ESTRUCTURA DE DATOS

Práctica 1. Eficiencia de algoritmos

Doble Grado de Informática y Matemáticas

Víctor Castro Serrano
Maximino Suárez van Gelderen

28 de septiembre de 2017

Condiciones de ejecución.

Dado que en los siguientes ejercicios hablaremos de la eficiencia de distintos programas, conviene detallar las condiciones en las que se han llevado a cabo las pruebas.

Hardware: Asus GL552VW, Intel Core i5-6300HQ CPU @ 2.30GHz 4 cores, Intel HD Graphics 530 (Skylake GT2), 12GB RAM.
Sistema Operativo: Ubuntu 16.04.3 LTS 64-bit.
Compilador: g++
Opciones de compilación: -g -o

Ejercicio 1.

En este ejercicio, comprobaremos tanto la eficiencia teórica como la eficiencia empírica del algoritmo de ordenación *burbuja*:

```
1  void ordenar(int *v, int n)
2  {
3    for (int i=0; i<n-1; i++)
4      for (int j=0; j<n-i-1; j++)
5        if (v[j]>v[j+1]) {
6          int aux = v[j];
7          v[j] = v[j+1];
8          v[j+1] = aux;
9        }
10 }
```

Comencemos analizando la eficiencia teórica del algoritmo, en el caso peor. Veamos primero el coste en operaciones elementales (OE) de cada línea:

Línea 3. Hay 4 OE: una asignación, una resta, una comparación y un incremento.
Línea 4. Hay 5 OE: igual que la línea anterior, pero se realizan dos restas.
Línea 5. Hay 4 OE: dos accesos a un vector, una suma y una comparación.
Línea 6. Hay 2 OE: asignación y acceso al vector.
Línea 7. Hay 4 OE: dos accesos, suma y asignación.
Línea 8. Hay 3 OE: acceso, suma y asignación.

Entonces, la eficiencia del algoritmo es:

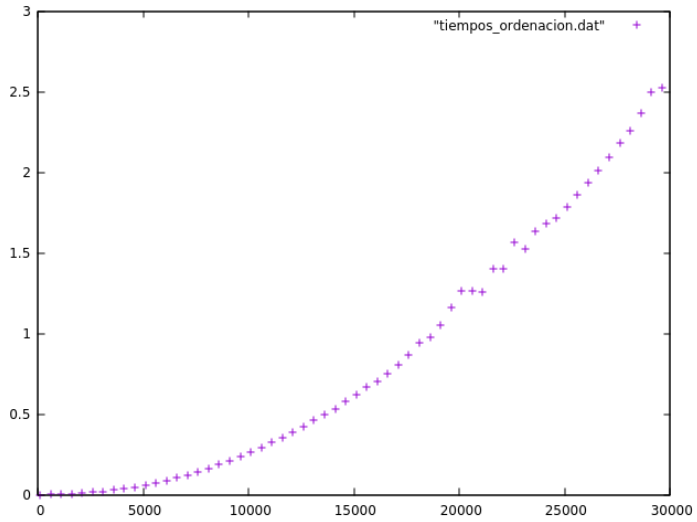
$$T(n) = \sum_{i=0}^{n-2} \left(4 + \sum_{j=0}^{n-i-2} 4 + 9 \right) = \sum_{i=0}^{n-2} (4 + 13(n-i-1)) = \frac{13n^2 - 5n - 8}{2}.$$

Por tanto, afirmamos que $T(n) \in O(n^2)$, y el algoritmo de ordenación burbuja es de orden de eficiencia $O(n^2)$.

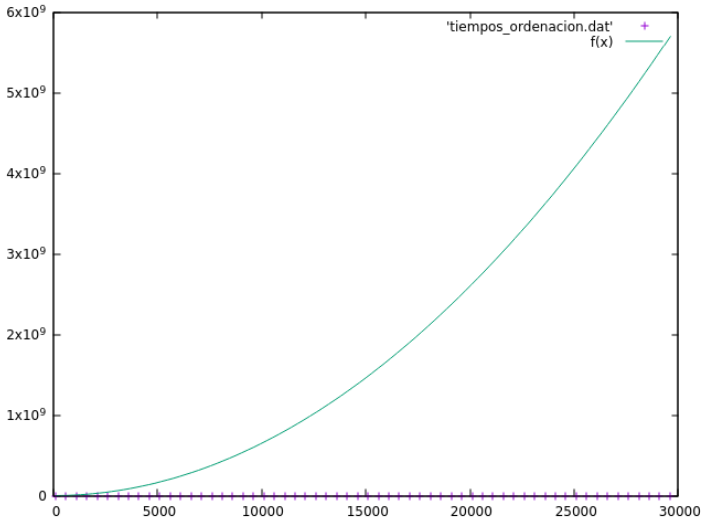
Ahora creamos un programa de prueba para analizar la eficiencia empírica, haciendo uso de la biblioteca *ctime* del lenguaje C++:

```
#include <iostream>
#include <ctime>
#include <cstdlib>
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 << "Genera un vector de TAM números aleatorios en [0,VMAX[" << endl;
exit(EXIT_FAILURE);
}
int main(int argc, char * argv[]) {
if (argc!=2)
// Lectura de parámetros
sintaxis();
int tam=atoi(argv[1]);
// Generación del vector aleatorio
int *v=new int[tam];
// Reserva de memoria
srand(time(0));
// Inicialización generador números pseudoaleatorios
for (int i=0; i<tam; i++) // Recorrer vector
v[i] = rand();
// Generar aleatorio [0,vmax[
clock_t tini;
tini=clock();
// Anotamos el tiempo de inicio

ordenar(v,tam); // de esta forma forzamos el peor caso
clock_t tfin;
tfin=clock();
// Anotamos el tiempo de finalización
// Mostramos resultados (Tamaño del vector y tiempo de ejecución en seg.)
cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;
delete [] v;
}
```

Al analizar la eficiencia empírica del algoritmo, obtenemos la siguiente gráfica:



Si representamos superpuestas la función de la eficiencia teórica y la empírica, obtenemos lo siguiente:



Observamos que, aunque la curva de la eficiencia teórica tiene la misma forma que la nube de puntos obtenidas experimentalmente, ambas curvas no coinciden al representarlas superpuestas. Esto se debe a que las constantes no coinciden, pues el costo de una operación elemental, e incluso el tiempo total de ejecución del algoritmo, dependen de las condiciones particulares de la máquina en la que se ejecuta.