Mª del Mar Alguacil Camarero

Práctica 1:Eficiencia

Doble Grado en Ingeniería Informática y Matemáticas

EJERCICIO 3

- Hardware usado:
  - CPU: Cuádruple núcleo Intel® Core™ i5-2430M CPU
  - Velocidad de reloj: 2.40GHz
  - Memoria RAM: 5,8 GiB
- Sistema operativo:
  - elementary OS 0.3 Freya (64-bit)

EXPLICACIÓN DEL ALGORITMO

```
1 #include <iostream>
2 #include <ctime>     // Recursos para medir tiempos
3 #include <cstdlib>   // Para generación de números pseudoaleatorios
4
5 using namespace std;
6
7 int operacion(int *v, int n, int x, int inf, int sup) {
8   int med;
9   bool enc=false;
10  while ((inf<sup) && (!enc)) {
11    med = (inf+sup)/2;
12    if (v[med]==x)
13      enc = true;
14    else if (v[med] < x)
15      inf = med+1;
16    else
17      sup = med-1;
18  }
19  if (enc)
20    return med;
21  else
22    return -1;
23 }
```

```
*******************************************
* v = puntero simple que actúa como “array”.  *
* n = tamaño de v.                            *
* x = elemento buscado.                       *
* inf = posición mínima de búsqueda.          *
* sup = posición máxima de búsqueda.          *
* Luego [inf, sup] es el intervalo de búsqueda *
*******************************************
```

La función operacion(int *v, int n, int x, int inf, int sup) utiliza un algoritmo de búsqueda de *x*, el cual recorre el vector pasado como parámetro y cesa la búsqueda si se encontrase dicho elemento, devolviendo la posición donde se localiza o -1 en caso contrario. Para ello se crea la variable *med* de enteros y *enc* de valores booleanos, iniciándola a "*false*" para que entre en el *while*, el cual está condicionado a que *inf* sea menor estricto que *sup*, para no recorrer el vector más de una vez, y a que no se haya encontrado el elemento *x* en dicho vector (*!enc*).
Para recorrer el vector se comprueba que en la posición situada a igual distancia de *inf* que de *sup* (*med*) se encuentre *x*, de ser así nos saldríamos del bucle y la función devolvería la posición en la que se localiza dicho elemento. Si esto no ocurriese, se cambia el valor de *inf* por *med+1*, si el elemento que se halla en la posición *med* del vector *v* es menor que *x*, y *sup* por *med-1*, si es mayor.
En el caso de que se recorra todo el vector *v* y no se encuentre el elemento buscado se devuelve el valor -1.

EFICIENCIA TEÓRICA

Línea 9: 1 Operaciones elementales (OE) (asignación)

Línea 10: 3 OE (comparación inf<sup, comprobación !enc, operación &&)

Línea 11: 3 OE (suma, división, asignación)

Línea 12: 2 OE (acceso al elemento v[med], comparacion v[med]==x)

Línea 13: 1 OE (asignación)

Línea 14: 2 OE (acceso al elemento v[med], comparacion v[med]<x)

Línea 15: 2 OE (incremento, asignación)

Línea 18: 2 OE (disminución, asignación)

Línea 19: 1 OE (comprobación enc)

Línea 20: 1 OE (devolución)

Línea 22: 1 OE (devolución)

$$T(n)=1+\sum_{i=0}^{n-1}(3+3+2+max(1,2+max(2,2)))+1+max(1,1)=1+\sum_{i=0}^{n-1}12+2=3+12n$$

EFICIENCIA EMPÍRICA

Compilamos con *g++ ejercicio_desc.cpp -o ejercicio_desc* y mostramos la gráfica de los datos de *tiempo_desc.dat*, hallados con *ejecuciones_desc.bash* con gnuplot.

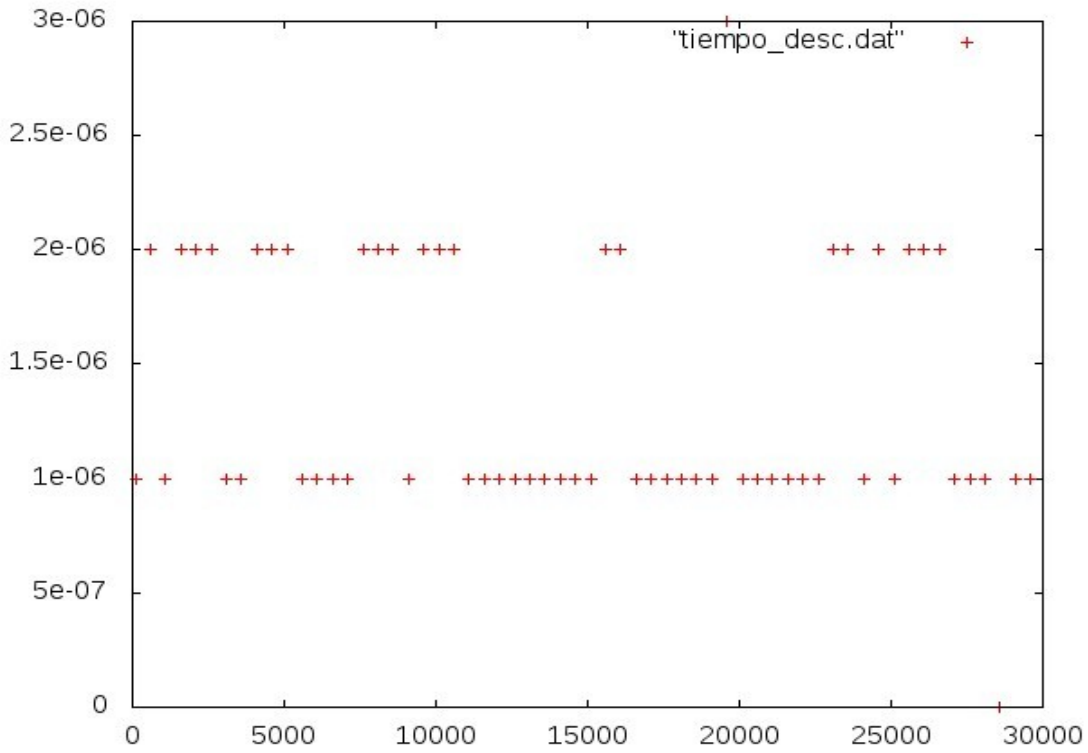
```
#ejecuciones_desc.bash

#!/bin/bash
```

```
inicio=100
fin=30000
incremento=500
ejecutable=ejercicio_desc
salida=tiempo_desc.dat

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

Siendo la gráfica la siguiente:



Los recursos de ctime no tienen tanta precisión como fuese deseable para estas tareas. En estas situaciones podemos ejecuta el mismo algoritmo muchas veces y dividir el tiempo total de ejecución por el numero de ejecuciones.