Práctica 1 ED, ejercicios 1, 2, 4, 5 y 6

Javier Gálvez Obispo

Para la realización de la práctica se ha utilizado una máquina virtual con procesador i5-6500, con una frecuencia de reloj de 3.2GHz, limitado a una cpu, 3GB de RAM y ubuntu 16.04 LTS de 32bits como sistema operativo.

Código fuente base:

```
#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 && cambio; i++){
                for(int j = 0; j < n-i-1; j++)
                        if (v[j] > v[j+1]){
                                cambio = true;
                                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" << endl;
  exit(EXIT_FAILURE);
}

int main(int argc, char * argv[])
{
  if (argc!=2)
    sintaxis();
  int tam=atoi(argv[1]);
  if (tam<=0)
    sintaxis();

  int *v=new int[tam];
  srand(time(0));
  for (int i=0; i<tam; i++)
    v[i] = rand() % tam;

  clock_t tini;
  tini=clock();
```

```
  ordenar(v,tam);
  clock_t tfin;
  tfin=clock();

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

  delete [] v;
}
```

Ejercicio 1.

Eficiencia teórica:

```
void ordenar (int *v, int n){
        for(int i = 0; i < n-1 && cambio; i++){                 |
                for(int j = 0; j < n-i-1; j++)          |       |
                        if (v[j] > v[j+1]){     |       |       |
                                cambio = true;|       |O(n)   |O(n)
                                int aux = v[j]; |O(1)   |       |
                                v[j] = v[j+1];  |       |       |
                                v[j+1] = aux;   |       |       |
                        }
        }
}
```

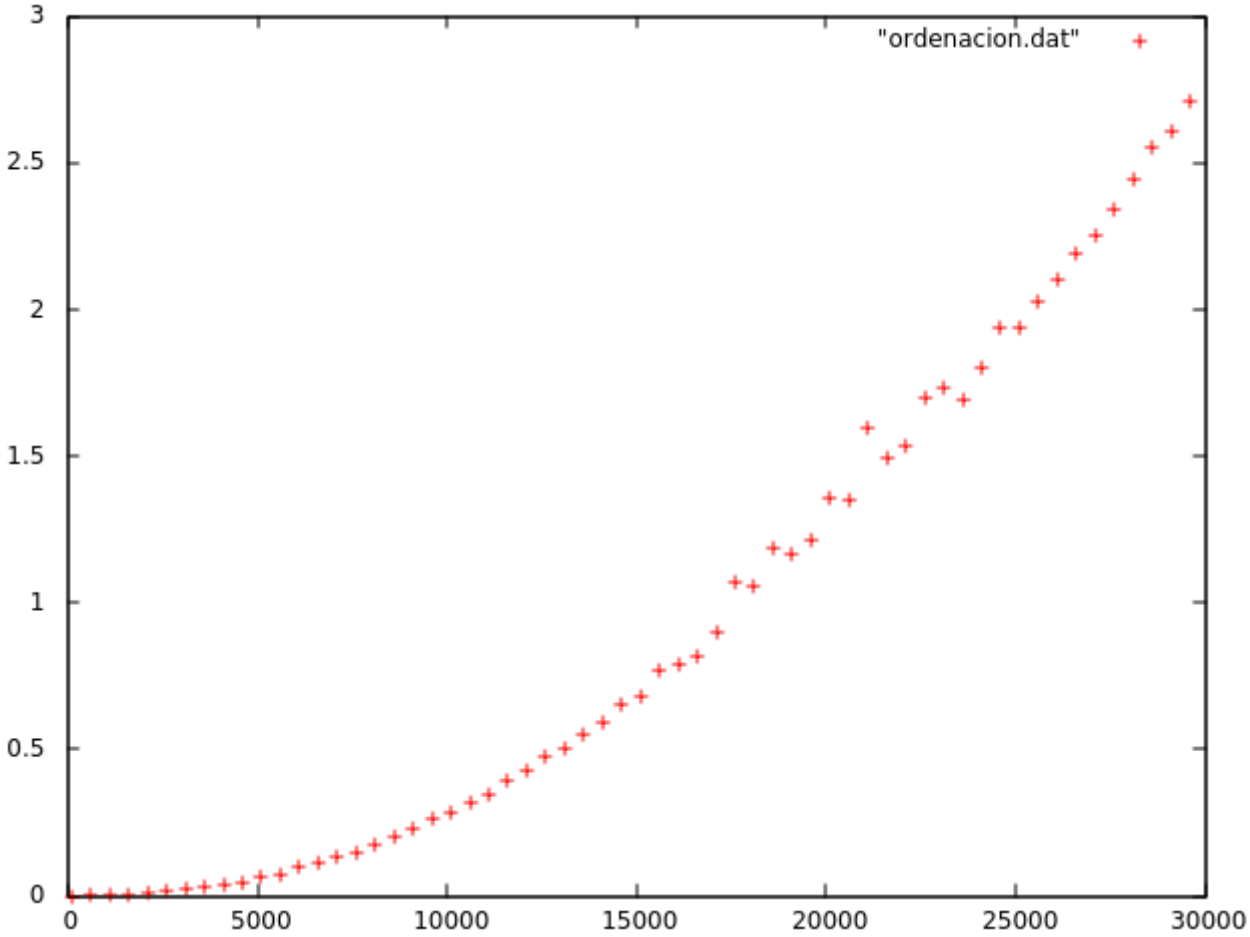
Entonces la eficiencia teórica es:
$O(1) * O(n) * O(n) = O(n^2)$

Para la compilación del programa se ha utilizado la siguiente orden:
g++ ordenacion.cpp -o ordenacion

El script utilizado para las ejecuciones es el siguiente:

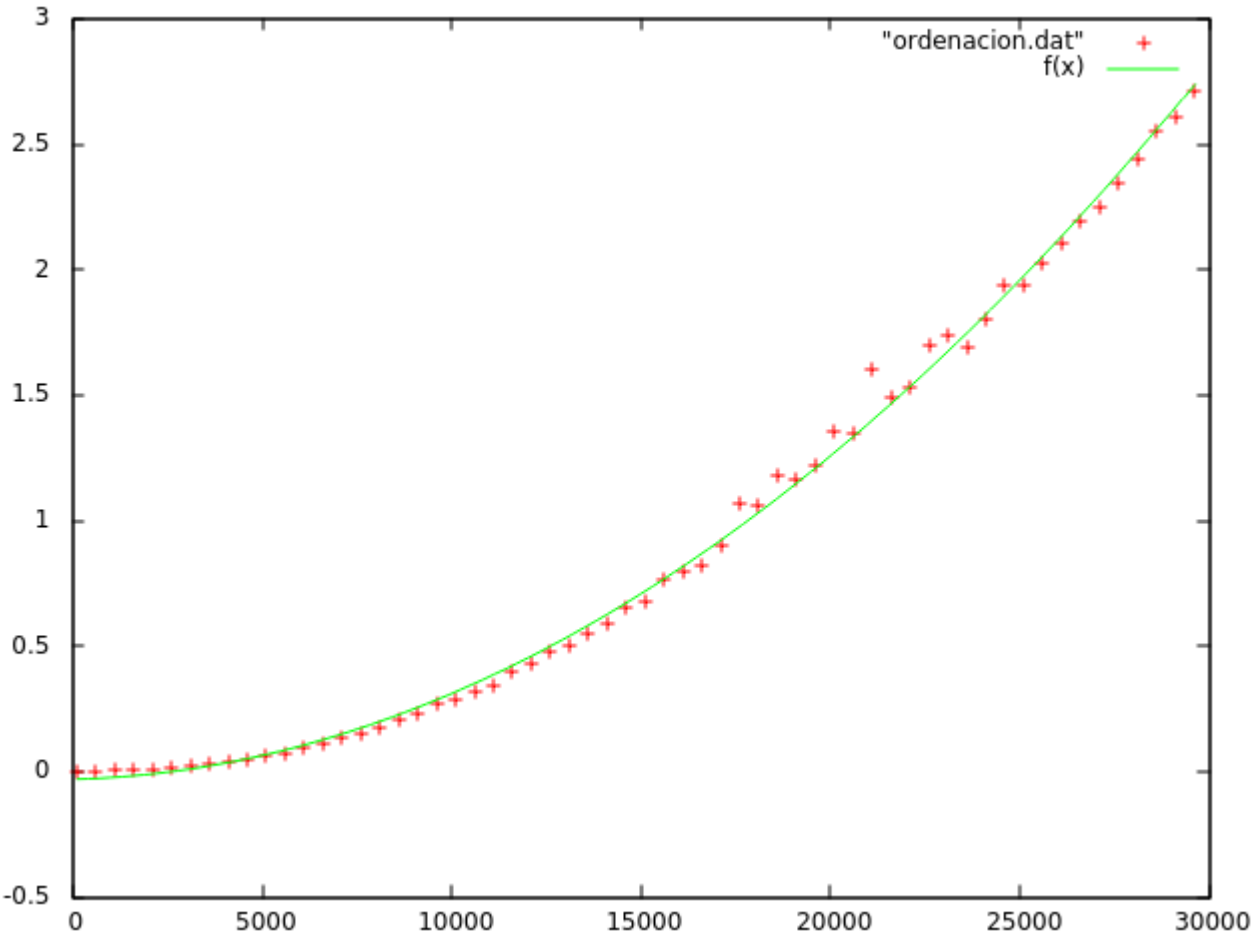
```
#!/bin/bash
inicio=100
fin=30000
incremento=500
ejecutable="ordenacion"
salida="ordenacion.dat"

i=$inicio
echo > $salida
while [ $i -lt $fin ]
do
  echo "Ejecución tam = " $i
  echo `./$ejecutable $i` >> $salida
  i=$[$i+$incremento]
done
```

El resultado al dibujar los datos con gnuplot es el siguiente:



Ejercicio 2

Ax^2+Bx+C
a = 3.02453e-09
b = 3.96199e-06
c = -0.0316537



Ejercicio 4.

Para la realización de este ejercicio tenemos que modificar el código fuente presentado anteriormente y cambiar la siguiente línea:

```
v[i] = rand() % tam;
```

por

```
v[i] = i;
```

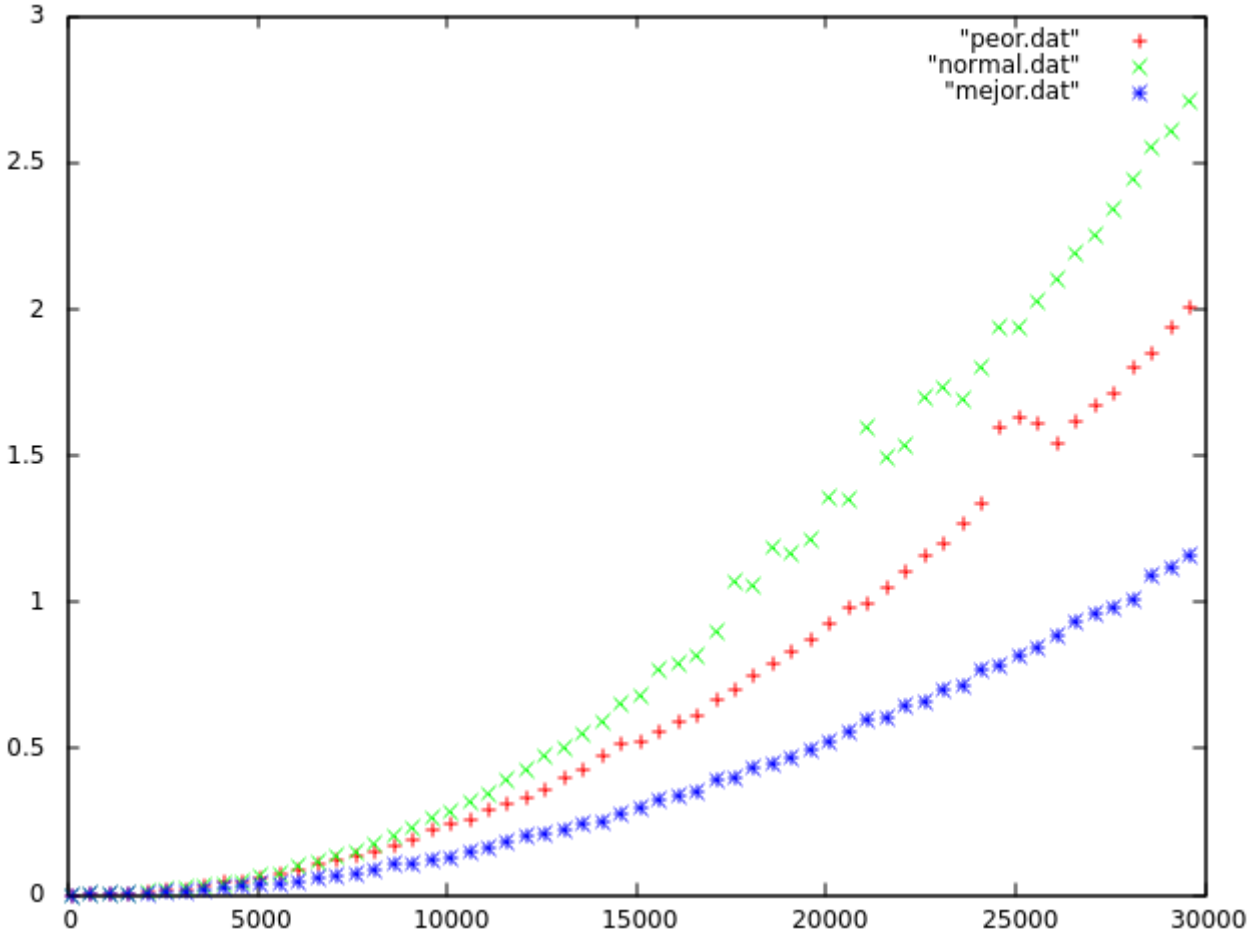
y

```
v[i] = -i;
```

para tener el mejor caso y el peor caso respectivamente.

Utilizamos la misma orden para compilar ahora y en el script modificamos los nombre de salida y del ejecutable.

Al dibujar los resultados del caso normal, el peor y el mejor en gnuplot obtenemos el siguiente gráfico:



Ejercicio 5.

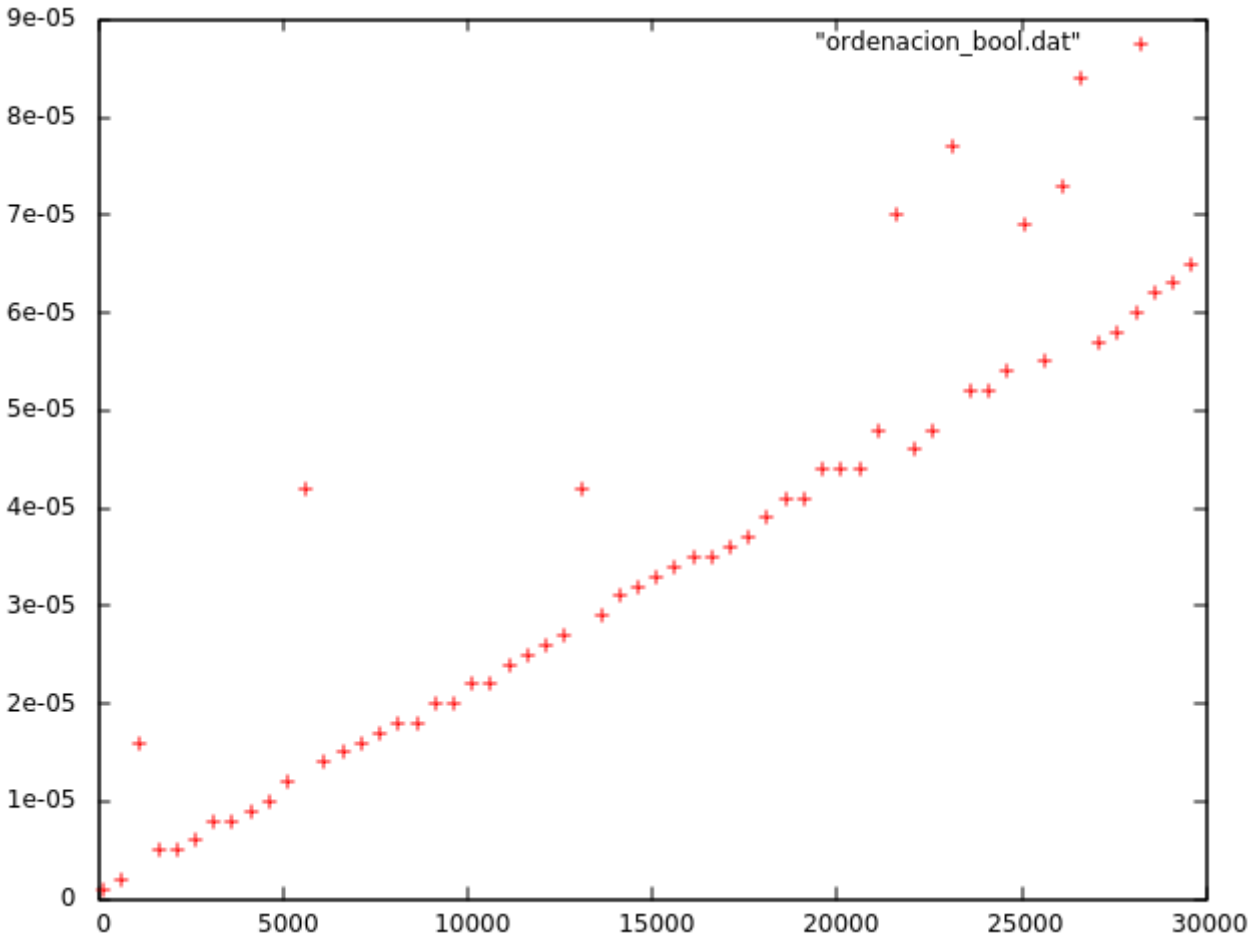
Modificamos el algoritmo de ordenación y suponemos que estamos en el mejor caso (v[i] = i;) entonces, la eficiencia teórica es:

```
void ordenar (int *v, int n){
        bool cambio = true;    |O(1)                                    |
        for(int i = 0; i < n-1 && cambio; i++){                |       |
                cambio = false;          |O(1)                 |       |
                for(int j = 0; j < n-i-1; j++)          |      |       |
                        if (v[j] > v[j+1]){       |      |      |       |O(n)
                                cambio = true;|         |O(n)  |O(1)   |
                                int aux = v[j]; |O(1)   |      |       |
                                v[j] = v[j+1];  |       |      |       |
                                v[j+1] = aux;   |       |      |       |
                        }
        }
}
```

O(1) * O(n) * O(1) = O(n)

La eficiencia del algoritmo de esta forma es O(n). Al estar en el mejor caso solo se va a realizar una iteración del primer bucle for.

Y como podemos comprobar con la gráfica, la eficiencia empírica se aproxima a una recta.



Ejercicio 6.

Utilizamos ahora la siguiente orden para compilar:

g++ -O3 ordenacion.cpp -o ordenacion_optimizado

y comprobamos que efectivamente mejora la eficiencia del programa comparando las gráficas: