Práctica 1: Eficiencia

Antonio Manuel Fernández Cantos

12 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 4: Mejor y peor caso

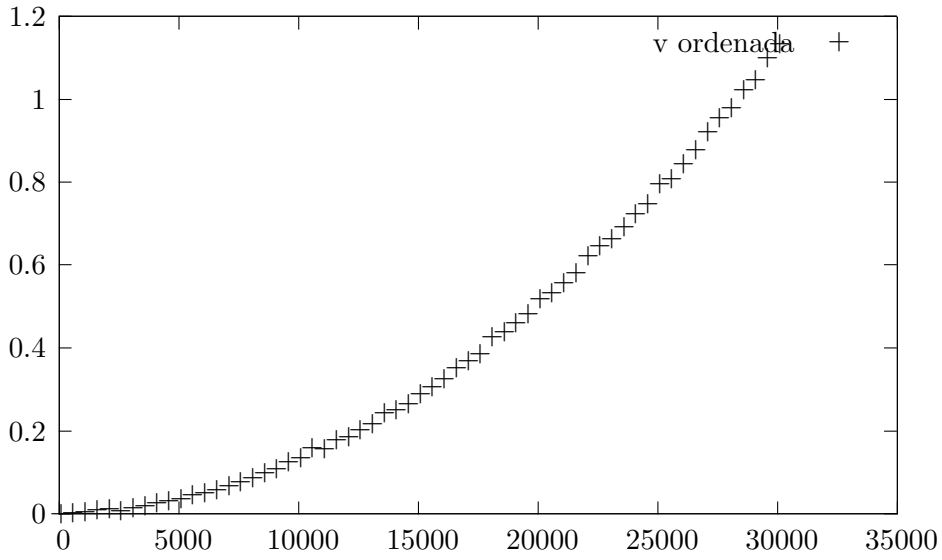
3.1. Mejor caso posible: Vector ordenado

Retomando el ejercicio de ordenación por el algoritmo burbuja, se ha modificado el código para que el vector de entrada esté ordenado.

La parte de código modificado es la siguiente:

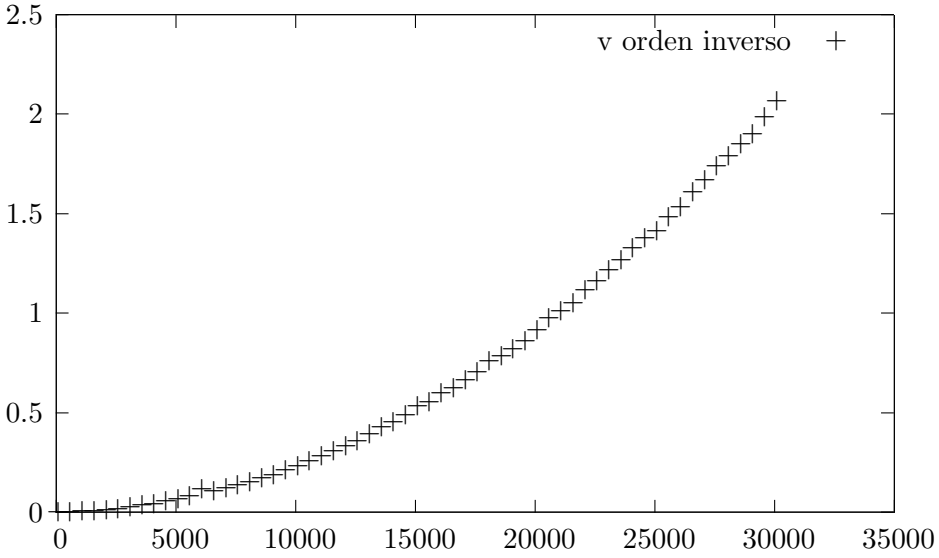
```
1  long double suma=0.0;
2  long double media=0.0;
3  for (int i=0; i<tam; i++)
4    v[i] = i;
5  for(int j=0; j<1000000;j++){
6    srand(time(0));
7
8    clock_t tini;
9    tini=clock();
10
11
12   operacion(v,tam,tam+1,0,tam-1);
13
14   clock_t tfin;
15   tfin=clock();
16   suma +=(tfin-tini)/(double)CLOCKS_PER_SEC;
17   }
18
19 media=suma/1000000.0;
20
21
22 cout << tam << "\t" << media << endl;
```

Su tiempo de ejecución viene representado en el siguiente gráfico:

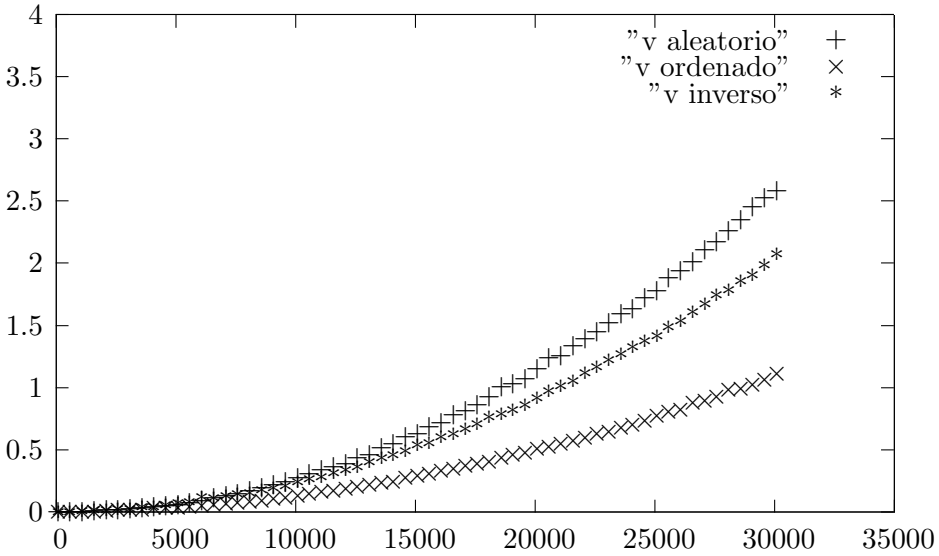


3.2. Vector orden inverso

Para que sea el peor caso posible, el vector debe de estar ordenado de forma inversa. Su eficiencia empírica viene representada en la siguiente gráfica:



3.3. Comparación ejercicio 1



Como podemos comprobar, el vector que no está ordenado es el que más tiempo tarda en ordenarse. El segundo que más tiempo tarda en ordenarse es el inverso, y el que menos tiempo tarda es el ordenado, porque como bien he dicho, ya está ordenado (el tiempo viene representado basicamente

por el tiempo que tarda en comprobar si está ordenado). El vector que no está ordenado en ningún sentido tarda más que el ordenado inverso debido a que hay comprobaciones en las cuales no tiene que ordenar y otras veces que si, con lo cual tarda más. El vector con orden inverso en cada iteración del bucle for está ordenando, con lo cual aprovecha mejor el tiempo de ordenación que el no ordenado.