PRÁCTICA 1

Blanca Cano Camarero DGIIM

Sumario

Ejercicio 1: Ordenación de la burbuja.

Código del programa ordenacion.cpp

```
#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();
```

```
  ordenar(v , tam);

  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
```

Contenido del script ejecuciones_ordenacion.csh

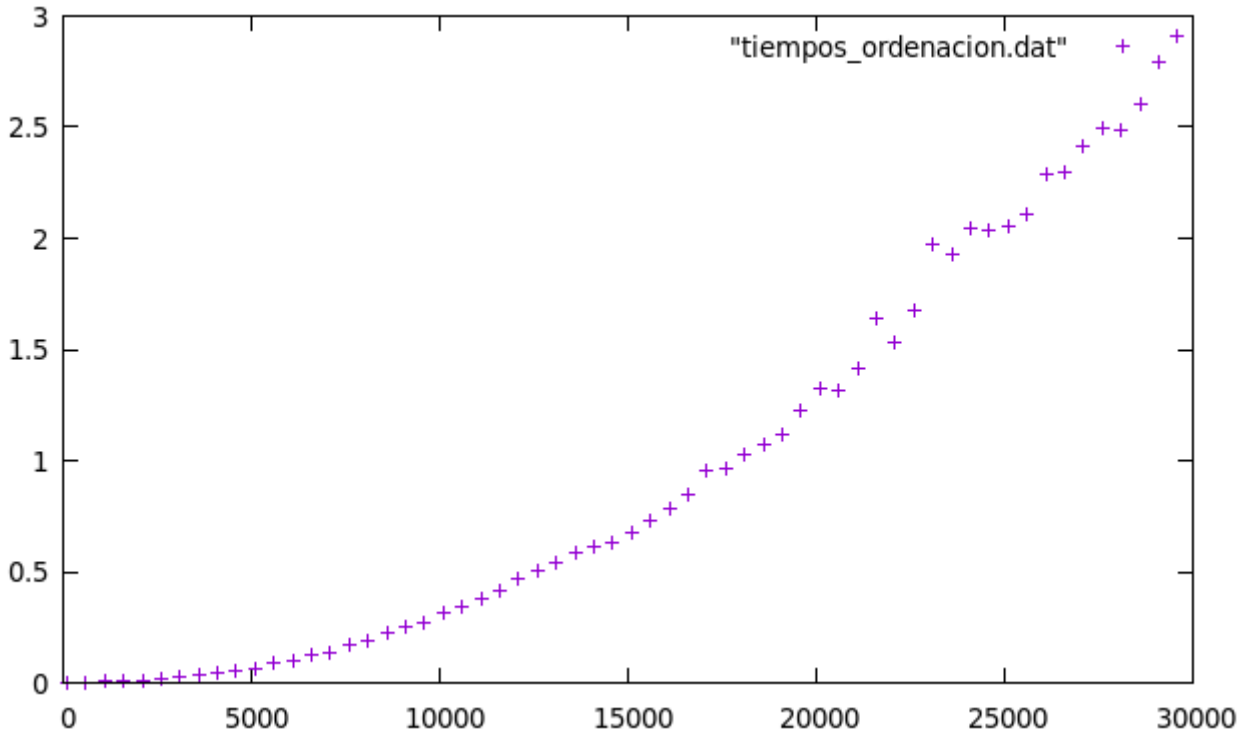
```
#!/bin/bash
inicio=100
fin=30000
incremento=500

ejecutable="ordenacion"
salida="tiempos_ordenacion.dat"

i=$inicio
echo > $salida
while [ $i -lt $fin ]
do
  echo "Ejecución tam = " $i
  echo `./$ejecutable $i 10000` >> $salida
  i=$[$i+$incremento]
done
```

Fichero obtenido tras interpretar el scritp

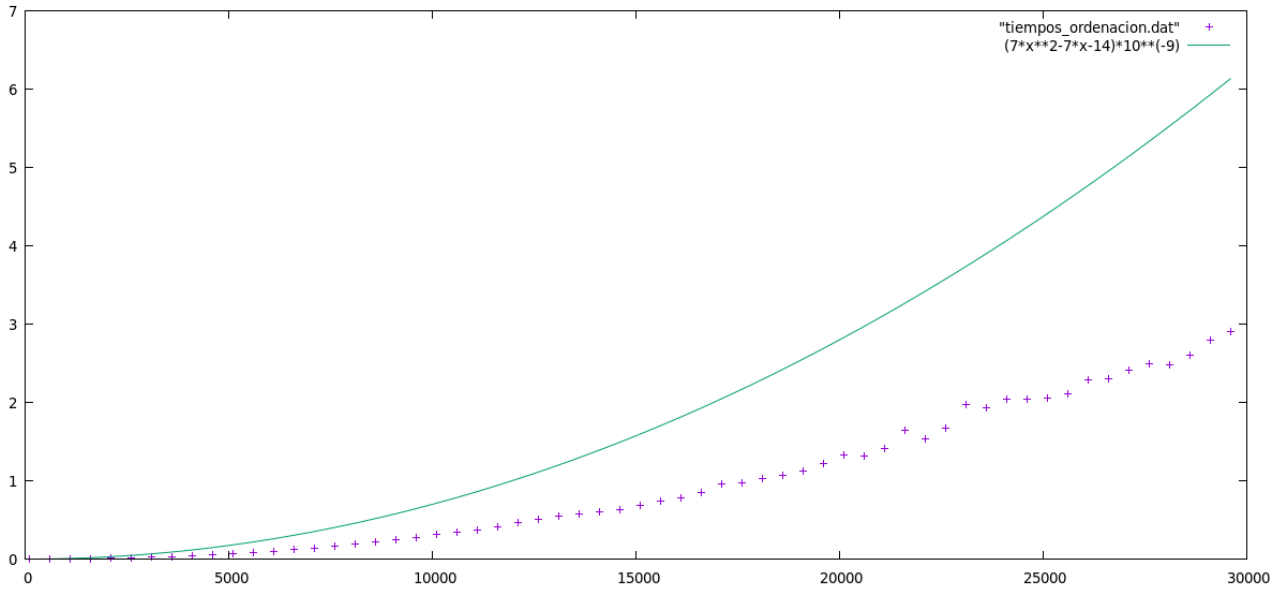
```
100 0.000154
600 0.003613
1100 0.006507
1600 0.006732
2100 0.012482
2600 0.018764
3100 0.024713
3600 0.033651
4100 0.04186
4600 0.057689
5100 0.065561
5600 0.087604
6100 0.096385
6600 0.123801
7100 0.136061
7600 0.170888
8100 0.186658
8600 0.223907
9100 0.25078
9600 0.273954
10100 0.316989
```

```
10600 0.347074
11100 0.374842
11600 0.415539
12100 0.466978
12600 0.507908
13100 0.545525
13600 0.582676
14100 0.610923
14600 0.630298
15100 0.679047
15600 0.734593
16100 0.787822
16600 0.847526
17100 0.952914
17600 0.968783
18100 1.0233
18600 1.07381
19100 1.12051
19600 1.2213
20100 1.32561
20600 1.31165
21100 1.41591
21600 1.64471
22100 1.52853
22600 1.67544
23100 1.97132
23600 1.92866
24100 2.0448
24600 2.03817
25100 2.05574
25600 2.112
26100 2.28652
26600 2.2955
27100 2.41158
27600 2.49272
28100 2.48531
28600 2.60009
29100 2.79336
29600 2.90663
```

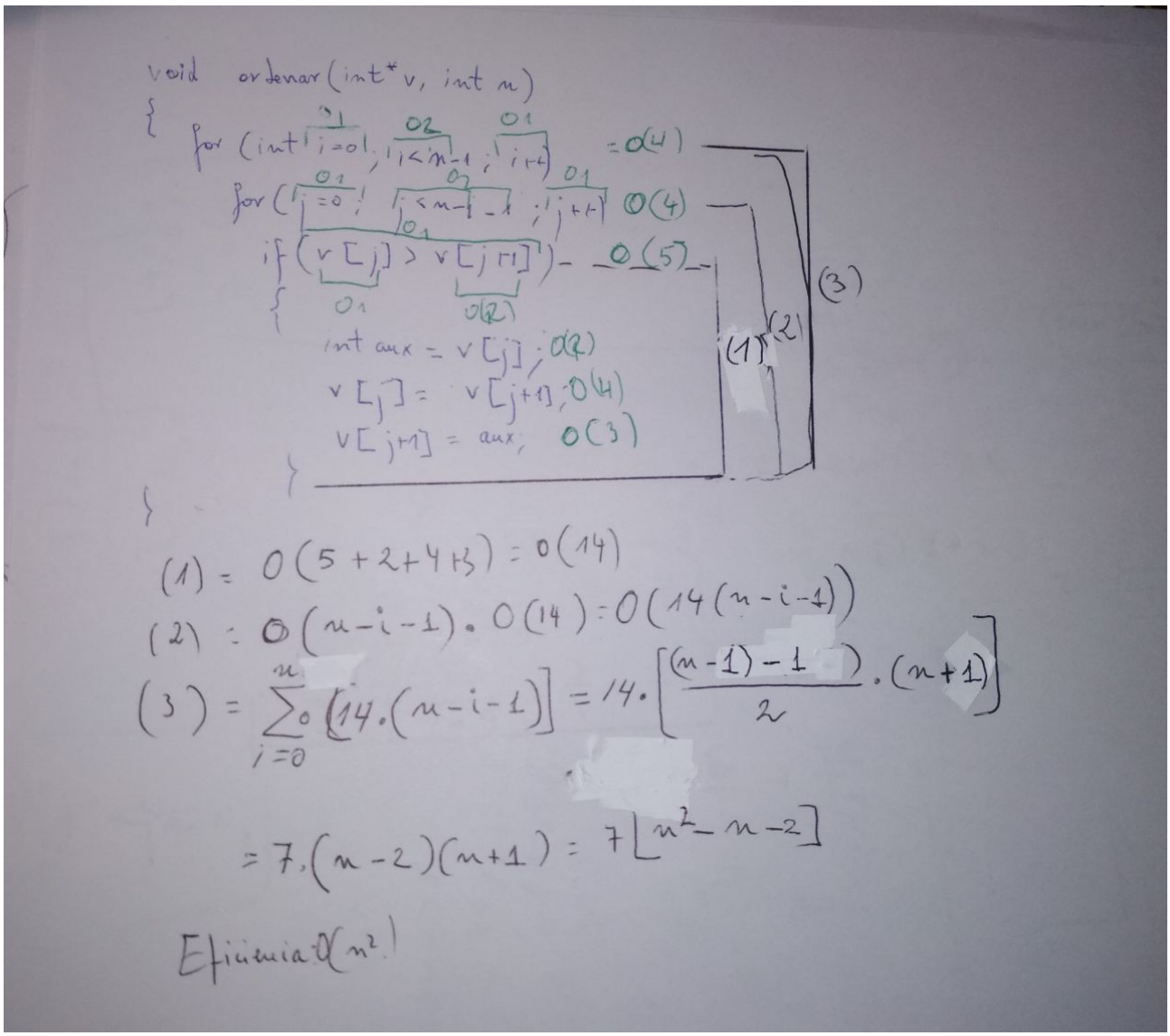
Representación de los datos con gnuplot:



Superponiendo curva teórica:



La curva teórica ha sido obtenida de la siguiente manera:



$(1) = O(5+2+4+3) = O(14)$

$(2) = O(n-i-1) \cdot O(14) = O(14(n-i-1))$

$(3) = \sum_{i=0}^{n} [14 \cdot (n-i-1)] = 14 \cdot \left[\frac{(n-1)-1}{2} \cdot (n+1) \right]$

$= 7 \cdot (n-2)(n+1) = 7[n^2 - n - 2]$

Eficiencia $O(n^2)$

Ejercicio 2: Ajuste en la ordenación de la burbuja

Tras ajustar por regresión los datos obtenidos de manera experimental, los coeficientes de regresión resultan:

Final set of parameters		Asymptotic Standard Error	
==========		==========	
a	= 3.45267e-09	+/- 1.017e-10	(2.944%)
b	= -4.39016e-06	+/- 3.12e-06	(71.07%)
c	= 0.000831158	+/- 0.02005	(2412%)

Ejercicio 3: Problemas de precisión

Código al que analizas su eficiencia

```
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;
}
```

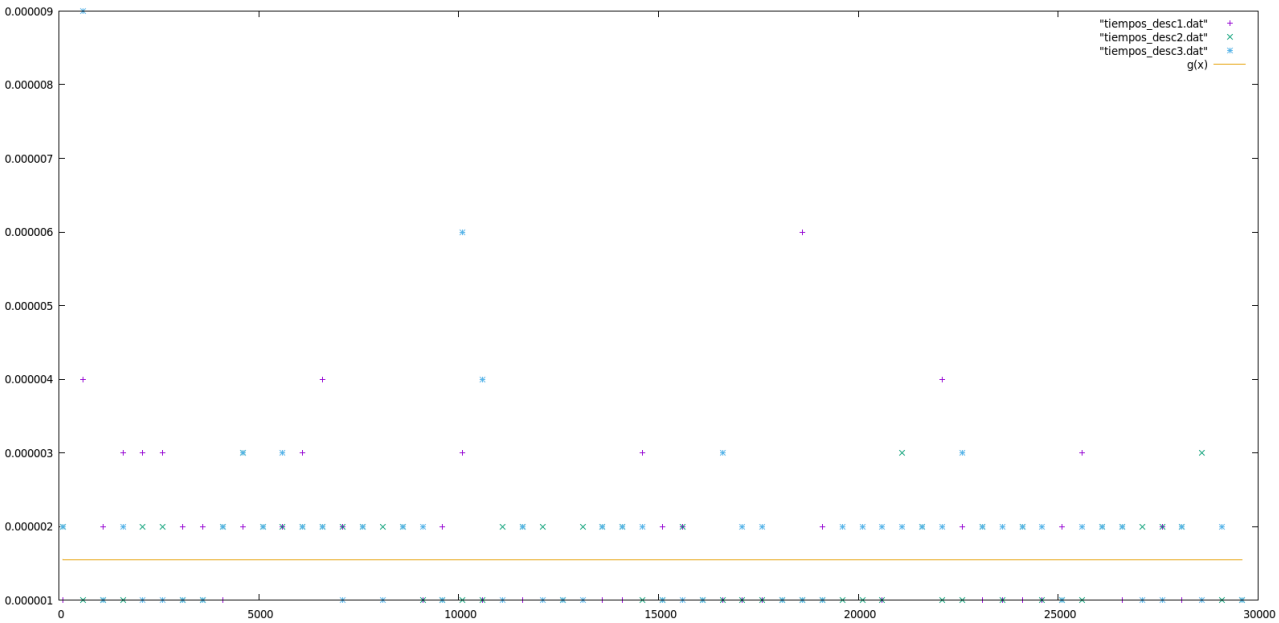
Eficiencia teórica: O(log_2(n))

Descripción del algoritmo:

Como argumentos en orden, esta función recibe un vector de enteros , su longitud, un valor x, que se entiende que se encontrará entre el rango de valores de enteros que hay dentro del vector. Las dos últimas variables hacen referencia a la primera posición a empezar del array y la última.

En el teórico caso de que el vector estuviera ordenado este algoritmo devolvería la posición en la que se encontra el valor x y en su defecto un -1.

La gráficas obtenida tras varias ejecuciones:



donde g(x) es la gráfica ajusta a $\log_2(1)$ por regresión lo cual es una constate.

Quizás esto pueda extrañar, ya que no coincide con su eficiencia teórica, una posible explicación es que en la mayoría de los casos, no se da el peor de los casos que es que no se encuentre el número.

Ejercicio 4: Mejor y peor caso

Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Debe modificar el código que genera los datos de entrada para situarnos en dos escenarios diferentes:

• El mejor caso posible. Para este algoritmo, si la entrada es un vector que ya está ordenado el tiempo de cómputo es menor ya que no tiene que intercambiar ningún elemento.

• El peor caso posible. Si la entrada es un vector ordenado en orden inverso estaremos en la peor situación posible ya que en cada iteración del bucle interno hay que hacer un intercambio.

Calcule la eficiencia empírica en ambos escenarios y compárela con el resultado del ejercicio 1.

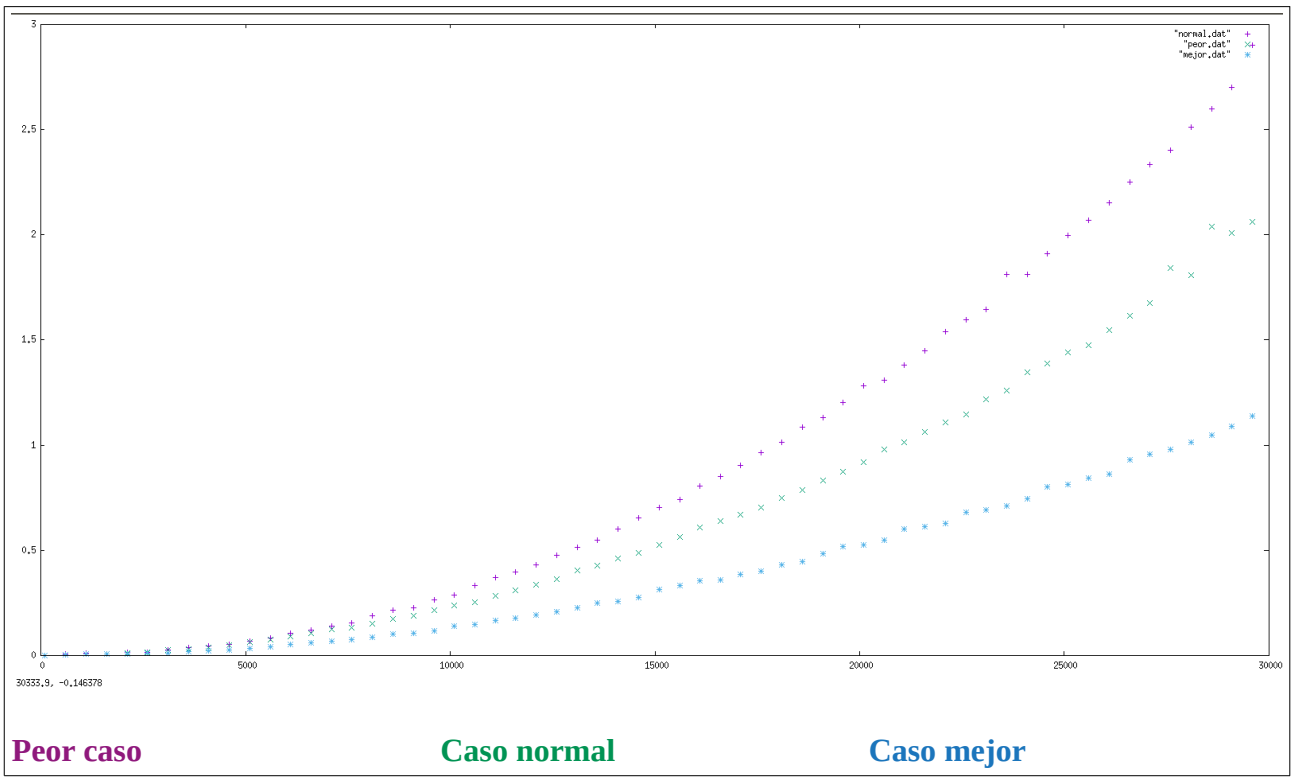
Ajuste en el mejor de los casos:

```
Final set of parameters          Asymptotic Standard Error
======================          ==========================
a               = 1.24742e-09     +/- 1.483e-11    (1.189%)
b               = 1.35183e-06     +/- 4.553e-07    (33.68%)
c               = -0.00220833     +/- 0.002925     (132.5%)
```

Ajuste en el peor de los casos:

```
Final set of parameters          Asymptotic Standard Error
======================          ==========================
a               = 2.44274e-09     +/- 4.845e-11    (1.983%)
b               = -3.63381e-06    +/- 1.487e-06    (40.92%)
c               = 0.0156758       +/- 0.009554     (60.95%)
```

Comparación visual de la ejecución:



Peor caso **Caso normal** **Caso mejor**

Ejercicio 5: Dependencia de la implementación

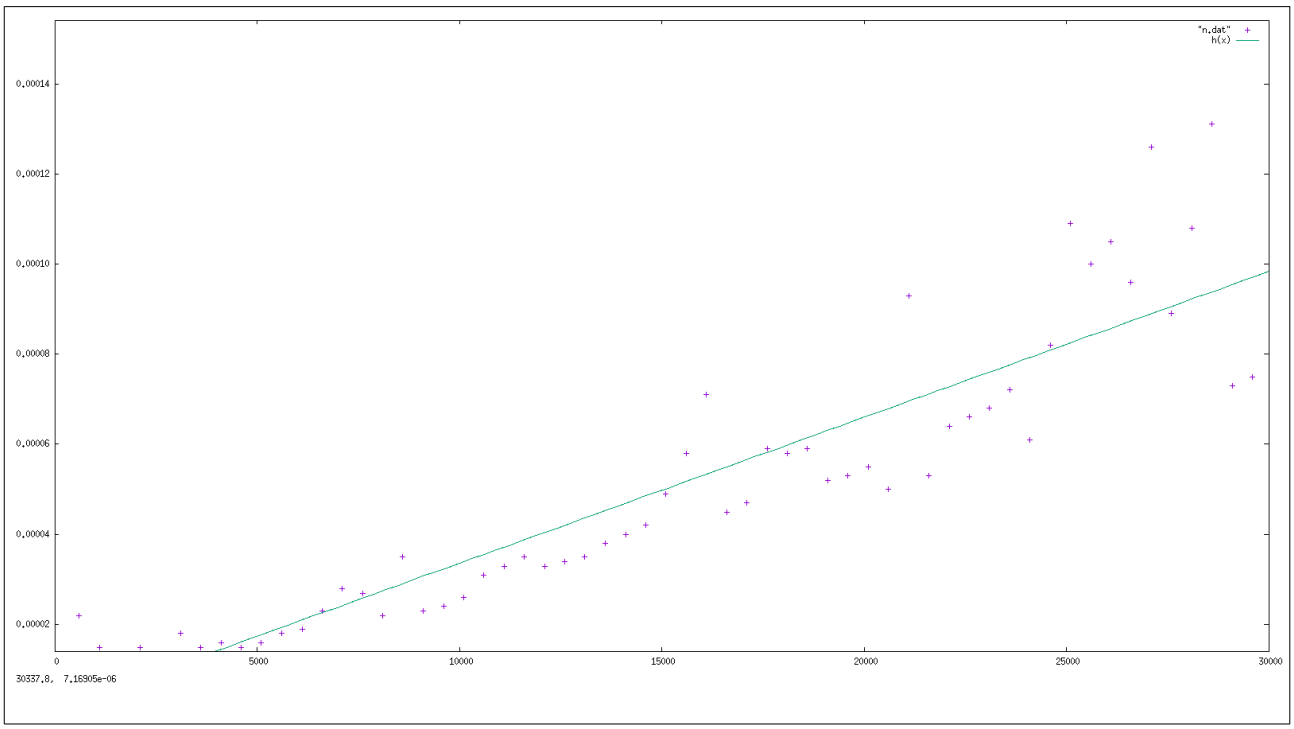
Considere esta otra implementación del algoritmo de la burbuja:

```
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;
      }
  }
}
```

En ella se ha introducido una variable que permite saber si, en una de las iteraciones del bucle externo no se ha modificado el vector. Si esto ocurre significa que ya está ordenado y no hay que continuar.

Considere ahora la situación del mejor caso posible en la que el vector de entrada ya está ordenado. ¿Cuál sería la eficiencia teórica en ese mejor caso? Muestre la gráfica con la eficiencia empírica y compruebe si se ajusta a la previsión.

La eficiencia teórica en el caso de que se encuentre ordenado es O(k)


"n.dat"
h(x)
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0
5000
10000
15000
20000
25000
30000
30337.8, 7.16905e-06