Practica 1
Analisis de eficiencia

ADRIAN ORDUÑA DÍAZ, RAFAEL LEYVA RUIZ
Universidad de Granada
16 de octubre de 2016

1. Introduccion

En esta primera práctica se ha analizado la eficiencia de diversos algoritmos tanto empíricamente como teoricamente, ademas se ha procedido a comprobar como se comportan los algoritmos en diversos casos, tanto extremos (mejor y peor caso) comop evaluando como afecta el entorno, el proceso de comilacion, etc. a las implementaciones y eficiencia de estos en el mundo real. Los algoritmos evaluados han sido los siguientes:

- Busqueda lineal.
- Ordenacion lineal o algoritmo burbuja.
- Busqueda binaria.
- Multiplicacion de matrices.
- Ordenacion por mezcla o mergesort.

Salvo que en algun apartado de este texto se indique lo contrario todos los algoritmos descritos a continuación se han compilado con el compilador g++ para maquinas de 64 bits, y ejecutado en la maquina descrita a continuación:

Lenovo ThinkPad T440p.

```
CPU:       Dual core Intel Core i5-4210M (-HT-MCP-) cache: 3072 KB
clock speeds: max: 3200 MHz 1: 2647 MHz 2: 2664
MHz 3: 2589 MHz 4: 2608 MHz

Memoria:

MemTotal:        7864344 kB
```

```
MemFree:          438576 kB
MemAvailable:    1471416 kB
Buffers:          242820 kB
Cached:          1421444 kB
SwapCached:            0 kB
```

Nota: Los ejercicios descritos en este documento no mantienen el mismo orden que en la presentación de la práctica, ya que cuando se comience a hablar de un algoritmo se detallaran todos los analisis y pruebas realizadas, no obstante antes de cada analisis se detallara el ejercicio en concreto del que se trata.

2. Busqueda lineal

El primer algoritmo analizado fue el algoritmo básico de busqueda, el primero en el que pensaría cualquiera cuando se le plantea el problema, que no es mas que recorrer el vector hasta encontrar el elemento buscado. De esta descripción se aprecia claramente que en su peor caso es lineal, ya que habria que recorrer todo el vector, si el elemento buscado aparece en la ultima posición.

2.1. Analisis de eficiencia teorico

El desarrollo matemático de el analisis de la eficiencia teoríca del codigo del algoritmo que se muestra arriba es el siguiente:

Tras hacer pruebas con distintos tamaños de entradas se ha obtenido la siguiente grafica de tiempos, en la cual se enfrenta el tamaño del vector en el que se ha buscado aplicando el algoritmo (eje x) y el tiempo que ha tardado en completarse el algoritmo para cada tamaño.

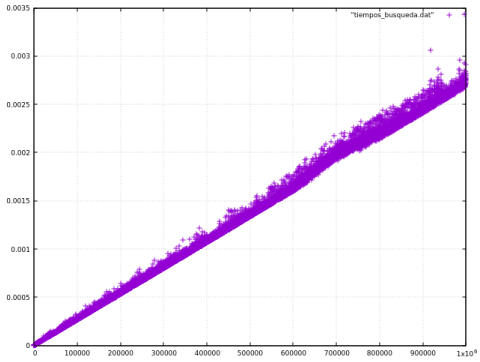

Figura 1: Datos de la salida del algoritmo de busqueda lineal

Como se aprecia el crecimiento de los tiempos de ejecucion del algoritmo es lineal conforme crece el tamaño de las entradas del problema. Tras obtener los datos anteriores, se uso la funcion fit de gnuplot para ajustar dicha salida con una funcion, para ver como se corresponde el tiempo teórico y el empirico, para ello se ajusto a la funcion $f(x) = a * x$, de tal modo que a nos indica la constante oculta del algoritmo. La grafica resultante es la siguiente:

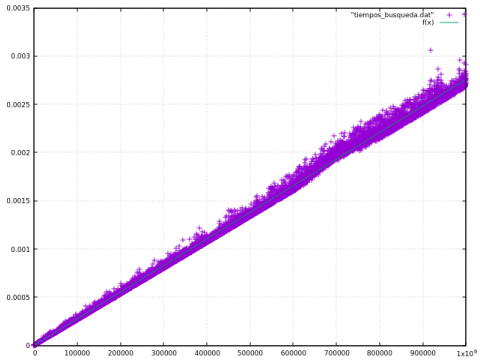

Figura 2: Salida del algoritmo de busqueda lineal ajustada a $f(x)$

3. Bubble sort

El siguiente algoritmo que se ha anañizado ha sido la ordenacion de la burbuja o bubble sort, la cual como demostraremos a continuacion es de orden $O(n^2)$, y se han realizado pruebas con distintas implementaciones asi como en distintos escenarios, como son mejor y peor caso.

3.1. Eficiencia teorica

$$T(n) = \sum_{i=0}^{n-1}(\sum_{j=0}^{n-i-1} (4 + max(3 + 4 + 3, 0))) = 14n^2 \quad (1)$$

Por lo que sabemos que el algoritmo tiene una $O(n^2)$.

3.2. Eficiencia empirica

Para demostrar que en efecto el tiempo de ejecucion de nuestro algoritmo evoluciona conforme nos ha indicado la eficiencia teorica, se hicieron varias ejecuciones con dicho algoritmo para diferentes tamaños de entradas, obteniendo la siguiente grafica:

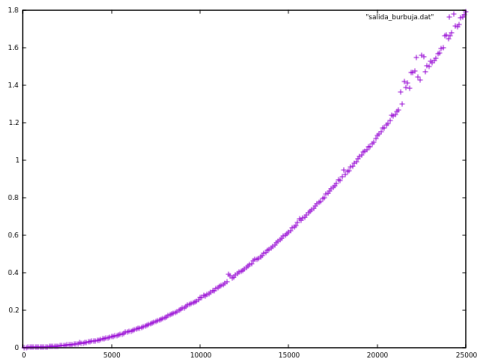

Figura 3: Grafica con los tiempos de ejecucion del algoritmo bubble sort

3.3. Ajuste de la grafica con $f(x) = x^2$

Si procedemos a comprobar como se ajusta la eficiencia teorica con la grafica obtenida en el apartado anterior veremos como efectivamente se corresponde a la perfección con una gráfica de una funcion polinomica de grado 2, lo cual nos demuestra lo inficiente de este algoritmo en cuanto el tamaño de las entradas crece. La grafica es la que se muestra a continuación:

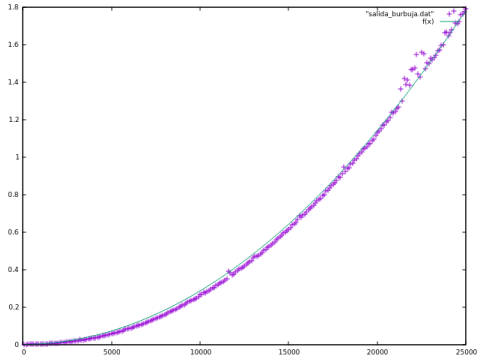

Figura 4: Ajuste de la salida de bubble sort con $f(x) = x^2$

3.4. Influencia del proceso de compilacion en la eficiencia empirica de un algoritmo

Como hemos comprobado en el anterior apartado, en efecto los tiempos de ejecucion del algoritmo en cuestion crecen en la medida que predijimos en el calculo teórico de la eficiencia, pero se nos plantea la duda de en que medida el proceso de compilación de nuestro programa, con lo cual se procedio a compilar el mismo programa con el que se habian obtenido los datos de las graficas anteriores, pero ahora se compilo usando el flag

-O3 aplicando asi el compilador un mayor grado de optimización a las instrucciones maquina generadas. Tras enfrentar la grafica del programa optimizado a la grafica anterior se obtuvo los siguientes resultados que a continuación analizaremos:

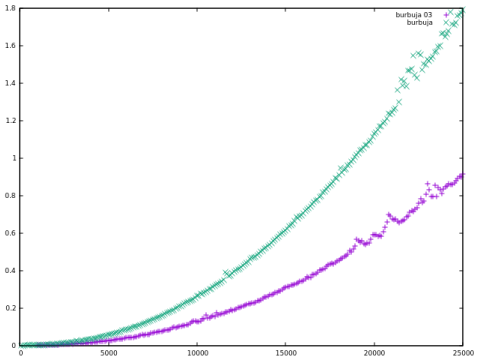

Figura 5: Algoritmo sin optimizavion vs algoritmo con optimizacion

Si nos fijamos en la anterior grafica se ve como el proceso de compilacion si que afecta y mucho a las constantes ocultas, que afectan a la eficiencia de nuestro algoritmo, ya que dos implementaciones, la primera de ellas de orden $2 * n^2$ y la segunda de orden $100 * n^2$ son de orden $O(n^2)$, pero como el lector adivinará la primera es mucho mejor ya que su factor multiplicativo es mucho menor, asi observamos que la segunda version en el maximo tamaño de entradas es 1 segundo mas rapida (en otros PC's estos tiempor podrian variar), y esta tendencia se mantendría hasta el infinito.

3.5. Mejor y Peor Caso

Cuando ejecutamos nuestros programas, estos trabajan sobre datos que cabe esperar que estén desordenados, pero se puede dar el caso en el que estos datos tengan una pecularidad que afecte al tiempo de ejecución.

3.5.1. Mejor caso

En este algoritmo, el mejor caso es que el vector de datos esté ordenado, ya que que no tiene que intercambiar ningún elemento, por lo que el tiempo de cómputo es menor.

En la siguiente gráfica, tras hacer pruebas con distintos tamaños de entrada, se ve el tiempo (eje Y) en relación a la entrada de datos (eje X).

Donde se aprecia que con independencia del tamaño del vector que haya que ordenar el tiempo es practicamente constante, ya que se podria ajustar

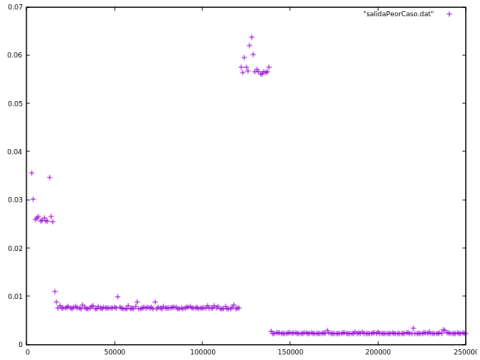

Figura 6: Datos de la salida del algoritmo de ordenación por burbuja

perfectamente a una recta, si obviamos las variaciones debidas a procesos del sistema operativo y otros aspectos que siempre afectan a estos analisis.

3.5.2. Peor caso

Este caso se da si el vector de datos de entrada está ordenado inversamente, ya que en cada iteracción habrá que hacer un intercambio, por lo que el tiempo de cómputo es mayor.

Observamos en la siguiente gráfica la compración entre el tiempo (eje Y) respecto del tamaño de datos de la entrada (eje X) del mejor y peor caso.

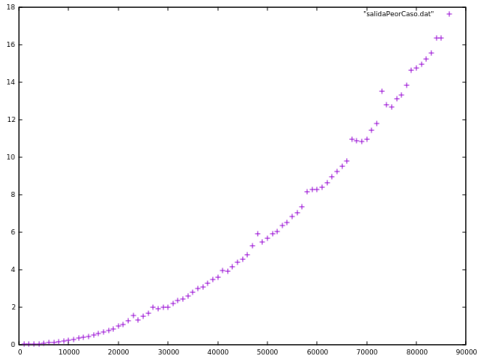

Figura 7: Peor caso en el algoritmo bubble sort

A continuacion se muestran las tres graficas enfrentadas, caso promedio, peor y mejor caso:

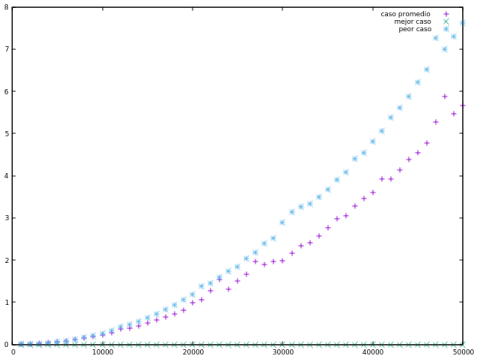

Figura 8: Casos enfrentados

4. Busqueda binaria

En otro ejercicio de los propuestos hemos tenido que abordar el problema de l afalta de precision a la hora de tomar los tiempos de ejecución de un algoritmo, para ello se nos presento una version del algoritmo de busqueda de orden $O(log_2(n))$ conocido como busqueda binaria. A continuacion demostraremos que en efecto el código proporcionado para las pruebas se corresponde con la citada eficiencia.

4.1. Calculo de la eficiencia teorica

4.2. Analisis de eficiencia empírico

En primer lugar se analizo el tiempo de ejecucion de el algoritmo para diferentes tamaños de entradas como se había hecho hasta el momento, es decir simplemente tomando los tiempos de reloj antes y despues de cada ejecucion, pero nos encontramos con que el resultado de la diferencia de estos tiempos (el tiempo que el algoritmo había empleado) era siempre 0. Esto era producido a causa de que clock, la funcion para tomar los tiempos que estabamos empleando, no disponia de la precisión necesaria para medir el tiempo que tardaba un algoritmo tan rapido como este. Para solventar este escollo se añadio una modificación al código original de modo que en cada ejecucion del programa principal la ejecucion del algoritmo se hacia 10000 veces, y a continuacion se dividia el tiempo que habían tomado las 10000 ejecuciones entre 10000 para asi obtener el tiempo de cada ejecucion, ademas el tamaño de las entradas fue mucho mayor que el empleado por ejemplo en las pruebas con el algoritmo de busqueda lineal. Asi para pruebas que comprenden tamaños de entradas de entre 1000000 y 300000000 de elementos, se ha obtenido la siguiente gráfica:

Si analizamos los siguientes datos se observa como al principio el tiempo

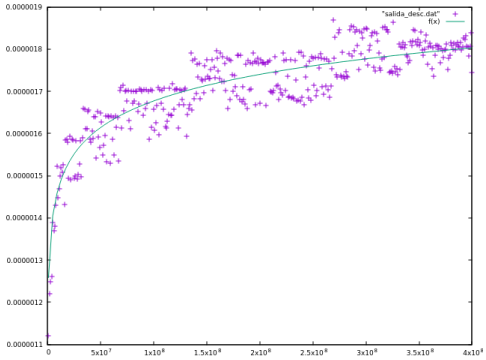

Figura 9: Grafica de la salida de ejecuciones del algoritmo busqueda binaria

de ejecucion crece mucho con cada incremento del tamaño de las entradas, pero luego apenas crece para aumentos de tamaño de 1000000 de datos, lo cual es un aumento considerable que no obstante apenas incrementa el tiempo de ejecución.

Además se ve como el tiempo de busqueda para un vector de tamaño 400000000 de datos es de menos de 0.0000019 segundos, lo cual denota que el algoritmo es muy rápido.

5. Multiplicacion matrices

Para este ejercicio se pedir diseñar un algoritmo de multiplicacion de matrices cuadradas y evaluar su eficiencia. El algoritmo diseñado es el siguiente:

5.1. Analisis de eficiencia teoríco

El analisis de eficiencia muestra que al ser tres bubles anidados el orden es n^3 y su desarrollo es el siguiente:

$$T(n) = \sum_{i=0}^{n}(\sum_{j=0}^{n}(\sum_{k=0}^{n}(1))) = n * \sum_{j=0}^{n}(\sum_{k=0}^{n}(1)) = n^2 * \sum_{k=0}^{n}(1) = n^3 \qquad (2)$$

5.2. Analisis empírico de eficiencie

El analisis empírico en nuestra maquina arrojo los siguientes resultados, los cuales verifican el orden cúbico del algoritmo diseñado:

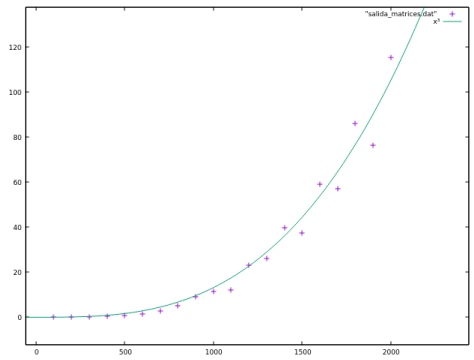

Figura 10: Grafica de las ejecuciones del algoritmo de multiplicación de matrices ajustada con la funcion $o(n^3)$

Ademas se comprobo tambien como afecta la optimización a dicho algoritmo, lo cual se muestra en la siguiente gráfica.

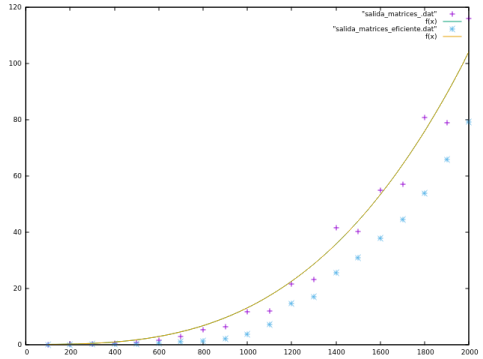

Figura 11: Diferencia de tiempos para el algoritmo sin optimizar y el algoritmo optimizado

Donde se aprecia que para el tamaño de matriz máximo probado la diferencia es de 40 segundo, lo que en diversos casos puede representar un tiempo crítico, tambien es importante terner en cuanta que para mayores tamaños estas diferencias seguirian aumentando en funcion de n^3 por lo que seria conveniente diseñar algoritmos con un menor tiempo de ejecucion(se verá en el segundo cuatrimestre).

6. Analisis de la ordenacion por mezcla o mergesort

En el ultimo ejercicio se ha analizado la eficiencia de uno de los algoritmos rapidos de ordenacion, en este caso el mergesort, que tiene una

eficiencia teoríca de $O(n * log(n))$, a continuacion se muestra como en efecto la eficiencia del algoritmo implementado se ajusta a la funcion anterior:

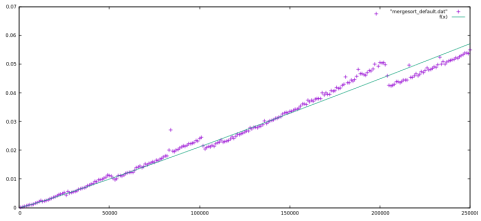

Figura 12: Grafica de la eficiencia del algoritmo mergesort

6.1. Importancia del parametro UMBRAL_MS

A continuacion se muestra como afecta el parametro UMBRAL_MS a la velocidad del algoritmo, ya que este indica el tamaño del caso base a partir del cual se realiza la ordenacion lineal y luego se combina la solucion.

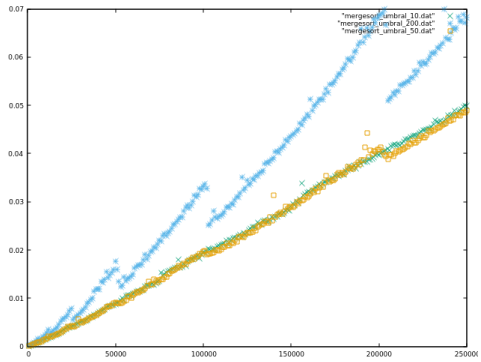

Figura 13: Pruebas con diferentes umbrales del algoritmo mergesort