Angel Barrilao Bensrhir

Practica 1 – Eficiencia

Ejercicio 3

Características del computador usado:
- Procesador: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
- Ram: 8 GB
- SO: Ubuntu 16.04 (Linux version 4.4.0-96-generic)
- Compilador: gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4)

NOTA: En la compilacion de las pruebas realizadas no se han utilizado flags de optimización ni de otro tipo.

Algoritmo

```
#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=0;
  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;

          operacion(v,n,x,inf,sup);
  }

  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
}
```

script

```
#!/bin/csh
@ inicio = 100
@ fin = 1000000
@ incremento = 100
set ejecutable = ejercicio_desc
set salida = tiempos_desc.dat

@ i = $inicio
echo > $salida
while ( $i <= $fin )
  echo Ejecución tam = $i
  echo `./{$ejecutable} $i` >> $salida
  @ i += $incremento
end
```

Angel Barrilao Bensrhir

Calculo de Eficiencia

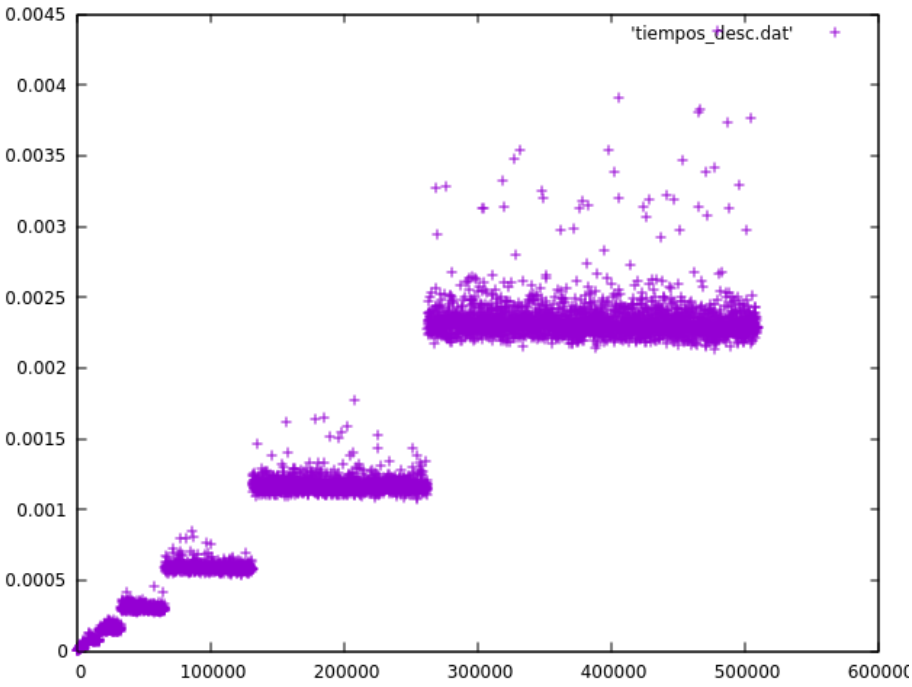
Teórica ?

$$T(n) = 2 + \sum_{inferior=0}^{superior} 3+2+1+2+2+2 \quad + T(n-1) + \ldots + \max(1,1)$$

Empírica ?

Tengo dudas sobre como de bien esta planteado el algoritmo y como se podría representar su función.

Aquí muestro solo los tiempos , sin ajuste a la función. (Haciendo escalones, bastante curioso)



Angel Barrilao Bensrhir

Aclaraciones

Este algoritmo intenta buscar un elemento en mitad de un array.

Su funcionamiento consiste en pasarle cinco parámetros y calcular la media con el principio y final del array, para después con ese numero resultante de la media usarlo para apuntar en la mitad del array.

Entonces:

- Si encuentra el elemento devolverá un booleano (true).

- Si no lo encuentra , pero el valor apuntado es menor que el que buscamos , reducimos una posición desde el inicio del array sumándole uno al principio.

- Si finalmente no coincide con los dos puntos anteriores, entonces decrementamos desde el final del array (reduciéndolo por el final).

Hasta aquí todo correcto. Pero ocurre un problema, los incrementos y decrementos no se hacen efectivos ya que no hay una continuidad, es decir los resultados no se guardan en ninguna parte.

Una posible solución a este problema seria hacer una llamada recursiva pasándole los nuevos valores calculados, hasta que llegue al caso base.

Si lo medimos sin ponerle la función recursiva, nos dará siempre un mismo valor muy pequeño (aprox 1e-06) ya que no hace apenas ninguna operación.

Pero si lo hacemos con una llamada recursiva se puede ir viendo como se va comportando según el tamaño del problema.