

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	2
2. Cálculo de la eficiencia empírica	6
2.1. Tabla con los algoritmos cuadráticos	6
2.2. Tabla con los algoritmos cúbicos	6
2.3. Tabla con los algoritmos $n\log(n)$	6
2.4. Tabla con el algoritmo de Fibonacci	6
2.5. Tabla con el algoritmo de Hanoi	6
2.6. Tabla con los algoritmos de ordenación	6
3. Gráficas	7
3.1. Ordenación	7
3.1.1. Burbuja	7
3.1.2. Inserción	7
3.1.3. Selección	8
3.1.4. Mergesort	8
3.1.5. Quicksort	9
3.1.6. Heapsort	9
3.1.7. Comparativa algoritmos de ordenación	9
3.2. Fibonacci	10
3.3. Hanoi	11
3.4. Floyd	11
3.5. Optimización de algunos algoritmos	12
4. Ordenador usado para la ejecución	13

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.

```
1  #!/bin/bash
2
3  if [ $# -ne 1 ]
4  then
5      echo "Uso: $0 <nombre>"
6      exit 1
7  fi
8
9  # HEAPSORT
10 g++ -std=c++11 ../src/heapsort.cpp
11 nelementos=200
12 echo "" > datos.dat
13 while [ $nelementos -lt 10000 ]; do
14     ./a.out $nelementos >> datos.dat
15     let nelementos=nelementos+100
16 done
17
18 gnuplot ./gnuplot/heapsort.gp # Salida: "fichero.jpeg"
19
20 mkdir ../Graficas/Heapsort 2> /dev/null
21 mkdir ../Graficas/Heapsort/Datos 2> /dev/null
22 mv fichero.jpeg ../Graficas/Heapsort/heapsort00_$1.jpeg
23 mv datos.dat ../Graficas/Heapsort/Datos/heapsort00_$1.dat
24 echo "Heapsort completado"
25
26
27 # MERGESORT
28 g++ -std=c++11 ../src/mergesort.cpp
29 nelementos=200
30 echo "" > datos.dat
31 while [ $nelementos -lt 10000 ]; do
32     ./a.out $nelementos >> datos.dat
33     let nelementos=nelementos+100
34 done
35
36 gnuplot ./gnuplot/mergesort.gp # Salida: "fichero.jpeg"
37
38 mkdir ../Graficas/Mergesort 2> /dev/null
39 mkdir ../Graficas/Mergesort/Datos 2> /dev/null
40 mv fichero.jpeg ../Graficas/Mergesort/mergesort00_$1.jpeg
41 mv datos.dat ../Graficas/Mergesort/Datos/mergesort00_$1.dat
42 echo "Mergesort completado"
43
44
45 # INSERCION
46 g++ -std=c++11 ../src/insercion.cpp
47 nelementos=200
48 echo "" > datos.dat
49 while [ $nelementos -lt 10000 ]; do
```

```

50     ./a.out $nelementos >> datos.dat
51     let nelementos=nelementos+100
52 done
53
54 gnuplot ./gnuplot/insercion.gp # Salida: "fichero.jpeg"
55
56 mkdir ../Graficas/Insercion 2> /dev/null
57 mkdir ../Graficas/Insercion/Datos 2> /dev/null
58 mv fichero.jpeg ../Graficas/Insercion/insercion00_$1.jpeg
59 mv datos.dat ../Graficas/Insercion/Datos/insercion00_$1.dat
60 echo "Insercion completado"
61
62
63 # SELECCION
64 g++ -std=c++11 ../src/seleccion.cpp
65 nelementos=200
66 echo "" > datos.dat
67 while [ $nelementos -lt 10000 ]; do
68     ./a.out $nelementos >> datos.dat
69     let nelementos=nelementos+100
70 done
71
72 gnuplot ./gnuplot/insercion.gp # Salida: "fichero.jpeg"
73
74 mkdir ../Graficas/Seleccion 2> /dev/null
75 mkdir ../Graficas/Seleccion/Datos 2> /dev/null
76 mv fichero.jpeg ../Graficas/Seleccion/seleccion00_$1.jpeg
77 mv datos.dat ../Graficas/Seleccion/Datos/seleccion00_$1.dat
78 echo "Seleccion completado"
79
80
81 # QUICKSORT
82 g++ -std=c++11 ../src/quicksort.cpp
83 nelementos=200
84 echo "" > datos.dat
85 while [ $nelementos -lt 10000 ]; do
86     ./a.out $nelementos >> datos.dat
87     let nelementos=nelementos+100
88 done
89
90 gnuplot ./gnuplot/quicksort.gp # Salida: "fichero.jpeg"
91
92 mkdir ../Graficas/Quicksort 2> /dev/null
93 mkdir ../Graficas/Quicksort/Datos 2> /dev/null
94 mv fichero.jpeg ../Graficas/Quicksort/quicksort00_$1.jpeg
95 mv datos.dat ../Graficas/Quicksort/Datos/quicksort00_$1.dat
96 echo "Quicksort completado"
97
98
99 # BURBUJA
100 g++ -std=c++11 ../src/burbuja.cpp
101 nelementos=200
102 echo "" > datos.dat
103 while [ $nelementos -lt 10000 ]; do

```

```

104     ./a.out $nelementos >> datos.dat
105     let nelementos=nelementos+100
106 done
107
108 gnuplot ./gnuplot/burbuja.gp # Salida: "fichero.jpeg"
109
110 mkdir ../Graficas/Burbuja 2> /dev/null
111 mkdir ../Graficas/Burbuja/Datos 2> /dev/null
112 mv fichero.jpeg ../Graficas/Burbuja/burbuja00_$1.jpeg
113 mv datos.dat ../Graficas/Burbuja/Datos/burbuja00_$1.dat
114 echo "Burbuja completado"
115
116
117 # FIBONACCI
118 g++ -std=c++11 ../src/fibonacci.cpp
119 nelementos=1
120 echo "" > datos.dat
121 while [ $nelementos -lt 50 ]; do
122     ./a.out $nelementos >> datos.dat
123     let nelementos=nelementos+2
124 done
125
126 gnuplot ./gnuplot/fibonacci.gp # Salida: "fichero.jpeg"
127
128 mkdir ../Graficas/Fibonacci 2> /dev/null
129 mkdir ../Graficas/Fibonacci/Datos 2> /dev/null
130 mv fichero.jpeg ../Graficas/Fibonacci/fibonacci00_$1.jpeg
131 mv datos.dat ../Graficas/Fibonacci/Datos/fibonacci00_$1.dat
132 echo "Fibonacci completado"
133
134
135 # HANOI
136 g++ -std=c++11 ../src/hanoi.cpp
137 nelementos=3
138 echo "" > datos.dat
139 while [ $nelementos -lt 30 ]; do
140     ./a.out $nelementos >> datos.dat
141     let nelementos=nelementos+1
142 done
143
144 gnuplot ./gnuplot/hanoi.gp # Salida: "fichero.jpeg"
145
146 mkdir ../Graficas/Hanoi 2> /dev/null
147 mkdir ../Graficas/Hanoi/Datos 2> /dev/null
148 mv fichero.jpeg ../Graficas/Hanoi/hanoi00_$1.jpeg
149 mv datos.dat ../Graficas/Hanoi/Datos/hanoi00_$1.dat
150 echo "Hanoi completado"
151
152
153 # FLOYD
154 g++ -std=c++11 ../src/floyd.cpp
155 nelementos=200
156 echo "" > datos.dat
157 while [ $nelementos -lt 1000 ]; do

```

```

158     ./a.out $nelementos >> datos.dat
159     let nelementos=nelementos+10
160 done
161
162 gnuplot ./gnuplot/floyd.gp # Salida: "fichero.jpeg"
163
164 mkdir ../Graficas/Floyd 2> /dev/null
165 mkdir ../Graficas/Floyd/Datos 2> /dev/null
166 mv fichero.jpeg ../Graficas/Floyd/floyd00_$1.jpeg
167 mv datos.dat ../Graficas/Floyd/Datos/floyd00_$1.dat
168 echo "Floyd completado"
169
170 rm a.out
171 rm fit.log

```

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

```

1  set terminal jpeg
2  set output "fichero.jpeg"
3
4  set title "Eficiencia burbuja"
5  set xlabel "Tamano del vector"
6  set ylabel "Tiempo (s)"
7  set fit quiet
8  f(x) = a*x*x+b*x+c
9  fit f(x) "datos.dat" via a, b, c
10 plot "datos.dat", f(x)

```

Los diferentes ajustes se han conseguido así:

```

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

```

2. Cálculo de la eficiencia empírica

2.1. Tabla con los algoritmos cuadráticos

2.2. Tabla con los algoritmos cúbicos

2.3. Tabla con los algoritmos $n\log(n)$

2.4. Tabla con el algoritmo de Fibonacci

Fibonacci	
1	2
t11	t12
t21	t22

Cuadro 1: Tabla de tiempos

2.5. Tabla con el algoritmo de Hanoi

2.6. Tabla con los algoritmos de ordenación

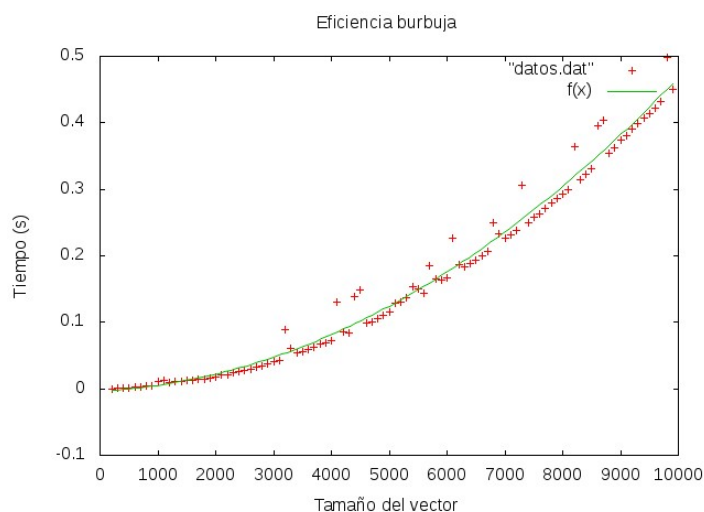
3. Gráficas

3.1. Ordenación

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

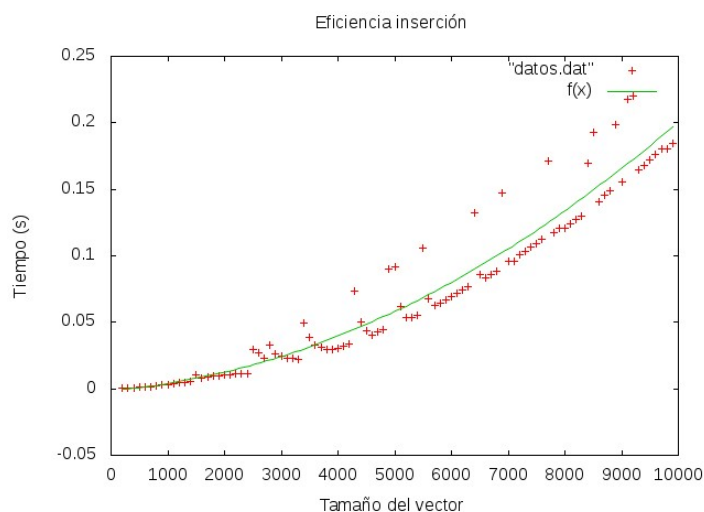
3.1.1. Burbuja

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



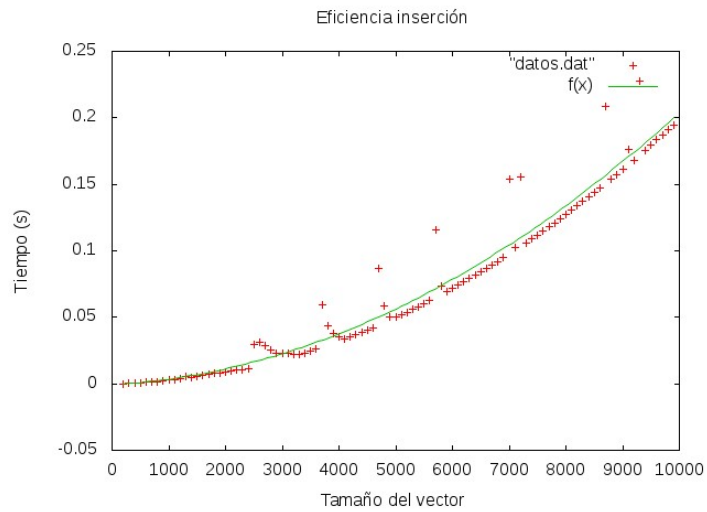
3.1.2. Inserción

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



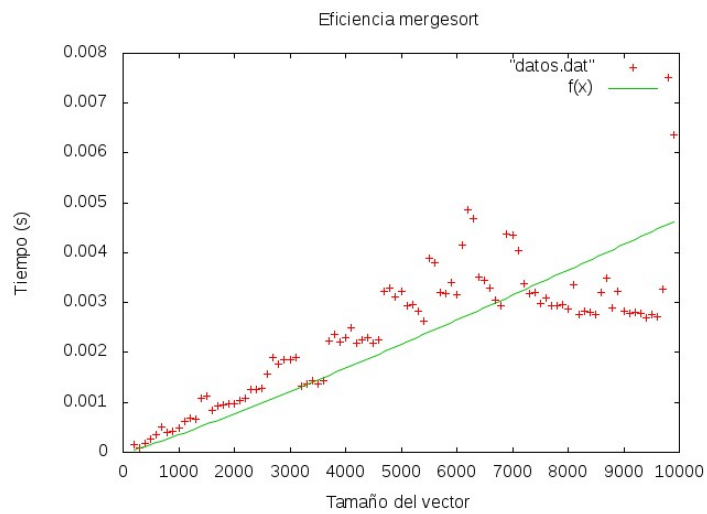
3.1.3. Selección

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



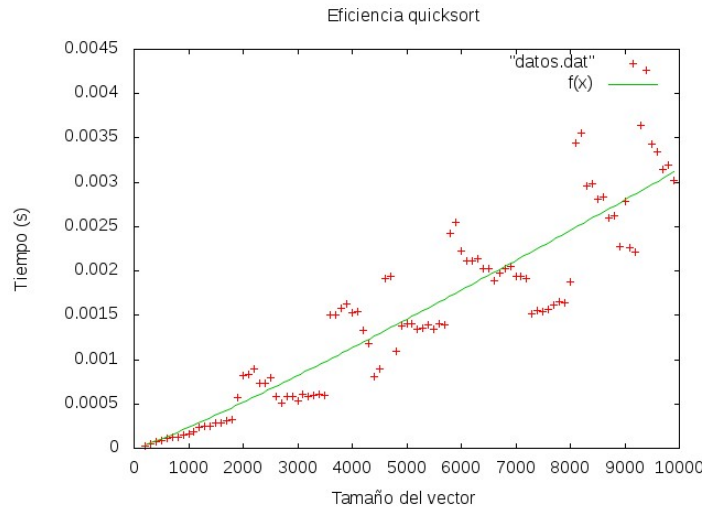
3.1.4. Mergesort

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



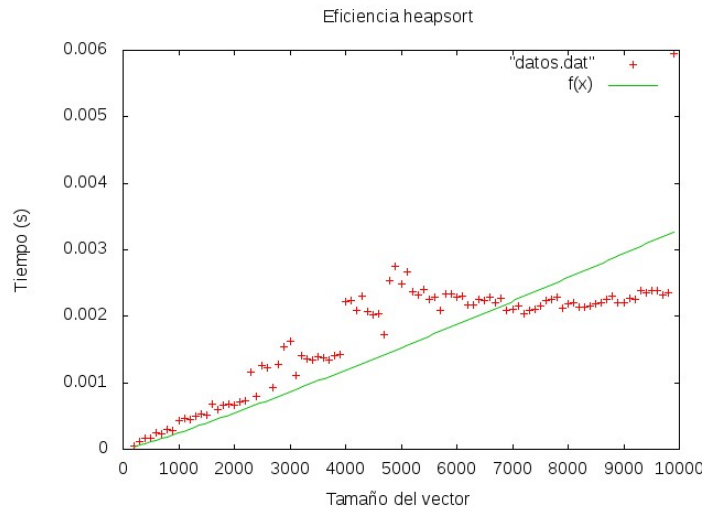
3.1.5. Quicksort

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



3.1.6. Heapsort

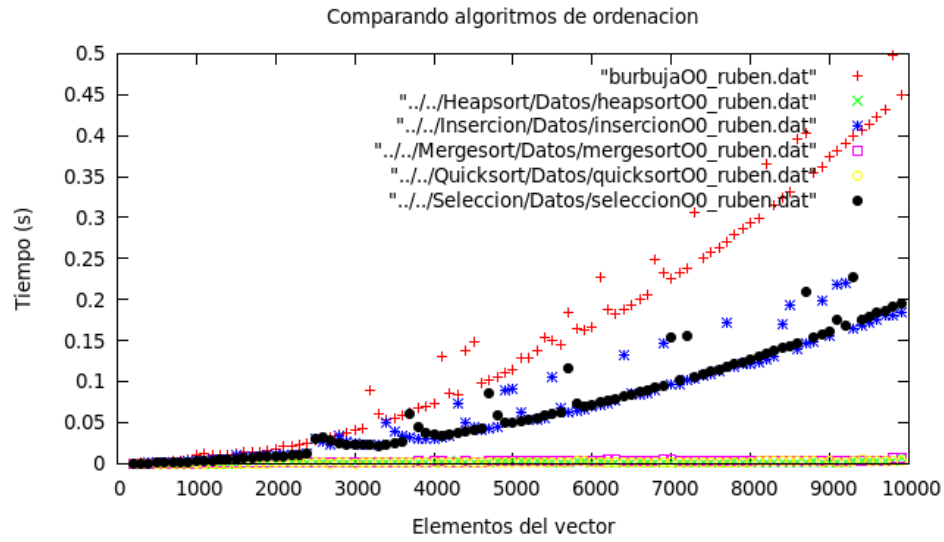
$$f(x) = a \cdot x \cdot \log_2(x) \implies a = 2,49016 \cdot 10^{-8} + / - 7,983 \cdot 10^{-10} (3,206 \%)$$



3.1.7. Comparativa algoritmos de ordenación

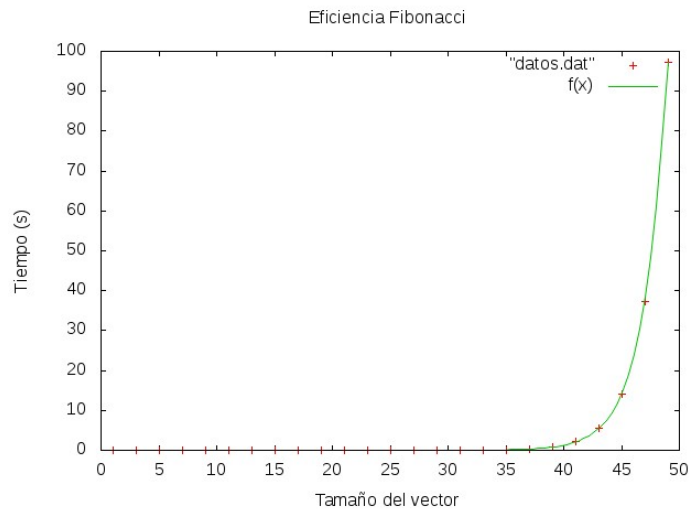
En este apartado comprobamos empíricamente las diferencias de eficiencia entre diferentes algoritmos de ordenación de un vector. Se observa una diferencia notable entre los algoritmos $O(n \log_2(n))$ y los $O(n^2)$, casi no se aprecian los primeros.

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.



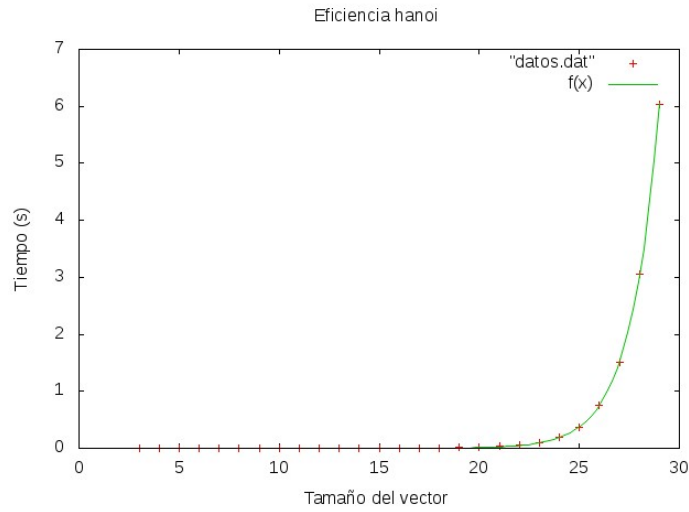
3.2. Fibonacci

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



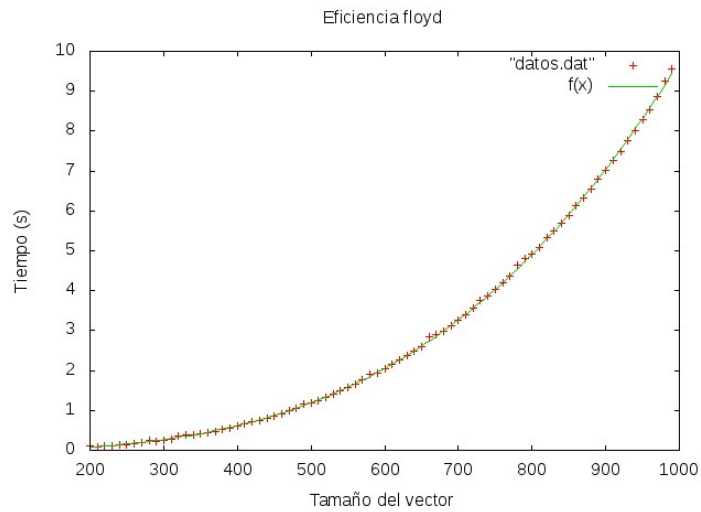
3.3. Hanoi

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



3.4. Floyd

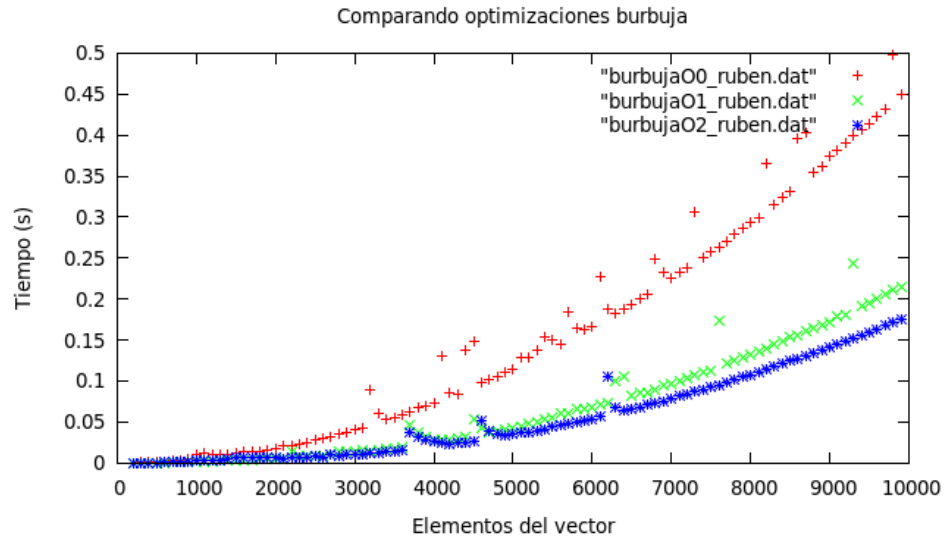
$$f(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d \Rightarrow \begin{cases} a = 1,11725 \cdot 10^{-8} + / - 3,725 \cdot 10^{-10} (3,334 \%) \\ b = -2,27723 \cdot 10^{-6} + / - 6,692 \cdot 10^{-7} (29,39 \%) \\ c = 0,00096037 + / - 0,0003713 (38,66 \%) \\ d = -0,115743 + / - 0,06234 (53,86 \%) \end{cases}$$



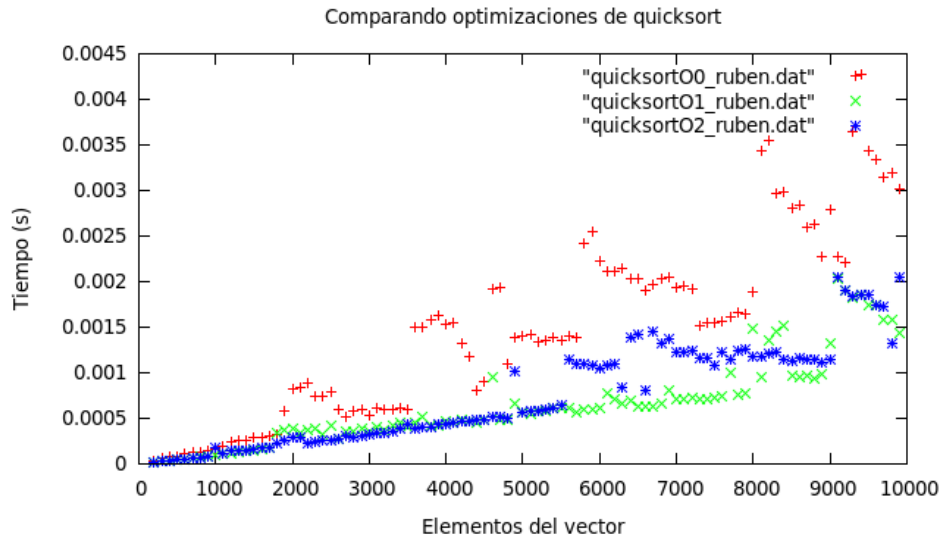
3.5. Optimización de algunos algoritmos

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).

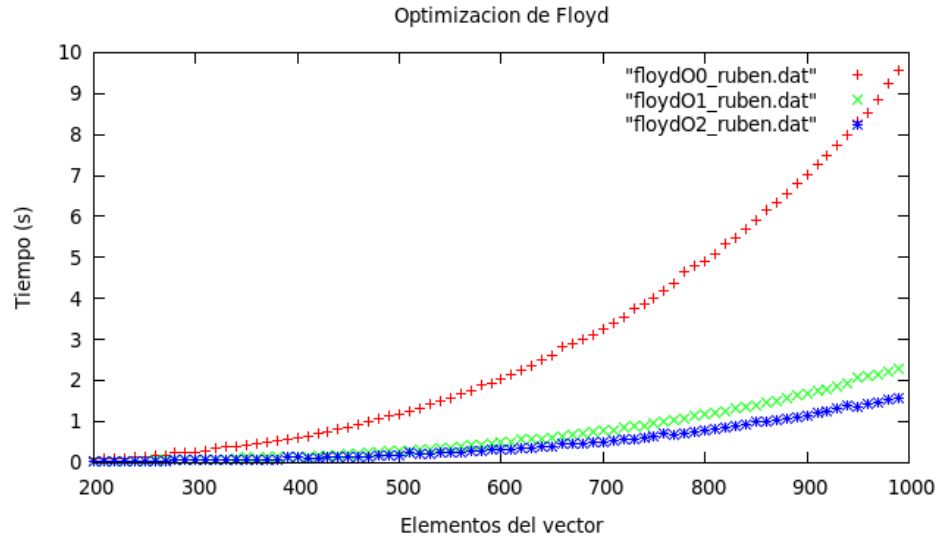
Esto es una prueba gráfica de que hay que tener en cuenta la eficiencia de los algoritmos, ya que la mejora hardware no es suficiente en caso de que tengamos restricciones de tiempo.



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 $O(n \log_2 n)$ 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