Práctica 1 ED, ejercicio 3

Javier Gálvez Obispo

Para la realización de la práctica se ha utilizado una máquina virtual con procesador i5-6500, con una frecuencia de reloj de 3.2GHz, limitado a una cpu, 3GB de RAM y ubuntu 16.04 LTS de 32bits como sistema operativo.

Código fuente:

```
#include <iostream>
#include <ctime>    // Recursos para medir tiempos
#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
}
```

El algoritmo implementado realiza la búsqueda binaria.

Eficiencia teórica:

```
int operacion(int *v, int n, int x, int inf, int sup) {
  int med;                  |O(1)                              |
  bool enc=false;           |O(1)                              |
  while ((inf<sup) && (!enc)) {        |                       |
    med = (inf+sup)/2;      |                    |             |
    if (v[med]==x)          |                    |             |
      enc = true;           |                    |O(log2n)     |
    else if (v[med] < x)    | O(1)               |             |
      inf = med+1;          |                    |             |O(log2n)
    else                    |                    |             |
      sup = med-1;          |                    |             |
  }                                                            |
  if (enc)          |                                          |
    return med;     |O(1)                                      |
  else              |                                          |
    return -1;      |                                          |
}
```

La eficiencia teórica es $O(\log_2 n)$

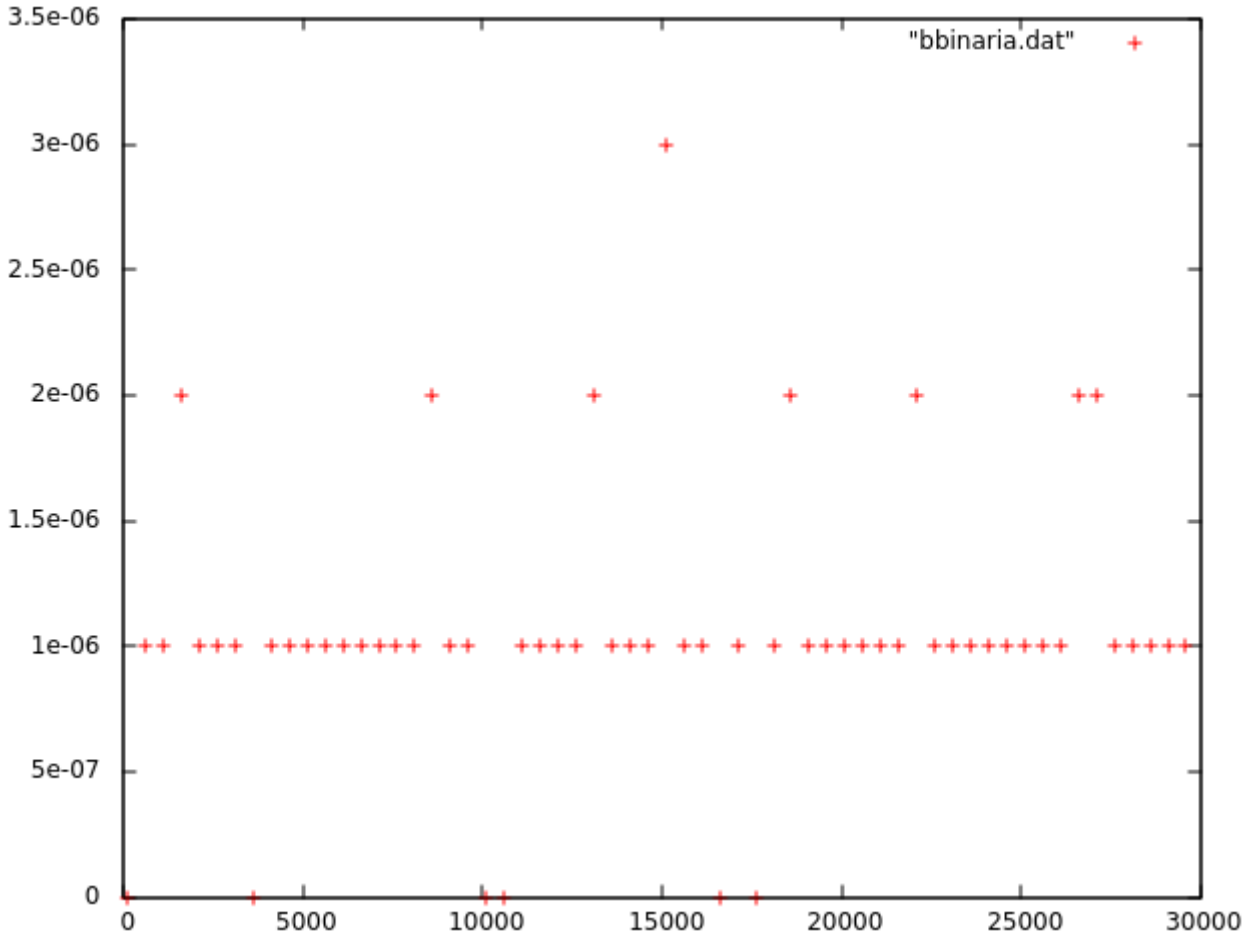
Para la compilación del programa se ha utilizado la siguiente orden:
g++ ejercicio_desc.cpp -o bbinaria

El script utilizado para las ejecuciones es el siguiente:

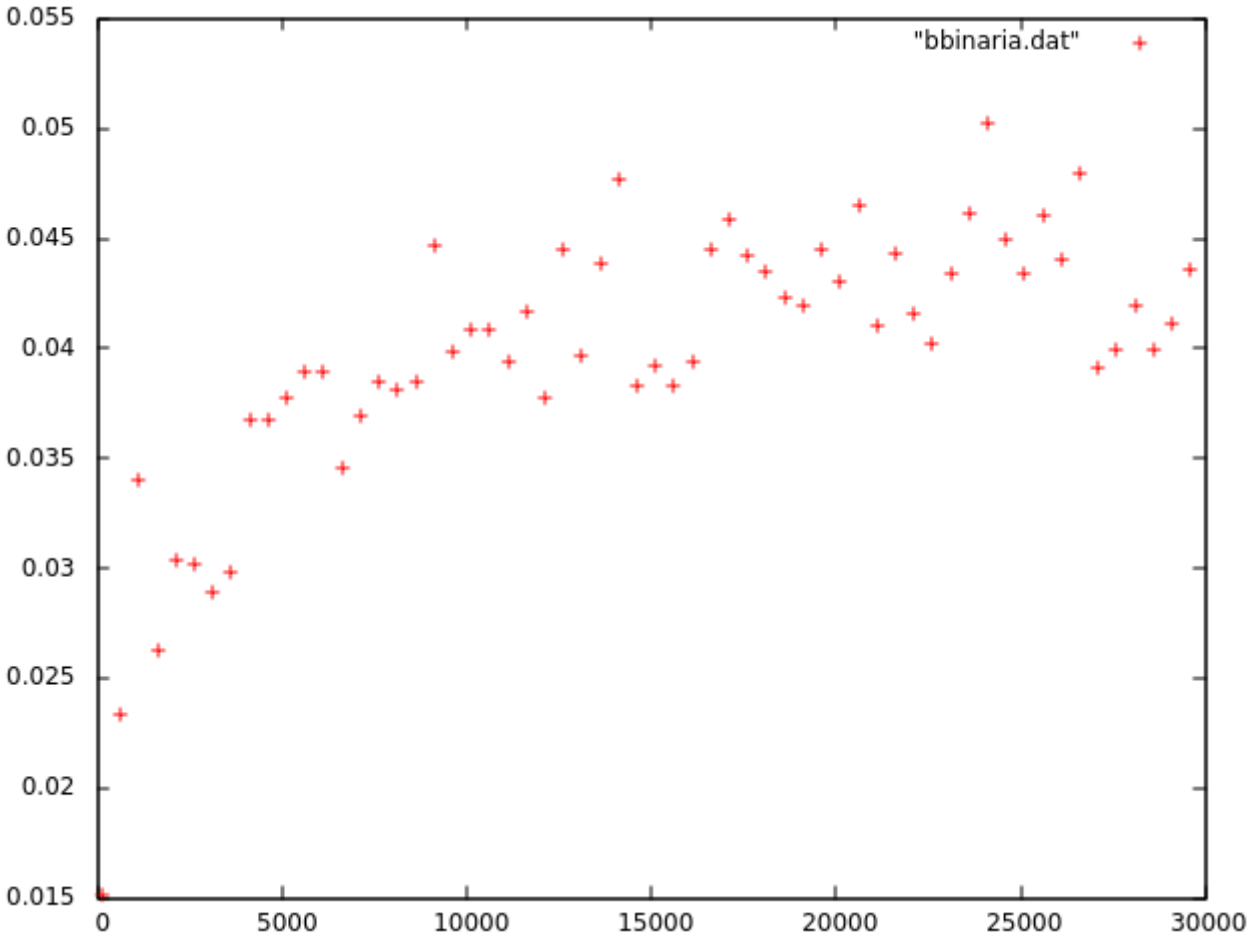
```
#!/bin/bash
inicio=100
fin=30000
incremento=500
ejecutable="bbinaria"
salida="bbinaria.dat"

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

Al dibujar la gráfica obtenemos:



Esta gráfica no muestra la eficiencia real del algoritmo porque los tiempos son muy próximos a 0 y ctime no tiene tanta precisión. Para solucionarlo metemos en un bucle for el algoritmo y luego dividimos el tiempo entre el número de ejecuciones realizadas. Obtenemos así la siguiente gráfica:



Que se aproxima mejor a la gráfica del logaritmo.