdlib> // Para generación de números pseudoaleatorios

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;
                                        }
                                }

void sintaxis()
{
  cerr << "Sintaxis:" << endl;
  cerr << "  TAM: Tamaño del vector (>0)" << endl;
  cerr << "  VMAX: Valor máximo (>0)" << endl;
  cerr << "Se genera un vector de tamaño TAM con elementos aleatorios en [0,VMAX[" << endl;
  exit(EXIT_FAILURE);
}
int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=3)
    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 del generador de 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); // de esta forma forzamos el peor caso
```

```
  clock_t tfin;    // Anotamos el tiempo de finalización
  tfin=clock();

  // Mostramos resultados
  cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;

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



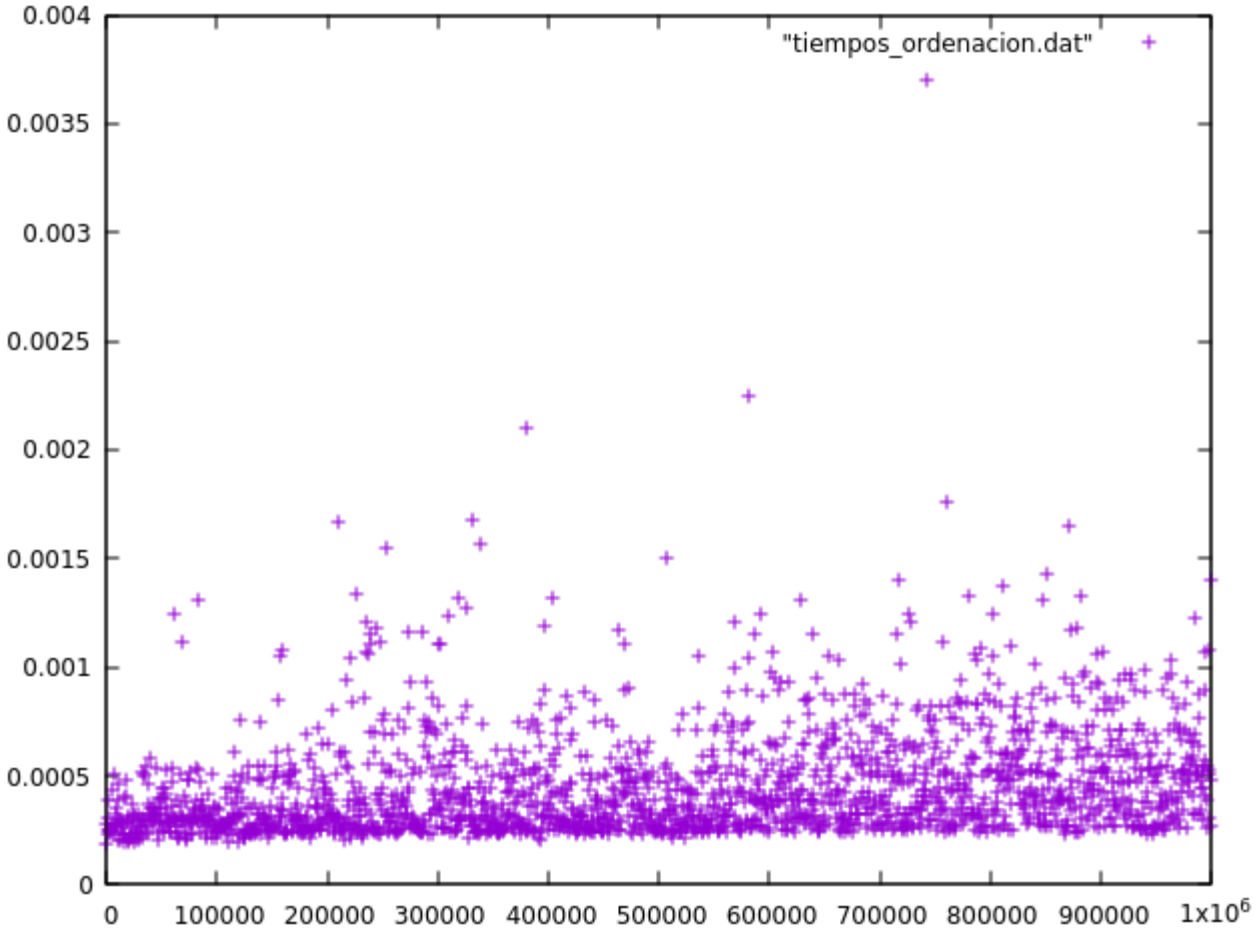
Ejercicio 2.

```
pablo@pablo-X470-AORUS-ULTRA-GAMING: ~/Escritorio/ED/Pra...
iter      chisq       delta/lim  lambda   a             b             c


After 12 iterations the fit converged.
final sum of squares of residuals : 0.10505
rel. change during last iteration : -3.76173e-12

degrees of freedom    (FIT_NDF)                        : 57
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.04293
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00184298

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 2.4654e-09       +/- 8.268e-11    (3.353%)
b               = -6.53486e-06     +/- 2.538e-06    (38.83%)
c               = 0.0204952        +/- 0.0163       (79.55%)

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> plot "tiempos_ordenacion.dat", f(x)
gnuplot>
```



Como podemos observar hay un buen ajuste de la recta ya que coincide con casi todos los puntos.

Ejercicio 3.



Ejercicio 4. Mejor y peor caso

Código Fuente del mejor caso:

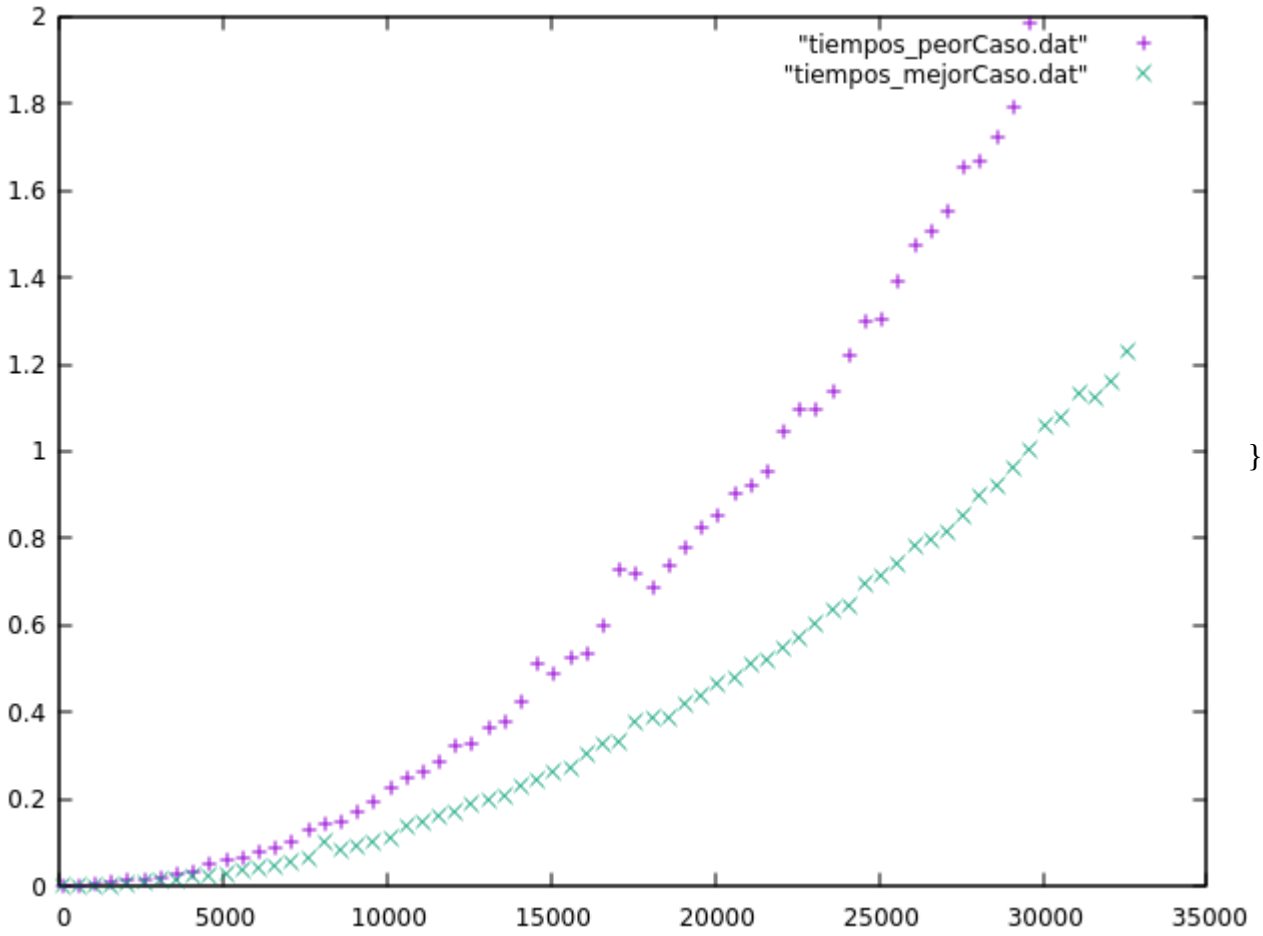
```
#include <iostream>
#include <ctime>    // Recursos para medir tiempos
#include <cstdlib> // Para generación de números pseudoaleatorios

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;
                                        }
                        }
```

```
void sintaxis()
{
  cerr << "Sintaxis:" << endl;
  cerr << "  TAM: Tamaño del vector (>0)" << endl;
  cerr << "  VMAX: Valor máximo (>0)" << endl;
  cerr << "Se genera un vector de tamaño TAM con elementos aleatorios en [0,VMAX[" << endl;
  exit(EXIT_FAILURE);
}
int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=3)
    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 del generador de números pseudoaleatorios
  for (int i=0; i<tam; i++)  // Recorrer vector
    v[i] = i;              //Vector ordenado

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


  ordenar(v,tam); // de esta forma forzamos el peor caso

  clock_t tfin;    // Anotamos el tiempo de finalización
  tfin=clock();

  // Mostramos resultados
  cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;

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

Código fuente del peor caso:

```
#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;
                                                  }
                                   }

void sintaxis()
{
  cerr << "Sintaxis:" << endl;
  cerr << "  TAM: Tamaño del vector (>0)" << endl;
  cerr << "  VMAX: Valor máximo (>0)" << endl;
  cerr << "Se genera un vector de tamaño TAM con elementos aleatorios en [0,VMAX[" << endl;
  exit(EXIT_FAILURE);
}
int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=3)
    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 del generador de números pseudoaleatorios
  for (int i=0; i<tam; i++)  // Recorrer vector
    v[i] = tam-i;              //Vector ordenado

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


  ordenar(v,tam); // de esta forma forzamos el peor caso
```

```
clock_t tfin;    // Anotamos el tiempo de finalización
tfin=clock();

// Mostramos resultados
cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;

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




```
}
```

Como podemos comprobar, el mejor caso es más eficiente que el peor caso, tal y como era de esperar