ESTRUCTURA DE DATOS

Práctica 1. Eficiencia de algoritmos

Doble Grado de Informática y Matemáticas

Víctor Castro Serrano
Maximino Suárez van Gelderen

28 de septiembre de 2017

Condiciones de ejecución.

Dado que en los siguientes ejercicios hablaremos de la eficiencia de distintos programas, conviene detallar las condiciones en las que se han llevado a cabo las pruebas.

Hardware: Asus GL552VW, Intel Core i5-6300HQ CPU @ 2.30GHz 4 cores, Intel HD Graphics 530 (Skylake GT2), 12GB RAM.
Sistema Operativo: Ubuntu 16.04.3 LTS 64-bit.
Compilador: g++
Opciones de compilación: -g -o

Ejercicio 3.

a) En este primer apartado, describiremos el funcionamiento del algoritmo proporcionado en el archivo *ejercicio_desc.cpp*.

Se trata de un algoritmo de **búsqueda binaria**, donde dado un vector v de n elementos enteros, busca el entero x en él. Para ello, se pasan como parámetros el inicio *(inf)* y el final (*sup*) de dicho vector. La función devuelve la posición donde se ha encontrado el elemento, o -1 si no estaba en el vector.

Para el correcto funcionamiento del algoritmo, es imprescindible que el vector esté **ordenado**, pues el procedimiento es el siguiente:

(*i*) Se establece la posición $med = (inf + sup)/2$.

(*ii*) Se comprueba si el elemento x está en la posición med.

(*iii*) Si está, hemos acabado. Si no está, se comprueba si el elemento $v[med]$ es mayor o menor que x.

- Si $v[med] < x$, actualizamos $inf = med + 1$.
- Si $v[med] > x$, actualizamos $sup = med - 1$.

(*iv*) Repetimos el proceso, hasta encontrar el elemento, o concluir que no está en el vector.

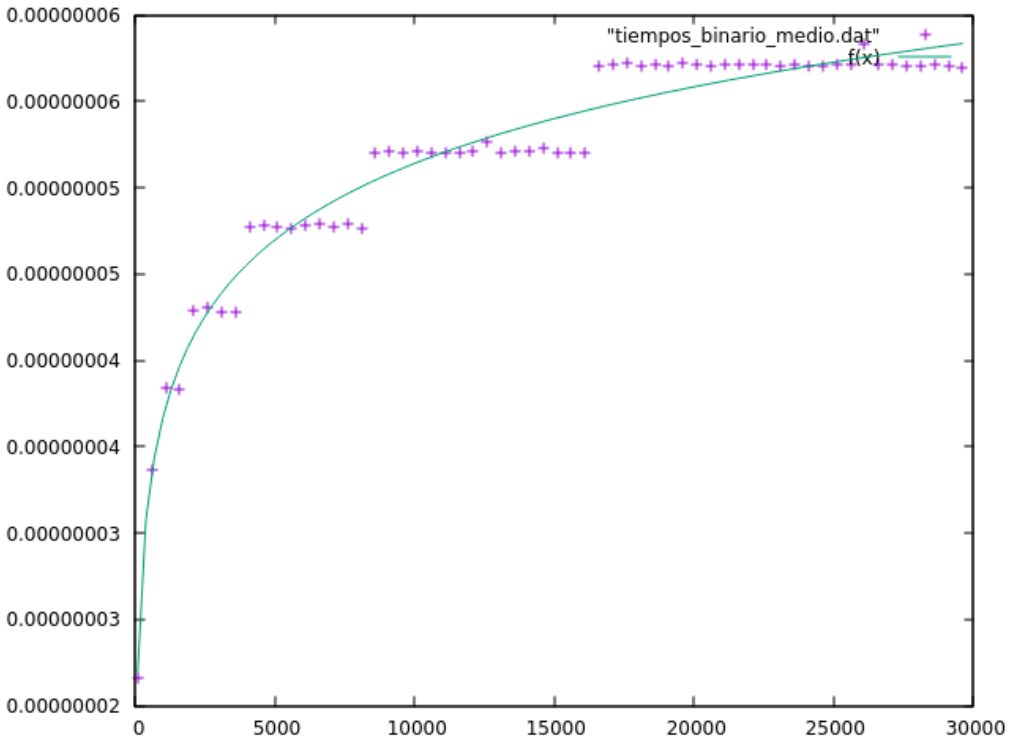
En resumen, el procedimiento se basa en dividir el vector original por la mitad, y si no está ahí el elemento buscado, nos quedamos únicamente con el sub-vector donde se puede encontrar

dicho elemento (recordemos que el vector está ordenado). Repitiendo el proceso, acabaremos encontrando el elemento, o descubriendo que no está en el vector, cuando ya no se puedan hacer más divisiones.

b) Para el cálculo de la eficiencia teórica de la *búsqueda binaria*, nos basamos en el hecho de que se divide sucesivamente en dos un vector de tamaño n. En el caso peor, el proceso continuará hasta que no se puedan hacer más divisiones del vector.

Como el resto de operaciones son elementales ($O(1)$), concluímos que la eficiencia del algoritmo es logarítmica, es decir, $T(n) \in O(\log(n))$.

c) Algunas ejecuciones tardan 0 segundos, suponemos que es porque el elemento se encontraba en el medio del vector. Para solucionar el problema, ejecutamos el mismo algoritmo muchas veces (en nuestro caso, 10000000) y dividimos el tiempo total de ejecución entre el numero de ejecuciones. Los datos se ven reflejados en *tiempos_binario_medio.dat* cuya grafica es la siguiente:



Ajustando la nube de puntos a la función exponencial $f(x) = 6{,}40319 \cdot 10^{-09} log(x) - 2{,}57655 \cdot 10^{-09}$, se puede observar claramente que la eficiencia empírica describe una gráfica de forma logarítimica, coincidiendo así con nuestros resultados teóricos.