Práctica 1: Eficiencia

Ejercicio 1: Ordenación de la burbuja

A) Código fuente

```
#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; 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 << "Se genera un vector de tamaño TAM con elementos aleatorios en [0,VMAX[" << endl;
  exit(EXIT_FAILURE);
}

int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=3)
    sintaxis();
  int tam=atoi(argv[1]);     // Tamaño del vector
  int vmax=atoi(argv[2]);    // Valor máximo
  if (tam<=0 || vmax<=0)
    sintaxis();

  // Generación del vector aleatorio
  int *v=new int[tam];       // Reserva de memoria
  srand(time(0));            // Inicialización del generador de números pseudoaleatorios
  for (int i=0; i<tam; i++)  // Recorrer vector
    v[i] = rand() % vmax;    // Generar aleatorio [0,vmax[

  clock_t tini;    // Anotamos el tiempo de inicio
  tini=clock();

  int x = vmax+1;  // Buscamos un valor que no está en el vector
  ordenar(v,tam); // de esta forma forzamos el peor caso

  clock_t tfin;    // Anotamos el tiempo de finalización
  tfin=clock();

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

  delete [] v;     // Liberamos memoria dinámica
}
```

B) Hardware usado

CPU: Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz x8
Memoria: 8GB

C) Sistema operativo

lsb_release -a

No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 14.04.3 LTS
Release: 14.04
Codename: trusty

D) Compilador y opciones de compilación

ignacio@ignacio-GE60-2PE:~$ g++ -v
Using built-in specs.
{...}
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)

Sin opciones especiales de compilación

E) Desarrollo completo del cálculo de la eficiencia teórica

Eficiencia teórica:

```
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}
```

Linea 5,6 y 7 O(1) por la regla de la suma
Linea 4 condicional O(1)
Linea 3 bucle for O(n)
Linea 2 bucle for O(n^2) por la regla del producto

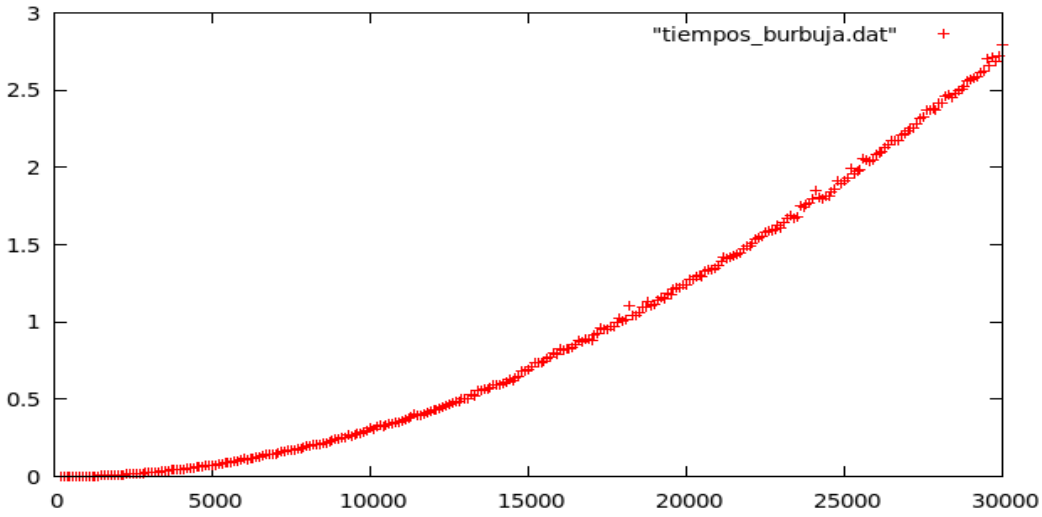
Eficiencia teórica total O(n^2)

F) Parámetros usados para el cálculo de la eficiencia empírica y gráfica

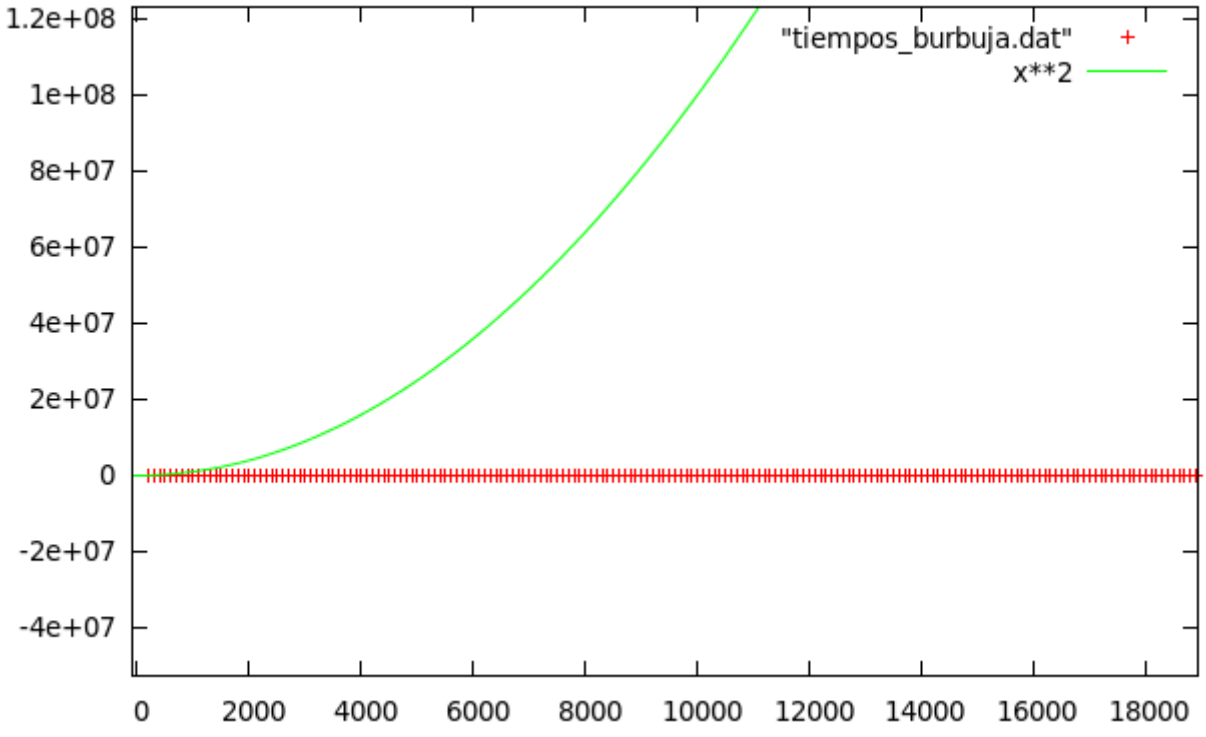
Datos:

inicio = 100
fin = 30000

Gráfica:



G) Pruebe a dibujar superpuestas la función con la eficiencia teórica y la empírica. ¿Qué sucede? Que los datos experimentales crecen muchisimo mas lento que los teoricos.



Ejercicio 2: Ajuste en la ordenación burbuja

A) Código fuente

El mismo que Ejercicio 1.

B) Hardware usado

CPU: Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz x8
Memoria: 8GB

C) Sistema operativo

lsb_release -a

No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 14.04.3 LTS
Release: 14.04
Codename: trusty

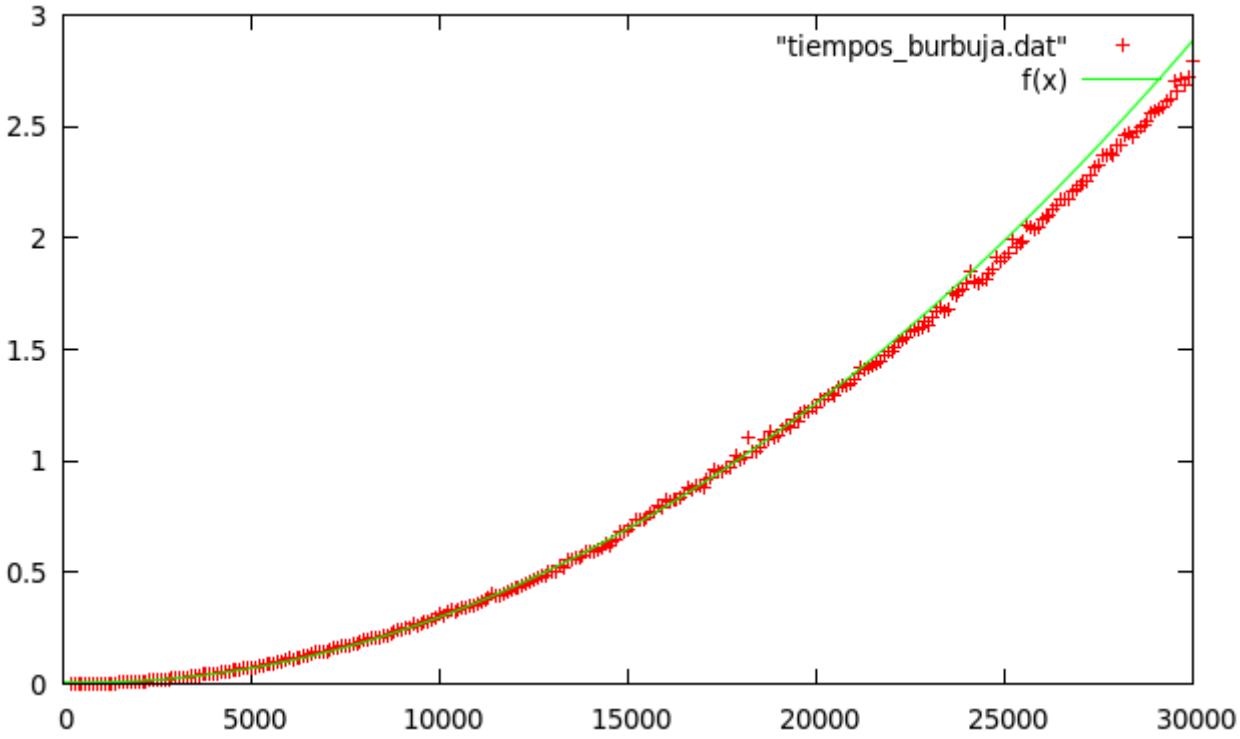
D) Compilador y opciones de compilación

ignacio@ignacio-GE60-2PE:~$ g++ -v
Using built-in specs.
{...}
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)

Sin opciones especiales de compilación

E) Replique el experimento de ajuste por regresión a los resultados obtenidos en el ejercicio 1 que calculaba la eficiencia del algoritmo de ordenación de la burbuja. Para ello considere que f(x) es de la forma ax2+bx+c.

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 3.33545e-09      +/- 2.913e-11    (0.8733%)
b               = -4.16814e-06     +/- 5.741e-07    (13.77%)
c               = 0.00899213       +/- 0.002377     (26.43%)
```



Ejercicio 3: Problemas de precisión

A) Código fuente

```
#include <cstdlib>  // Para generación de números pseudoaleatorios

using namespace std;

int operacion(int *v, int n, int x, int inf, int sup) {
  int med;
  bool enc=false;
  while ((inf<sup) && (!enc)) {
    med = (inf+sup)/2;
    if (v[med]==x)
      enc = true;
    else if (v[med] < x)
      inf = med+1;
    else
      sup = med-1;
  }
  if (enc)
    return med;
  else
    return -1;
}

void sintaxis()
{
  cerr << "Sintaxis:" << endl;
  cerr << "  TAM: Tamaño del vector (>0)" << endl;
  cerr << "Se genera un vector de tamaño TAM con elementos aleatorios" << endl;
  exit(EXIT_FAILURE);
}

int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=2)
    sintaxis();
  int tam=atoi(argv[1]);     // Tamaño del vector
  if (tam<=0)
    sintaxis();

  // Generación del vector aleatorio
  int *v=new int[tam];       // Reserva de memoria
  srand(time(0));            // Inicialización del generador de números pseudoaleatorios
  for (int i=0; i<tam; i++)  // Recorrer vector
    v[i] = rand() % tam;

  clock_t tini;    // Anotamos el tiempo de inicio
  tini=clock();

  // Algoritmo a evaluar
  operacion(v,tam,tam+1,0,tam-1);

  clock_t tfin;    // Anotamos el tiempo de finalización
  tfin=clock();

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

  delete [] v;     // Liberamos memoria dinámica
}
```

B) Hardware usado

CPU: Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz x8
Memoria: 8GB

C) Sistema operativo

lsb_release -a

No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 14.04.3 LTS
Release: 14.04
Codename: trusty

D) Compilador y opciones de compilación

ignacio@ignacio-GE60-2PE:~$ g++ -v
Using built-in specs.
{...}
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)

Sin opciones especiales de compilación

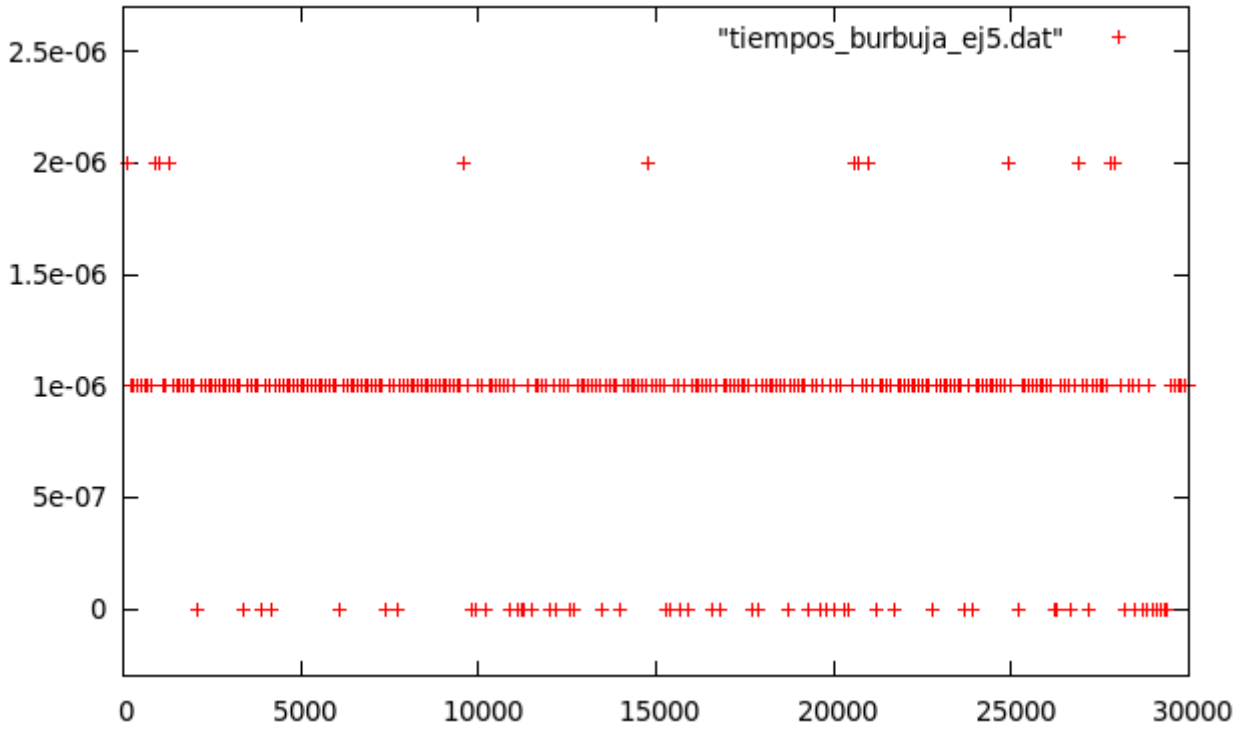
E) Desarrollo completo del cálculo de la eficiencia teórica

```
1 int operacion(int *v, int n, int x, int inf, int sup) {
2       int med;
3       bool enc=false;
4       while ((inf<sup) && (!enc)) {
5               med = (inf+sup)/2;
6               if (v[med]==x)
7                       enc = true;
8               else if (v[med] < x)
9                       inf = med+1;
10              else
11                      sup = med-1;
12      }
13      if (enc)
14              return med;
15       else
16              return -1;
17 }
```

El orden de eficiencia en el peor de los casos de este algoritmo es **O(log (n))**, ya que en el peor de los casos (cuando el vector este desordenado por completo) los indicices inf y sup solo recorrerán hasta la mitad del vector.

F) Parámetros usados para el cálculo de la eficiencia empírica y ajuste

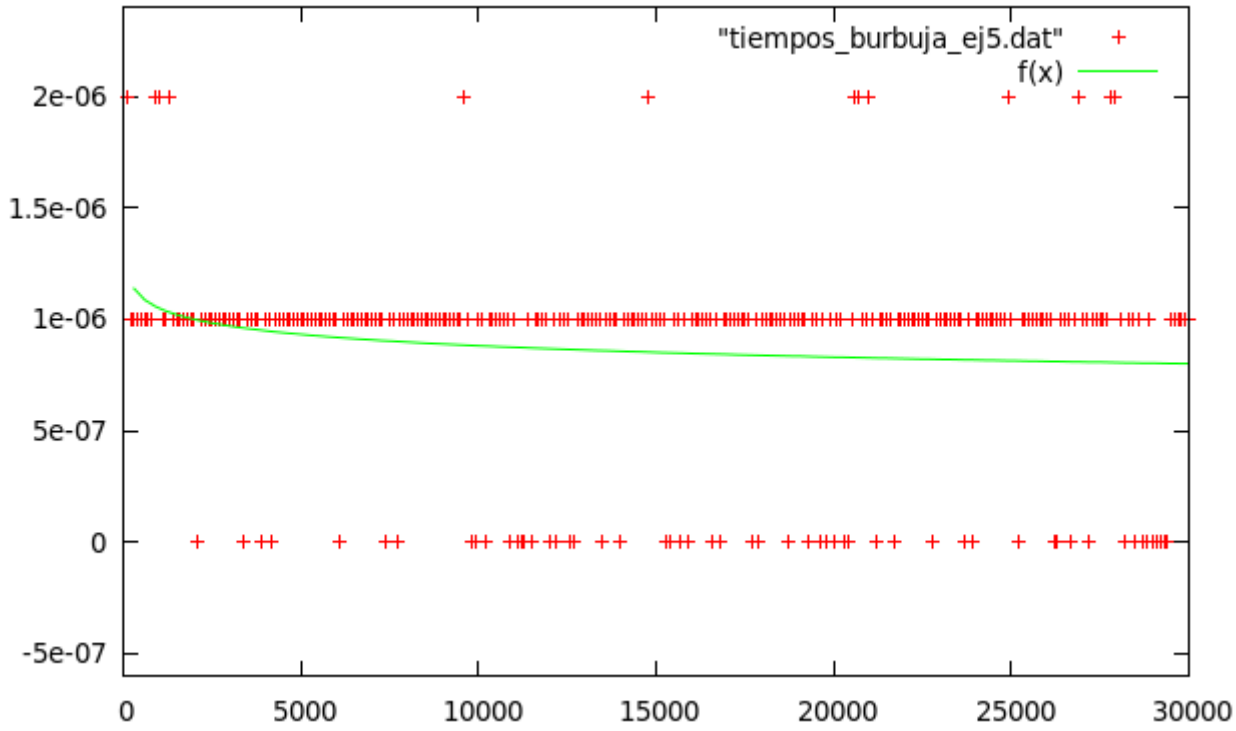
En este caso y dado que la ejecución de los programas es demasiado rápida necesitariamos otra herramienta de medida de tiempo para obtener valores con mayor precisión. Aun así, las gráficas de la eficiencia así como el ajuste son:



Ajuste:

La función a ajustar en este caso es f(x) = a*log(x)+b:

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = -7.33853e-08     +/- 2.896e-08    (39.47%)
b               = 1.5574e-06       +/- 2.714e-07    (17.43%)
```



G) ¿Qué hace este programa?

El código de la función operación trata de buscar la posición de un elemento de un vector mediante el algoritmo de búsqueda dicotómica. Este algoritmo compara el elemento a buscar con un elemento cualquiera del array (el elemento central en este caso): si el valor de éste es mayor que el del elemento buscado se repite el procedimiento en la parte del array que va desde el inicio de éste hasta el elemento tomado, en caso contrario se toma la parte del array que va desde el elemento tomado hasta el final. De esta manera obtenemos intervalos cada vez más pequeños, hasta que se obtenga un intervalo indivisible. Si el elemento no se encuentra dentro de este último entonces se deduce que el elemento buscado no se encuentra en todo el array.

Este algoritmo es muy útil cuando tenemos un vector previamente ordenado.

Ejercicio 4: Mejor y peor caso

A) Código fuente

- Mejor caso:

La única modificación realizada con respecto al código del ejercicio 1 ha sido:

```
for (int i=0; i<tam; i++)  // Recorrer vector
        v[i] = i;    // asigna valores ordenados al vector
```

- Peor caso:

La única modificación realizada con respecto al código del ejercicio 1 ha sido:

```
for (int i=0; i<tam; i++)  // Recorrer vector
        v[i] = tam-i;    // asigna valores ordenados a la inversa al vector
```

B) Hardware usado

CPU: Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz x8
Memoria: 8GB

C) Sistema operativo

lsb_release -a

No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 14.04.3 LTS
Release: 14.04
Codename: trusty

D) Compilador y opciones de compilación

ignacio@ignacio-GE60-2PE:~$ g++ -v
Using built-in specs.
{...}
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)

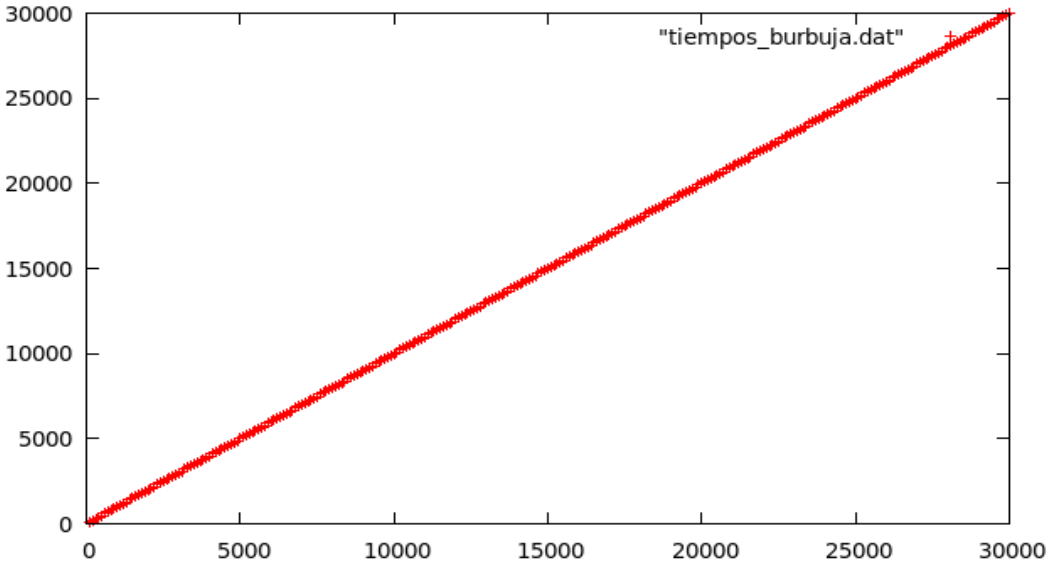
Sin opciones especiales de compilación

E) Desarrollo completo del cálculo de la eficiencia teórica y gráfica

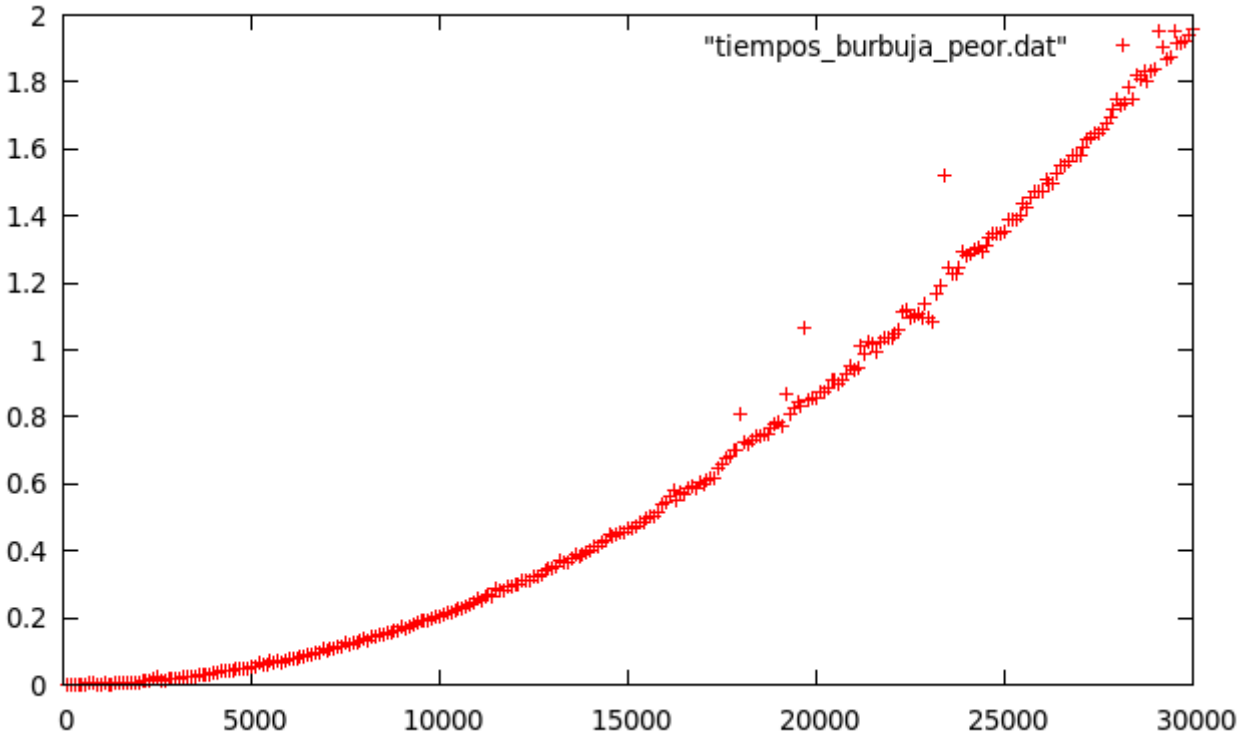
Igual que en el ejercicio 1

F) Parámetros usados para el cálculo de la eficiencia empírica y gráfica

- Mejor caso:
  Datos:
  inicio = 100, fin = 30000, incremento = 100



- Peor caso:
  Datos:
  inicio = 100, fin = 30000, incremento = 100

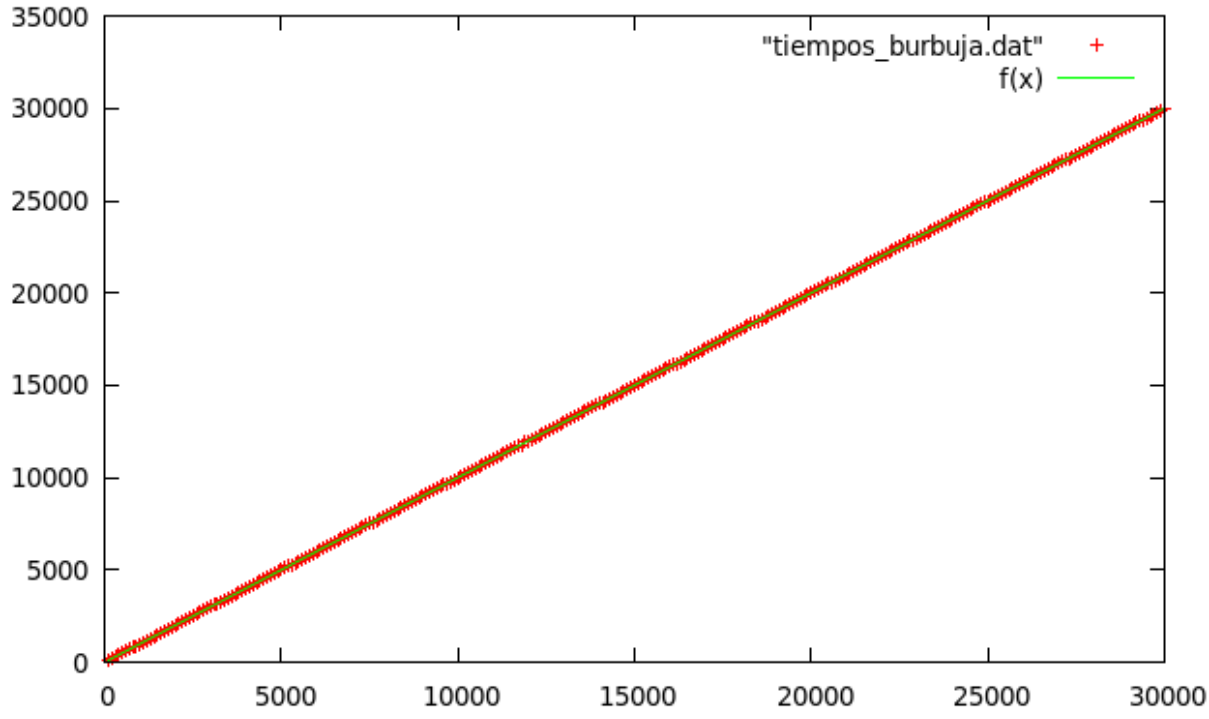


Como se puede apreciar, la gráfica de la eficiencia empírica para el peor de los casos es muy similar a la calculada teóricamente y para casos aleatorios, mientras que para el mejor de los casos la gráfica sigue un desarrollo más parecido a una recta (n) que a una parábola (n^2)

G) Ajuste de la curva teórica a la empírica: mostrar resultados del ajuste y gráfica

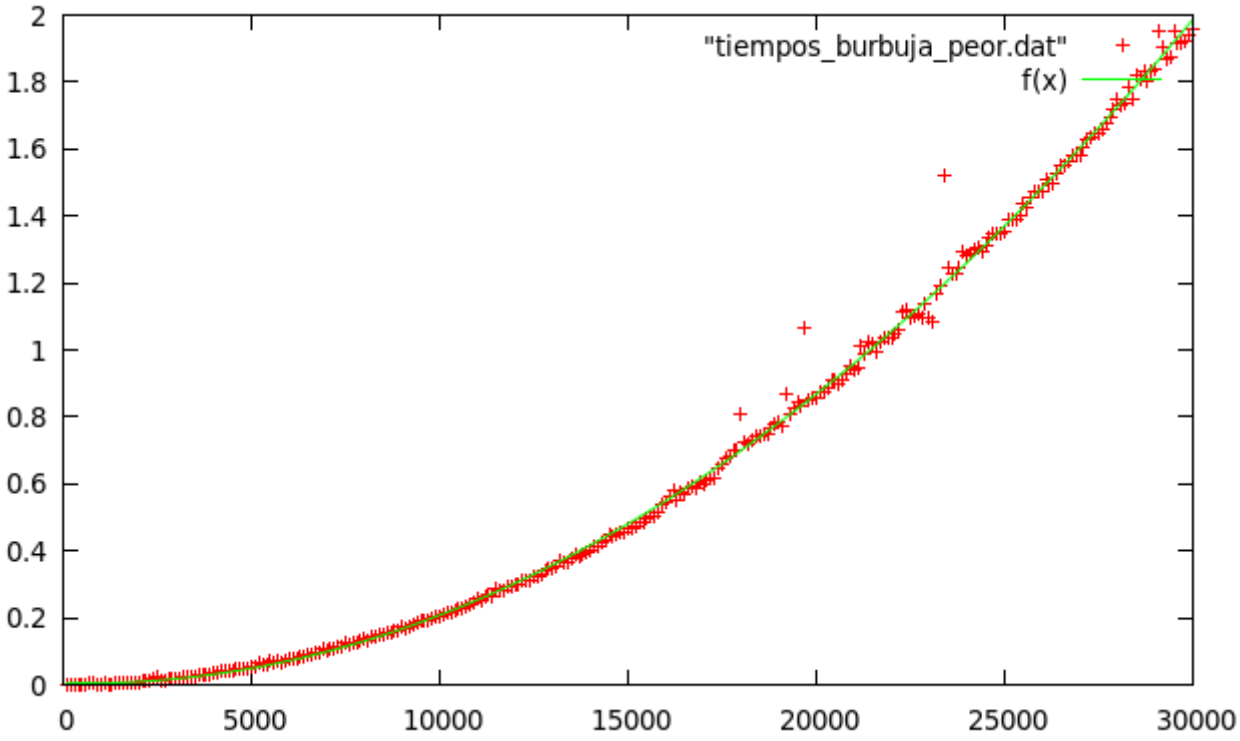
- Mejor caso

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 1.92101e-18      +/- 1.322e-20    (0.6884%)
b               = 1                +/- 5.202e-16    (5.202e-14%)
c               = 2.47464e-10      +/- 1.438e-11    (5.81%)
```



- Peor caso:

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 2.2885e-09       +/- 2.426e-11    (1.06%)
b               = -2.63832e-06     +/- 7.539e-07    (28.57%)
c               = 0.00597093       +/- 0.004914     (82.3%)
```



Ejercicio 5: Dependencia de la implementación

A) Código fuente

La nueva función burbuja es:

```
void ordenar(int *v, int n) {
  bool cambio=true;
  for (int i=0; i<n-1 && cambio; i++) {
    cambio=false;
    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;
      }
  }
}
```

B) Hardware usado

CPU: Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz x8
Memoria: 8GB

C) Sistema operativo

lsb_release -a

No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 14.04.3 LTS
Release: 14.04
Codename: trusty

D) Compilador y opciones de compilación

ignacio@ignacio-GE60-2PE:~$ g++ -v
Using built-in specs.
{...}
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)

Sin opciones especiales de compilación

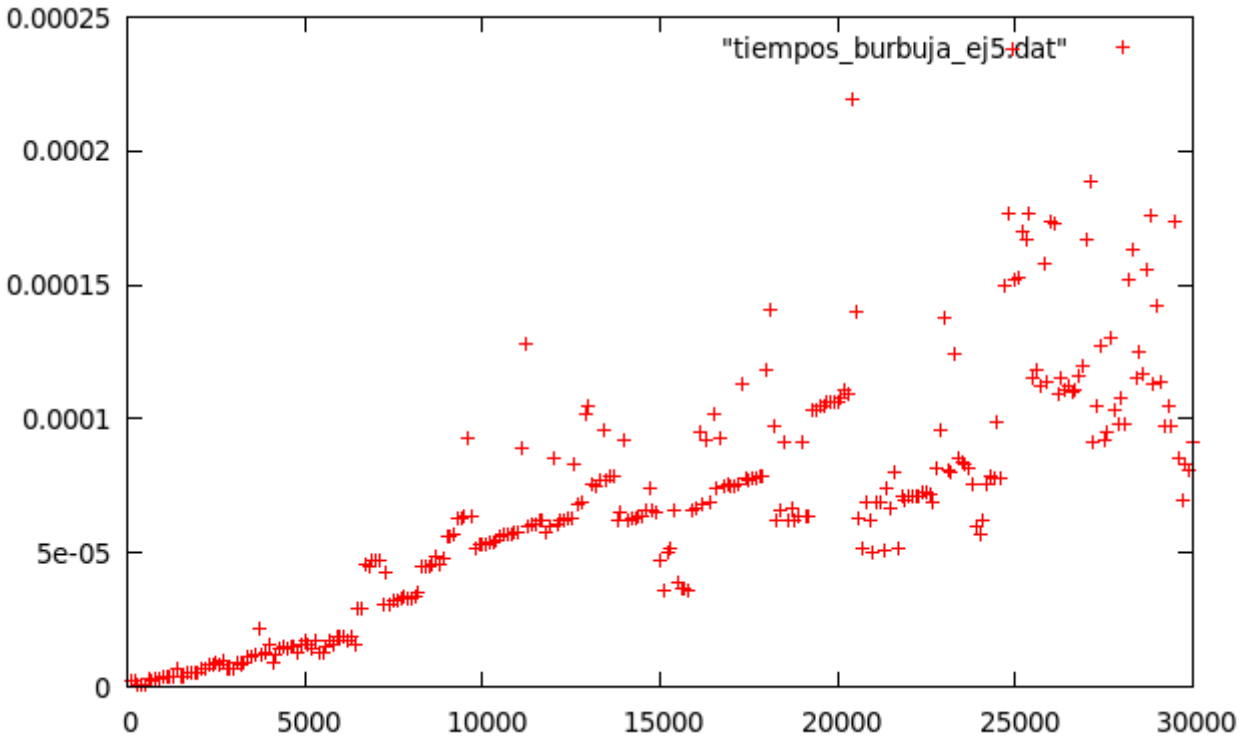
E) Desarrollo completo del cálculo de la eficiencia teórica en el mejor de los casos

En el mejor caso es orden n ya que supone que el bucle externo (el que tiene como iterador la variable i) solo se hace una vez porque cambio nunca llega a ser true.

F) Parámetros usados para el cálculo de la eficiencia empírica y gráfica

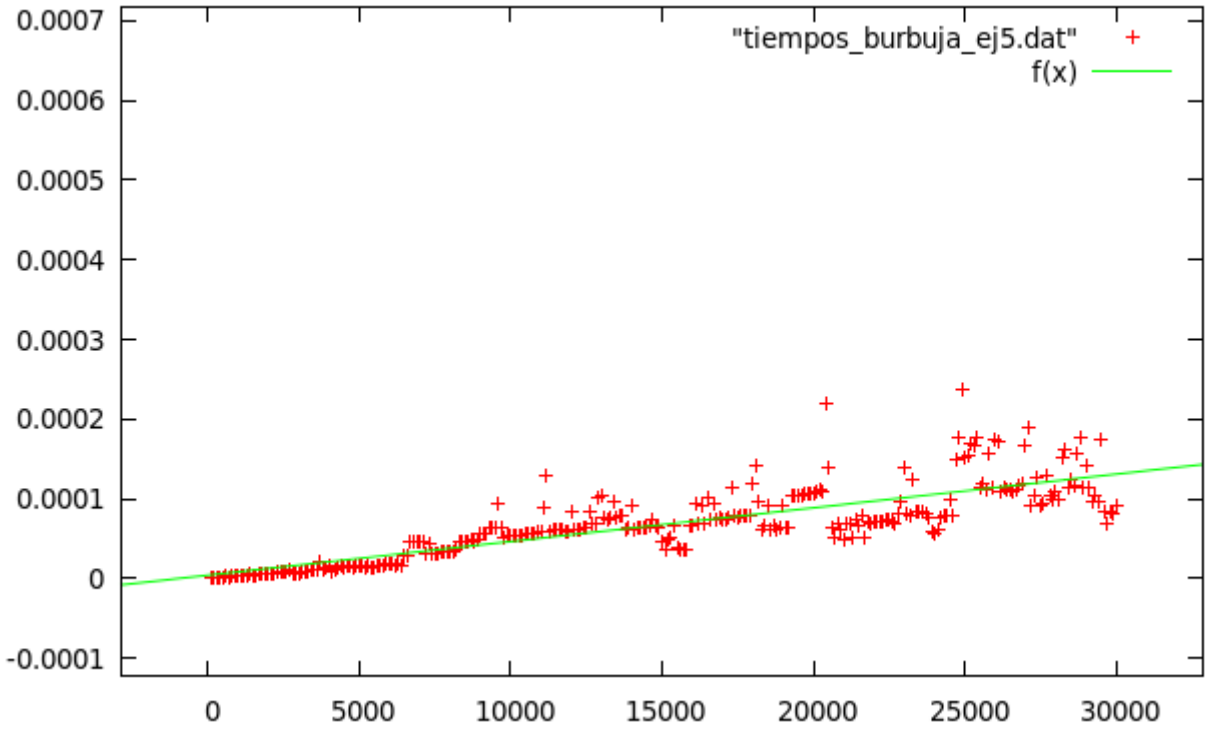
Datos:

inicio = 100, fin = 30000, incremento = 100



G) Ajuste de la curva teórica a la empírica: mostrar resultados del ajuste y gráfica

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 2.2885e-09       +/- 2.426e-11    (1.06%)
b               = -2.63832e-06     +/- 7.539e-07    (28.57%)
c               = 0.00597094       +/- 0.004914     (82.3%)
```



Por lo general la gráfica si se ajusta a los valores predichos aunque para un numero alto de tamaño de vector la dispersión de la muestra aumenta.