

E.D Practicas

Eficiencia

Pablo Borrego Megías A3

Hardware usado:

```
Actividades Terminal 17 de oct 18:27 pablo@pablo-X470-AORUS-ULTRA-GAMING: ~

[sudo] contraseña para pablo:
ls: no se puede acceder a 'cpu': No existe el archivo o el directorio
pablo@pablo-X470-AORUS-ULTRA-GAMING:~$ sudo ls cpu.
ls: no se puede acceder a 'cpu.': No existe el archivo o el directorio
pablo@pablo-X470-AORUS-ULTRA-GAMING:~$ sudo lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPUs: 32-bit, 64-bit
Orden de los bytes:    Little Endian
Address sizes:         43 bits physical, 48 bits virtual
CPU(s):                12
Lista de la(s) CPU(s) en línea: 0-11
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»: 6
«Socket(s)»:          1
Modo(s) NUMA:          1
ID de fabricante:      AuthenticAMD
Familia de CPU:         23
Modelo:                8
Nombre del modelo:     AMD Ryzen 5 2600X Six-Core Processor
Revisión:              2
Frequency boost:       enabled
CPU MHz:               2698.093
CPU MHz máx.:          3600.0000
CPU MHz mín.:          2200.0000
BogoMIPS:              7199.26
Virtualización:        AMD-V
Caché L1d:             192 KiB
Caché L1i:             384 KiB
Caché L2:              3 MiB
Caché L3:              16 MiB
CPU(s) del nodo NUMA 0: 0-11
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf:    Not affected
Vulnerability Mds:     Not affected
Vulnerability Meltdown: Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Full AMD retpoline, IBPB conditional, STIBP disabled, RSB filling
Vulnerability Srbds:   Not affected
Vulnerability Tsx async abort: Not affected
Indicadores:           fpu vme de pse tsc msr pae mce cx8 apic sep
                        mtrr pge mca cmov pat pse36 clflush mmx fx
                        sr sse sse2 ht syscall nx mmxext fxsr_opt p
                        dpe1gb rdtscp lm constant_tsc rep_good nopl
                        nonstop_tsc cpuid extd_apicid aperfmperf p
                        ni pclmulqdq monitor ssse3 fma cx16 sse4_1
                        sse4_2 movbe popcnt aes xsave avx f16c rdra
                        nd lah_f_lm cmp_legacy svm extapic cr8_legac
                        y abm sse4a misalignsse 3dnowprefetch osvw
                        skinit wdt tce topoext perfctr_core perfctr
                        _nb bpext perfctr_llc mwaitx cpb hw_pstate
                        sme ssbd sev ibpb vmmcall fsgsbase bmi1 avx
                        2 smep bmi2 rdseed adx smap clflushopt sha_
```

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
```

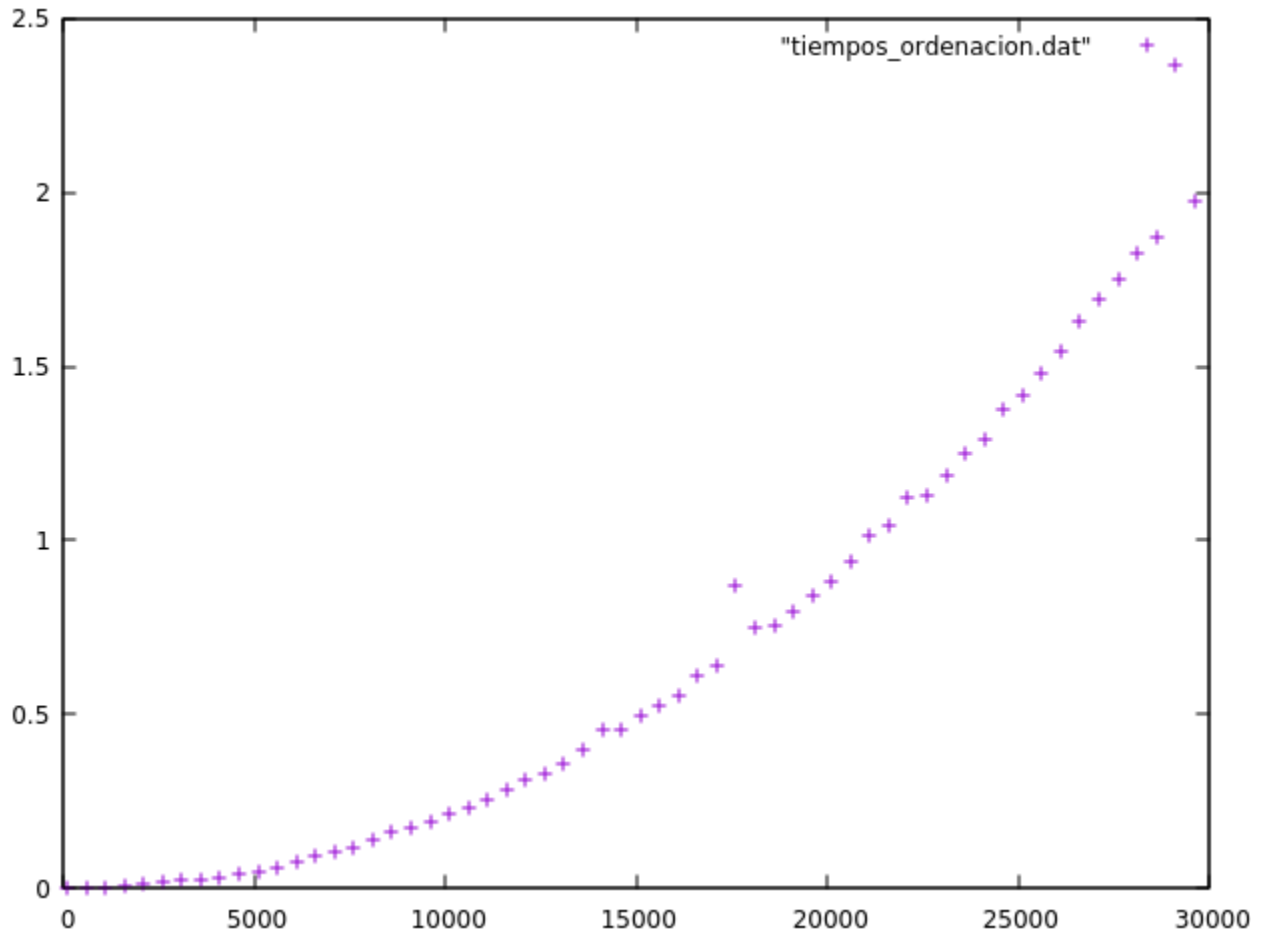
```

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.

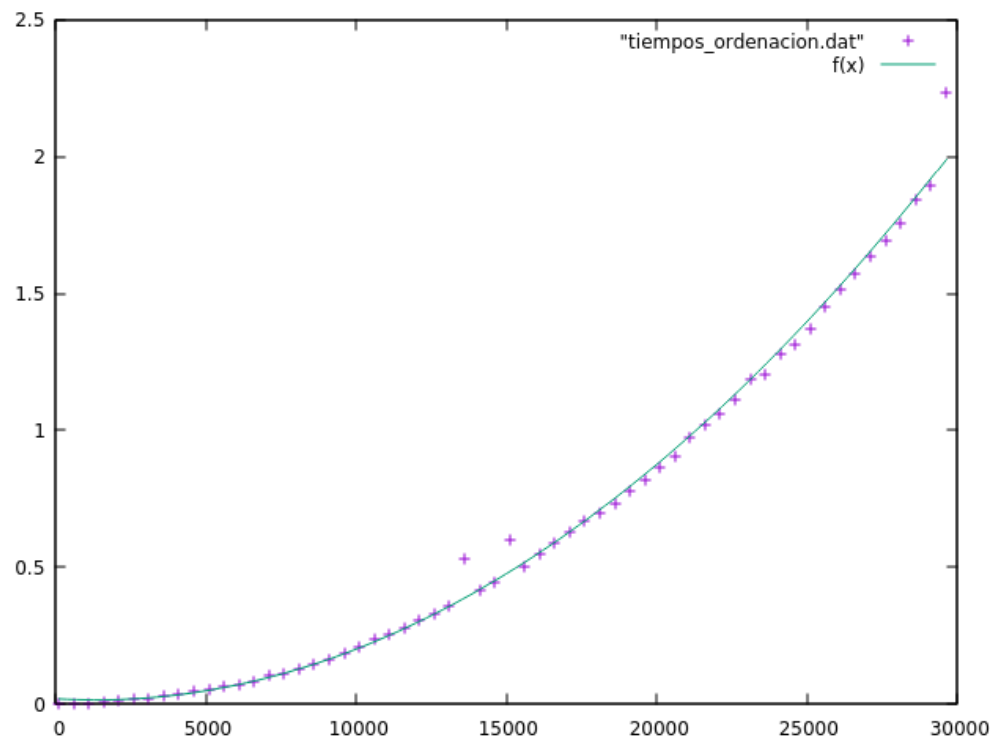
```
pablo@pablo-X470-AORUS-ULTRA-GAMING: ~/Escritorio/ED/Pra...
iter      chisq      delta/lim  lambda    a          b          c

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

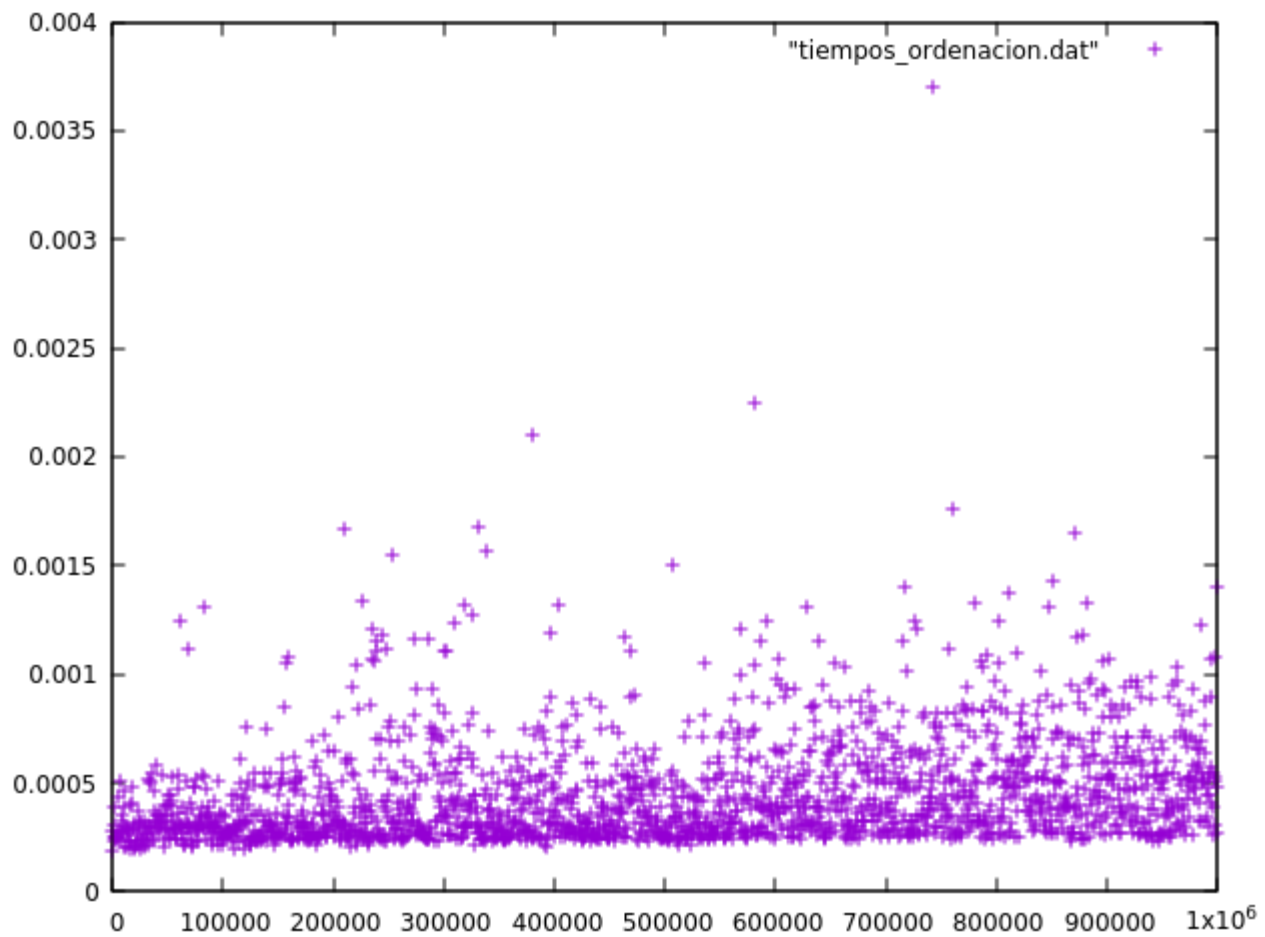
Final set of parameters      Asymptotic Standard Error
=====
a          = 2.4654e-09      +/- 8.268e-11    (3.353%)
b          = -6.53486e-06    +/- 2.538e-06   (38.83%)
c          = 0.0204952      +/- 0.0163      (79.55%)

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
```

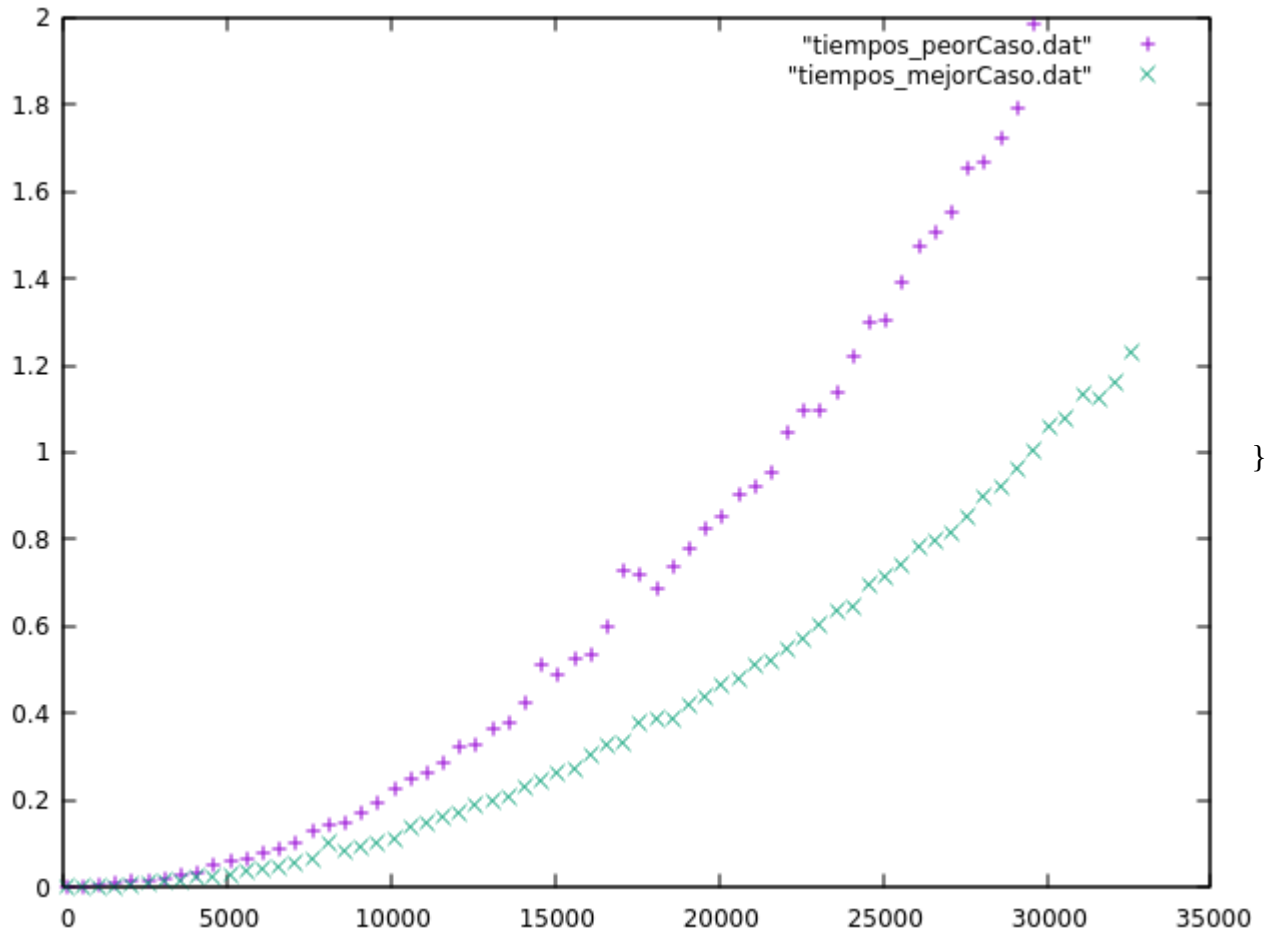
```

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