Práctica 1: Eficiencia

Antonio Manuel Fernández Cantos

11 de octubre de 2015

Índice

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 1: Ordenacion por burbuja

3.1. Definición

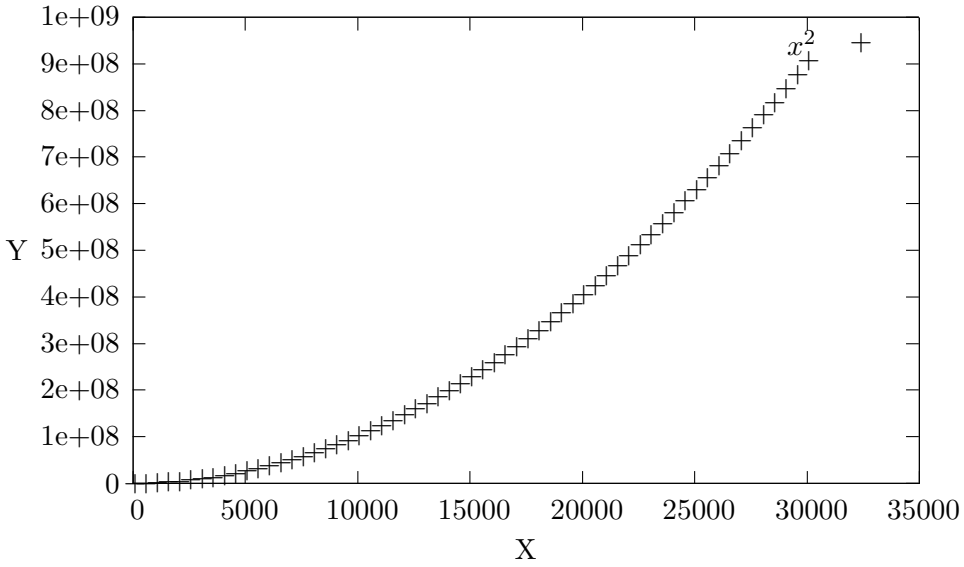
La Ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas".

3.2. Código

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

3.3. Eficiencia teórica

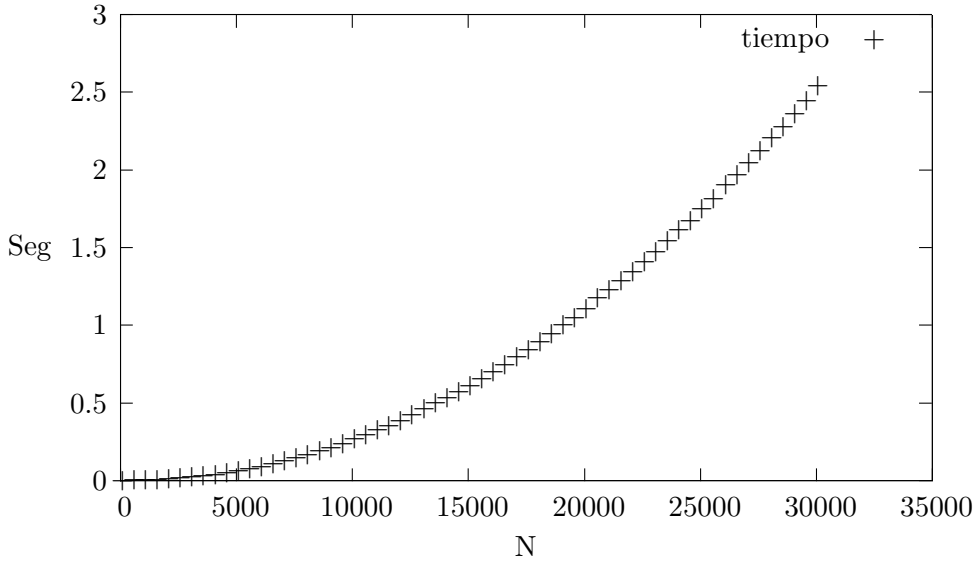
El orden de eficiencia en el peor de los casos (notación O grande) es $O(n^2)$. La siguiente gráfica representa la eficiencia teórica del algoritmo de ordenación por burbuja:



El eje X representa el tamaño del vector a ordenar y el eje Y es el resultado de elevar x al cuadrado.

3.4. Eficiencia empírica

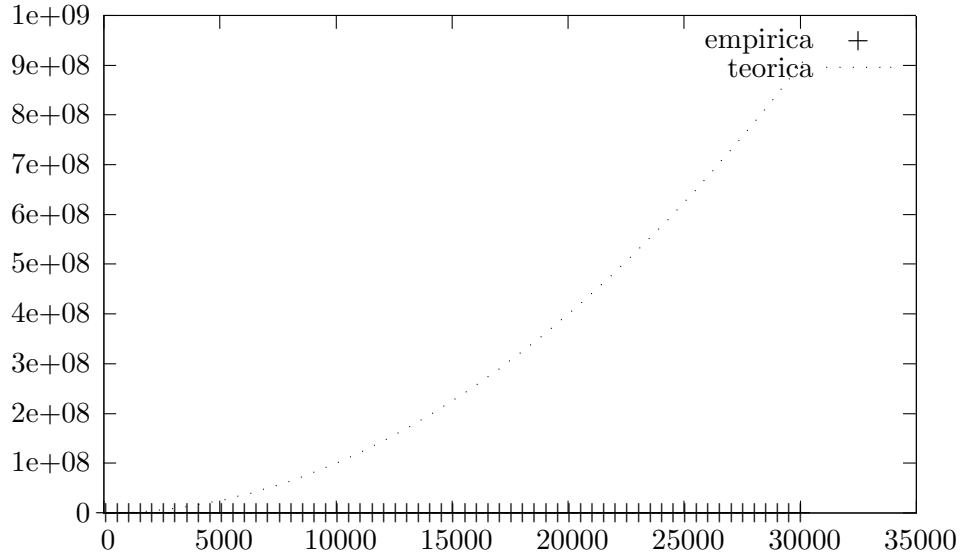
Para el cálculo de la eficiencia empírica se le ha dado a la funcion Ordenacion distintos tamaños del vector. El algoritmo tardaba más tiempo en ordenar el vector conforma más grande era. Se ha comenzado con un tamaño de vector 100 y se ha finalizado con un tamaño 30100. Los resultados vienen representados en la siguiente gráfica:



El eje x representa los distintos tamaños del vector. El eje y representa el tiempo (segundos).

3.5. Comparación teórica y empírica

Cuando medimos la eficiencia empírica de un algoritmo podemos predecir que tipo de curva gráficamente tendremos en la eficiencia empírica. Lo que se traduce en si conforme vamos aumentando los datos de entrada, la respuesta del algoritmo será más rápida o más lenta. Sin embargo no podemos predecir el tiempo en segundos que tardará el algoritmo en responder según los datos introducidos. La siguiente gráfica muestra lo que estamos comentando en estas lineas:



Con esta gráfica podemos concluir que con la eficiencia teórica podemos estimas como de rápido crecerá la gráfica pero no el tiempo exacto que obtendremos al finalizar el algoritmo.