Memoria de algorítmica

Rubén Morales Pérez Francisco Javier Morales Piqueras
Bruno Santindrian Manzanedo Ignacio de Loyola Barragan Lozano
Francisco Leopoldo Gallego Salido

16 de marzo de 2016

Índice

1. Explicación del método utilizado

Para la obtención de los datos deseados hemos realizado un script de bash que genera las tablas de datos y las gráficas con su correspondiente ajuste.

```
\input{../Script/script.sh}
```

Para la obtención de las gráficas de forma directa utilizamos script de gnuplot que tienen la forma siguiente, en este caso adjuntamos "burbuja.gp".

```
set terminal jpeg
set output "fichero.jpeg"

set title "Eficiencia burbuja"
set xlabel "Tamano del vector"
set ylabel "Tiempo (s)"
set fit quiet
f(x) = a*x*x+b*x+c
fit f(x) "datos.dat" via a, b, c
plot "datos.dat", f(x)
```

Los diferentes ajustes se han conseguido así:

```
f(x) = a*x*x*x+b*x*x+c*x+d
g(x) = a*x*x+b*x+c
h(x) = a*x*(log(x)/log(2))
i(x) = a*(((1+sqrt(5))/2)**x)
```

2. Cálculo de la eficiencia empírica

2.1. Tabla con los algorítmos cuadráticos

2.2. Tabla con los algorítmos cúbicos

2.3. Tabla con los algorítmos nlog(n)

2.4. Tabla con el algorítmo de Fibonacci

Fibonaccii	
1	2
t11	t12
t21	t22

Cuadro 1: Tabla de tiempos

2.5. Tabla con el algorítmo de Hanoi

2.6. Tabla con los algoritmos de ordenación

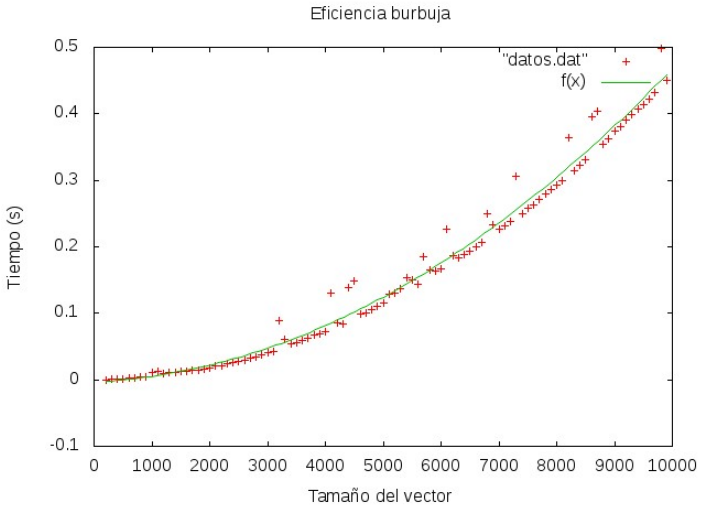
3. Gráficas

3.1. Ordenaciíon

En este apartado compararemos 6 algoritmos diferentes de ordenación dentro de un vector. Cada algoritmo lleva su ajuste correspondiente.

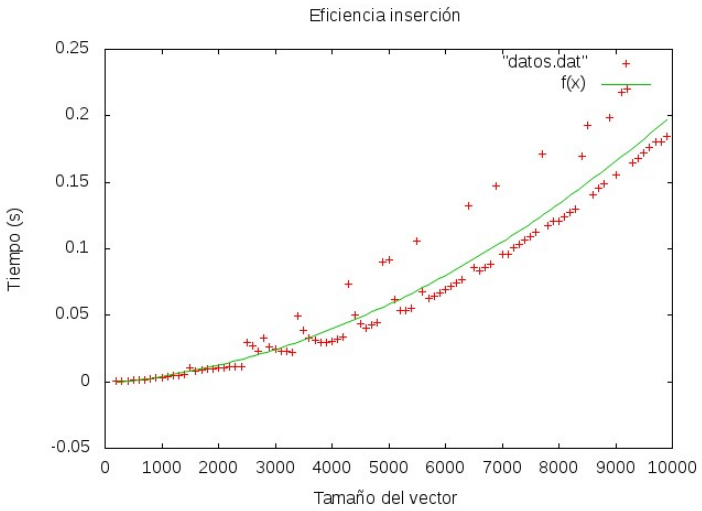
3.1.1. Burbuja

$$f(x) = a \cdot x^2 + b \cdot x + c \implies \begin{cases} a = 4{,}31433 \cdot 10^{-9} +/- 2{,}378 \cdot 10^{-10} (5{,}511\,\%) \\ b = 3{,}94506 \cdot 10^{-6} +/- 2{,}476 \cdot 10^{-6} (62{,}75\,\%) \\ c = -0{,}00311235 +/- 0{,}005425 (174{,}3\,\%) \end{cases}$$



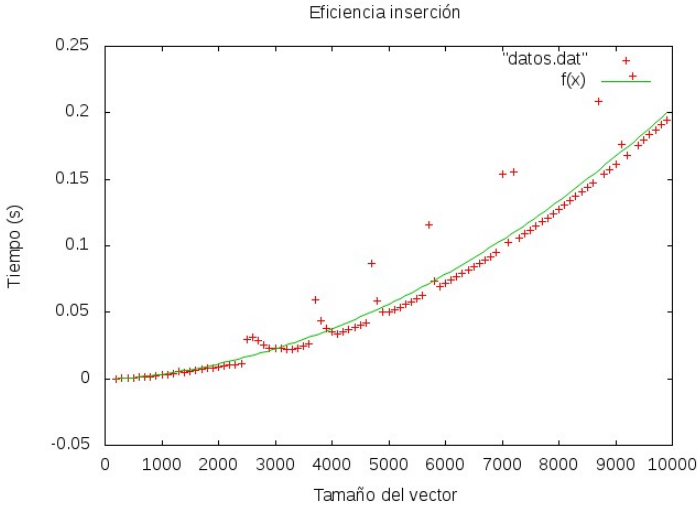
3.1.2. Inserción

$$f(x) = a \cdot x^2 + b \cdot x + c \implies \begin{cases} a = 2{,}36229 \cdot 10^{-9} +/- 2{,}503 \cdot 10^{-10} (10{,}6\,\%) \\ b = -2{,}27723 \cdot 10^{-6} +/- 2{,}606 \cdot 10^{-6} (114{,}5\,\%) \\ c = 0{,}00096037 +/- 0{,}005712 (594{,}8\,\%) \end{cases}$$



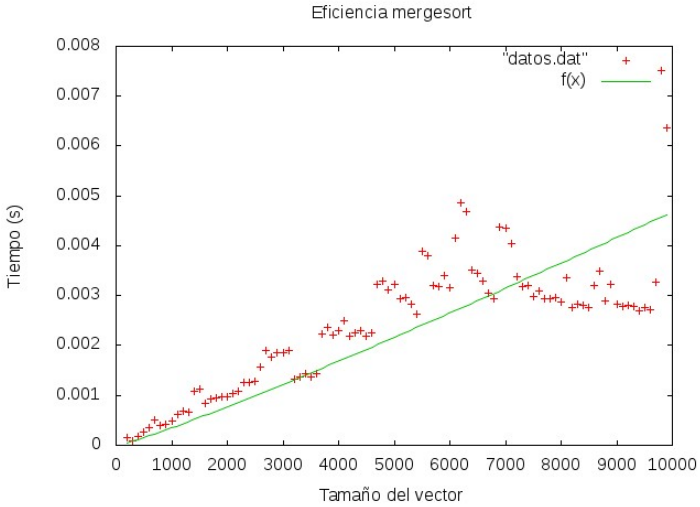
3.1.3. Selección

$f(x) = a \cdot x^2 + b \cdot x + c \implies a = 2{,}36327 \cdot 10^{-9} +/- 3{,}232 \cdot 10^{-11} (1{,}368\,\%)$



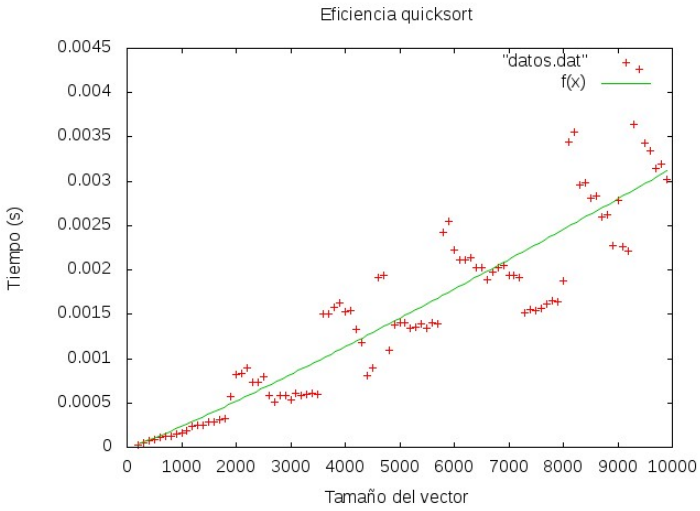
3.1.4. Mergesort

$f(x) = a \cdot x \cdot log_2(x) \implies a = 3{,}5231 \cdot 10^{-8} +/- 1{,}191 \cdot 10^{-9} (3{,}382\,\%)$



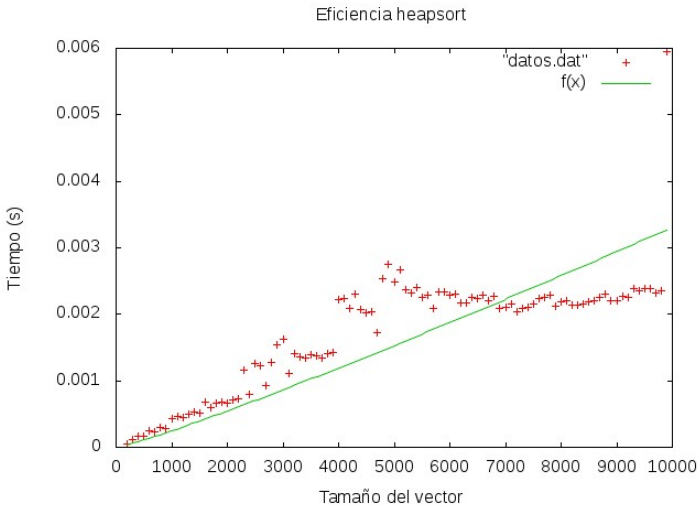
3.1.5. Quicksort

$f(x) = a \cdot x \cdot log_2(x) \implies a = 2{,}3704 \cdot 10^{-8} +/- 5{,}497 \cdot 10^{-10} (2{,}319\,\%)$



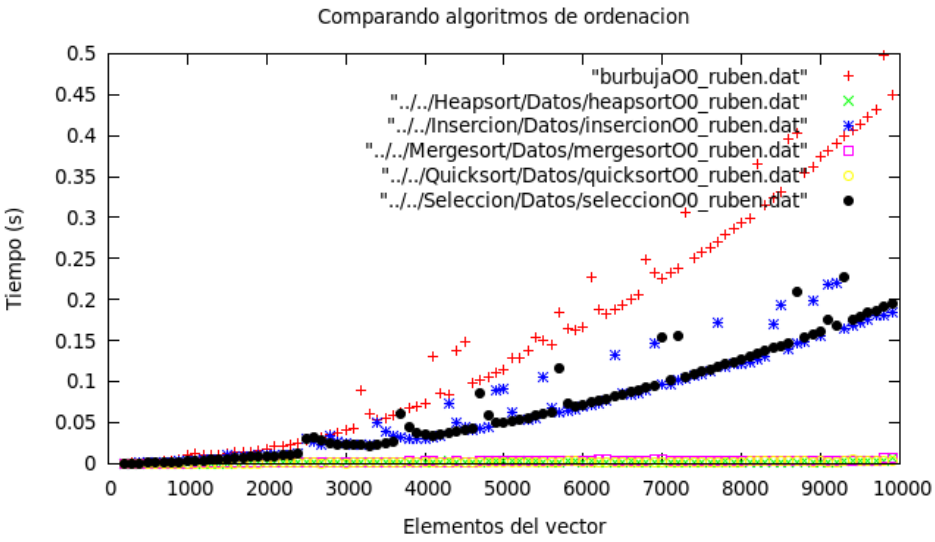
3.1.6. Heapsort

$f(x) = a \cdot x \cdot log_2(x) \implies a = 2{,}49016 \cdot 10^{-8} +/- 7{,}983 \cdot 10^{-10} (3{,}206\,\%)$



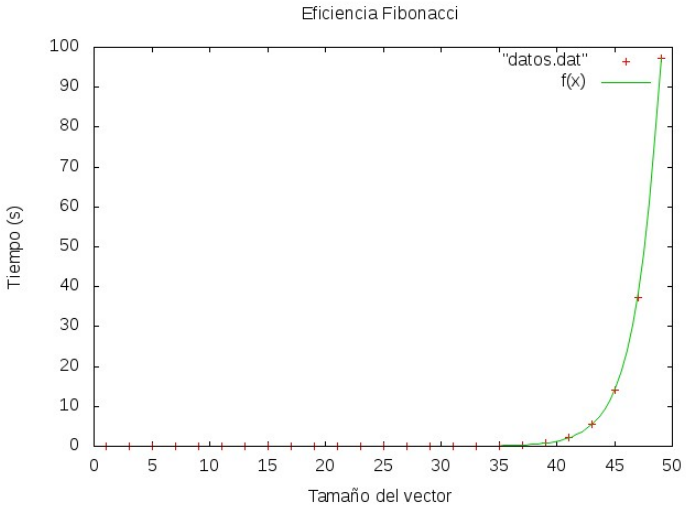
3.1.7. Comparativa algoritmos de ordenaciíon

En este apartado comprobamos empíricamente las diferencias de eficiencia entre diferentes algoritmos de ordenación de un vector. Se observa una diferencia notable entre los algoritmos $O(nlog_2(n))$ y los $O(n^2)$, casi no se aprecian los primeros.
También nos percatamos de la diferencia dentro de los mismos algoritmos con eficiencia $O(n^2)$, debido a la constante multiplicativa que los acompaña, inserción y selección son parecidos y burbuja tarda bastante más que los anteriores.



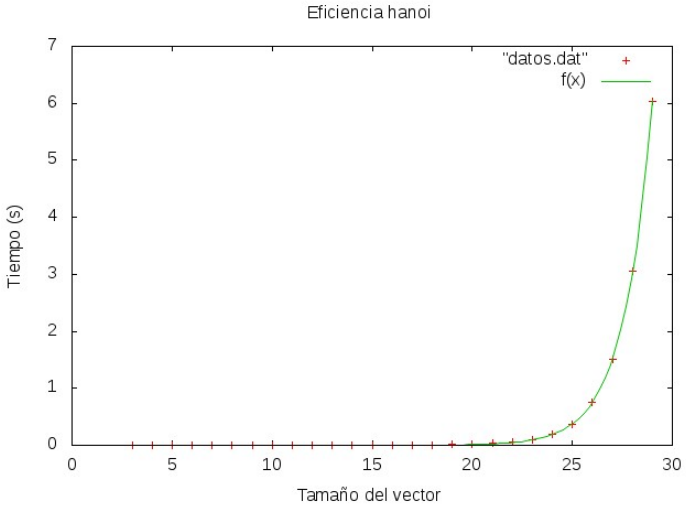
3.2. Fibonacci

$f(x) = a \cdot ((1+\sqrt{(}5))/2)^x \implies a = 5{,}59738 \cdot 10^{-9} +/- 2{,}093 \cdot 10^{-12} (0{,}0374\,\%)$



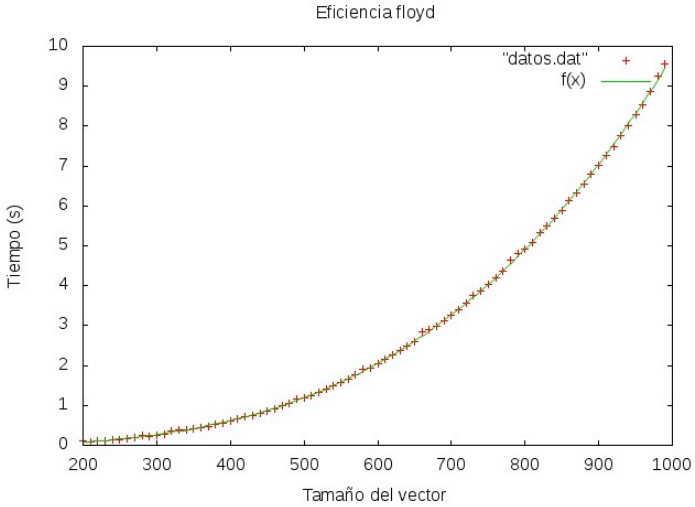
3.3. Hanoi

$f(x) = a \cdot (2^x) \implies a = 1{,}12636 \cdot 10^{-8} +/- 1{,}391 \cdot 10^{-11} (0{,}1235\,\%)$



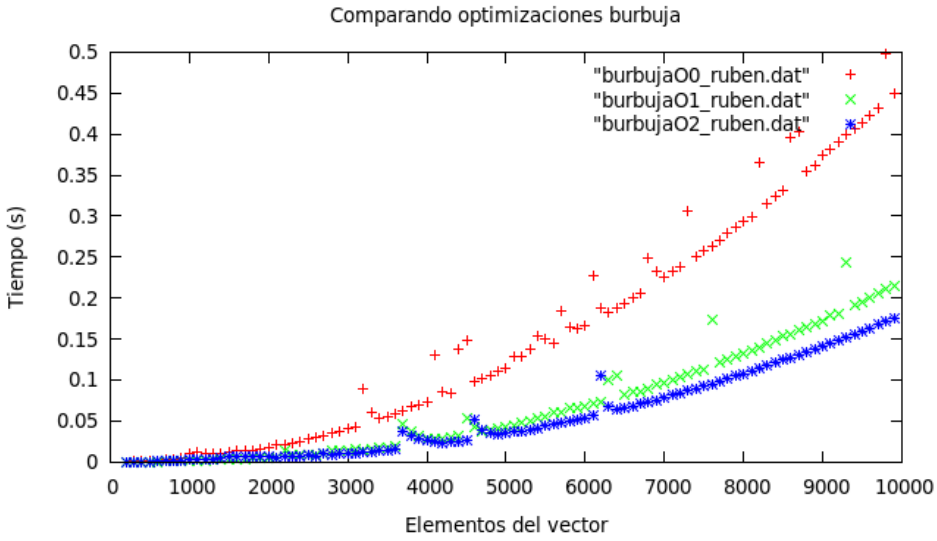
3.4. Floyd

$$f(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d \implies \begin{cases} a = 1{,}11725 \cdot 10^{-8} +/- 3{,}725 \cdot 10^{-10} (3{,}334\,\%) \\ b = -2{,}27723 \cdot 10^{-6} +/- 6{,}692 \cdot 10^{-7} (29{,}39\,\%) \\ c = 0{,}00096037 +/- 0{,}0003713 (38{,}66\,\%) \\ d = -0{,}115743 +/- 0{,}06234 (53{,}86\,\%) \end{cases}$$

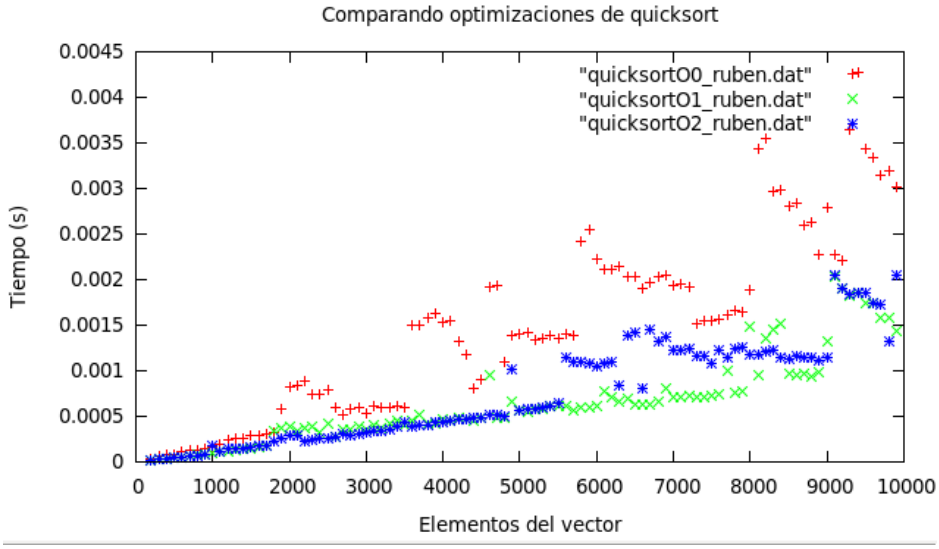


3.5. Optimización de algunos algoritmos

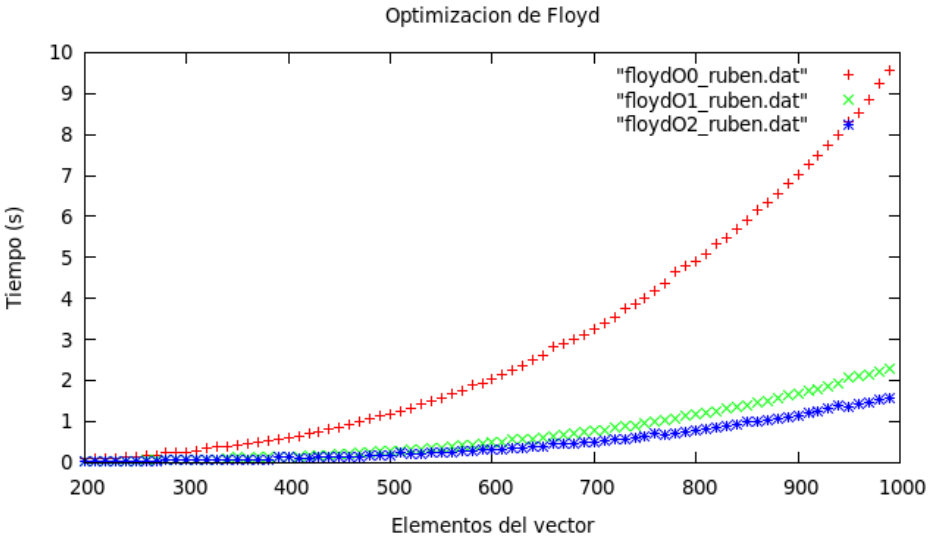
Como podemos comprobar, por mucho que optimicemos el algoritmo de burbuja no llega a igualarse al mejor algoritmo de ordenación (en término medio), quicksort. La optimización más agresiva sin riesgo de pérdida de información es -O2 y llega a ser 10 veces más lento que quicksort sin optimización (con 10.000 elementos).
Esto es una prueba gráfica de que hay que tener en cuenta la eficiencia de los algoritmos, ya que la mejora hardware no es suficiente en caso de que tengamos restricciones de tiempo.



Una observación curiosa es que el algoritmo Quicksort realmente es un algoritmo con eficiencia en el caso peor de $O(n^2)$, ya que si le pasamos un vector que está ordenado es cuadrático. Por otro lado Heapsort es un $O(nlog_2 n)$ puro, pero en término medio es peor que Quicksort, ya que los datos que se suelen pasar a estos algoritmos no están ordenados.



Después de comparar dichos algoritmos de ordenación optimizaremos el algoritmo floyd, tipo de algoritmo con programación dinámica para encontrar el camino mínimo en grafos ponderados.



4. Ordenador usado para la ejecución

HP Pavilion g series (Pavilion g6)
Sistema operativo: ubuntu 14.04 LTS
Memoria: 3.8 GiB (4Gb)
Procesador: Inter Core i3-2330M CPU @ 2.20GHz x 4
Gráficos: Intel Sandybridge Mobile
Tipo de SO: 64 bits
Disco: 487.9 GB