Práctica 1: Eficiencia

Antonio Manuel Fernández Cantos

12 de octubre de 2015

Índice

1. Introducción **1**

2. Componentes utilizados **1**

3. Ejercicio 5: Dependencia de la implementación **2**
3.1. Introducción 2
3.2. Código 3
3.3. Eficiencia teórica 3
3.4. Comparativa eficiencia teórica y empírica 3

1. Introducción

La práctica consiste en calcular la eficiencia **teórica y empírica** de un código en c++ y **realizar un ajuste de la curva de eficiencia teórica a la empírica.** Se utilizará la biblioteca **ctime** para poder obtener los resultados empíricos. Dentro de la biblioteca ctime tenemos la función **clock()** que devuelve el número de ticks que han transcurrido desde un momento determinado, es esta función la que usaremos para medir la diferencia de tiempo entre el inicio del algoritmo y su finalización.

2. Componentes utilizados

En el cálculo empírico, el algoritmo tardará más o menos en función de:

- **Hardware usado**:
  - CPU
  - RAM
  - HDD
- **Sistema Operativo**

- **Compilador (y sus opciones de compilación)**
- **Bibliotecas**

Todos estos componentes se tienen en cuenta cuando se obtiene el tiempo que tarda nuestro algoritmo en ejecutar todas las sentencias. Dependiendo de la potencia de nuestro ordenador y de las librerias usadas, el algoritmo tardará más o menos. Para la realización del cálculo empírico de los ejercicios, he usado los siguientes componentes:

- **Hardware usado**:
  - Procesador: 8x Intel(R) Core(TM) i7-3630QM CPU@2.40MHz
  - RAM: 6GB
  - CPU clock: 1200 MHz
  - HDD: 750GB
- **Sistema Operativo**: Ubuntu 14.04.3 LTS
- **Compilador**: GCC sin opciones de compilación
- **Bibliotecas**:
  - iostream (E/S)
  - ctime (Para medir el tiempo de ejecución de un algoritmo)
  - cstdlib (Para generar números pseudoaleatorios)

3. Ejercicio 5: Dependencia de la implementación

3.1. Introducción

Este ejercicio consiste en modificar el código del primer ejercicio, introduciendo una variable booleana para saber si el vector está ordenado. Con esta variable lo que hacemos es ahorrarnos iteraciones si el vector está ordenado.

3.2. Código

```
1 void ordenar(int *v, int n) {
2   bool cambio=true;
3   for (int i=0; i<n-1 && cambio; i++) {
4     cambio=false;
5     for (int j=0; j<n-i-1; j++)
6       if (v[j]>v[j+1]) {
7         cambio=true;
8         int aux = v[j];
9         v[j] = v[j+1];
10        v[j+1] = aux;
11      }
12    }
13  }
14 }
```

3.3. Eficiencia teórica

En este apartado analizaremos la eficiencia teórica del mejor caso posible (vector de entrada ordenado).

Lo que nos interesa analizar es el O grande y no el tiempo de ejecución, así que iremos poniendo los O grandes para un vector ordenado de inicio.

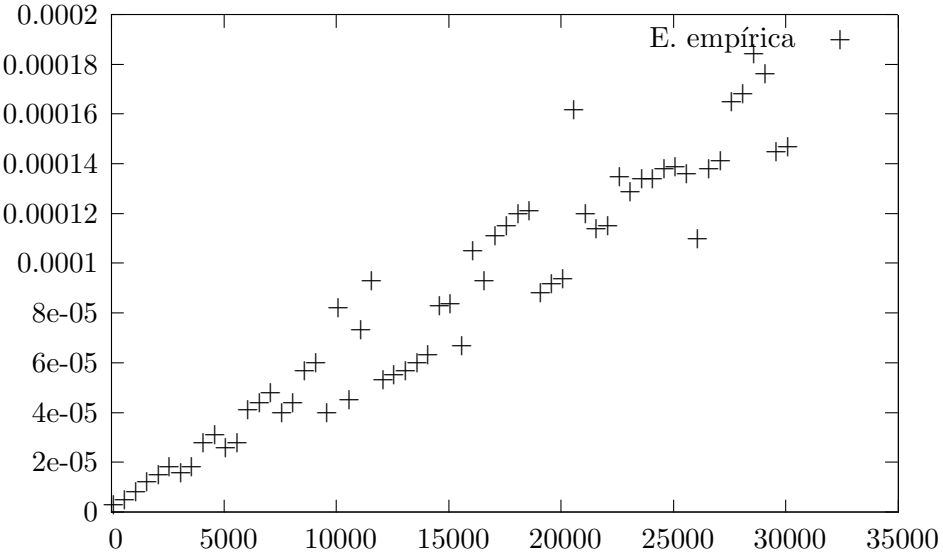
- Linea 2: O(1)
- Linea 3: En este caso tenemos un O(1) ya que solo realizará una iteración al estar el vector ordenado. La variable booleana de la siguiente linea no cambiará. En el caso de estuvieramos en un vector desordenado, el peor de los casos sería O(n).
- Linea 4: O(1)
- Linea 5: Aquí si se recorrerá todo el bucle for. La variable i en el primer caso (y único) caso vale 0. Entonces tenemos que j=0 y j < n-1. Con lo cual tenemos un O(n).

Desde la linea 6 hacia abajo no nos interesa analizarlo ya que solo realiza las operaciones del if pero no entra en ningún momento al no cumplirse la condición.

En definitiva el primer for es O(1) y el for anidado tenemos O(n). Al tener un anidamiento, tenemos que usar la regla del producto y obtenemosque $O(1) * O(n) = O(n)$

3.4. Comparativa eficiencia teórica y empírica

Esta es la gráfica de su eficiencia empírica:



Como podemos observar el algoritmo de ordenación con la variable que hemos introducido para que una vez el vector ordenado salga, es tan rápido que incluso pierde un poco de precisión a la hora de calcular el tiempo ya que trabaja con un tiempo muy pequeño.

Vemos que se corresponder con el crecimiento de la eficiencia teórica, las dos tienen crecimiento lineal, O(n).