

# Presentación práctica de eficiencia

Asignatura: 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

## Presentación

### Introducción

## Algoritmos de ordenación

### Burbuja

### Inserción

### Selección

### Mergesort

### Quicksort

### Comparación

## Resto de algoritmos

### Fibonacci

### Hanoi

### Floyd

## Optimización

### Optimización

## Diferentes ordenadores

## Eficiencia

Hemos usado los siguientes algoritmos en esta práctica

### Ordenación

- ▶ Burbuja
- ▶ Inserción
- ▶ Selección
- ▶ Mergesort
- ▶ Quicksort
- ▶ Heapsort

### Otros

- ▶ Fibonacci
- ▶ Hanoi
- ▶ Floyd

## Script

El siguiente script automatizaba el proceso de obtención de las gráficas y los datos.

### script.sh

```
g++ -std=c++11 ../src/algoritmo.cpp
nelementos=200
while [ $nelementos -lt 10000 ]; do
    ./a.out $nelementos » datos.dat
    let nelementos=nelementos+100
done
gnuplot ./gnuplot/algoritmo.gp # Salida: fichero.jpeg
mv fichero.jpeg ../Graficas/Algoritmo/algoritmo.jpeg
mv datos.dat ../Graficas/Algoritmo/Datos/algoritmo.dat
```

## Scripts de gnuplot

Podemos hacer que gnuplot automatice su trabajo.

Suponemos que el fichero donde están los datos es "datos.dat", como indicamos anteriormente los scripts de gnuplot se ejecutan:

gnuplot algoritmo.gp

### algoritmo.gp

```
set terminal pngcairo
set output "fichero.jpeg"
set title "Eficiencia algoritmo"
set xlabel "Tamaño 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)
```

### Funciones ajustadas

$$f(x) = ax^3 + bx^2 + cx + d$$

$$g(x) = ax^2 + bx + c$$

$$h(x) = ax \cdot \log_2(x)$$

$$i(x) = a \cdot ((1 + \sqrt{(5)}))/2)^x$$

## Scripts de gnuplot

Podemos hacer que gnuplot automatice su trabajo.

Suponemos que el fichero donde están los datos es "datos.dat", como indicamos anteriormente los scripts de gnuplot se ejecutan:

```
gnuplot algoritmo.gp
```

### algoritmo.gp

```
set terminal pngcairo
set output "fichero.jpeg"
set title "Eficiencia algoritmo"
set xlabel "Tamaño 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)
```

### Funciones ajustadas

$$f(x) = ax^3 + bx^2 + cx + d$$

$$g(x) = ax^2 + bx + c$$

$$h(x) = ax \cdot \log_2(x)$$

$$i(x) = a \cdot ((1 + \sqrt{(5)}))/2)^x$$

## Scripts de gnuplot

Podemos hacer que gnuplot automatice su trabajo.

Suponemos que el fichero donde están los datos es "datos.dat", como indicamos anteriormente los scripts de gnuplot se ejecutan:

```
gnuplot algoritmo.gp
```

### algoritmo.gp

```
set terminal pngcairo
set output "fichero.jpeg"
set title "Eficiencia algoritmo"
set xlabel "Tamaño 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)
```

### Funciones ajustadas

$$f(x) = ax^3 + bx^2 + cx + d$$

$$g(x) = ax^2 + bx + c$$

$$h(x) = ax \cdot \log_2(x)$$

$$i(x) = a \cdot ((1 + \sqrt{(5)}))/2)^x$$

## 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



# Burbuja

## Función

Este algoritmo tiene una eficiencia cuadrática, debemos ajustar una función del tipo  $f(x) = ax^2 + bx + c$

## Ajuste

$$f(x) = a \cdot x^2 + b \cdot x + c$$

En el ajuste también tenemos un margen de error:

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

# Burbuja

## Función

Este algoritmo tiene una eficiencia cuadrática, debemos ajustar una función del tipo  $f(x) = ax^2 + bx + c$

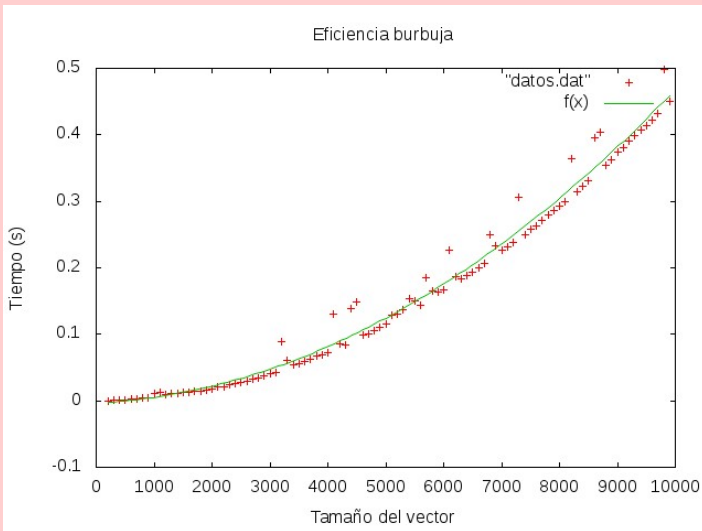
## Ajuste

$$f(x) = a \cdot x^2 + b \cdot x + c$$

En el ajuste también tenemos un margen de error:

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

# Imagen



# Inserción

## Función

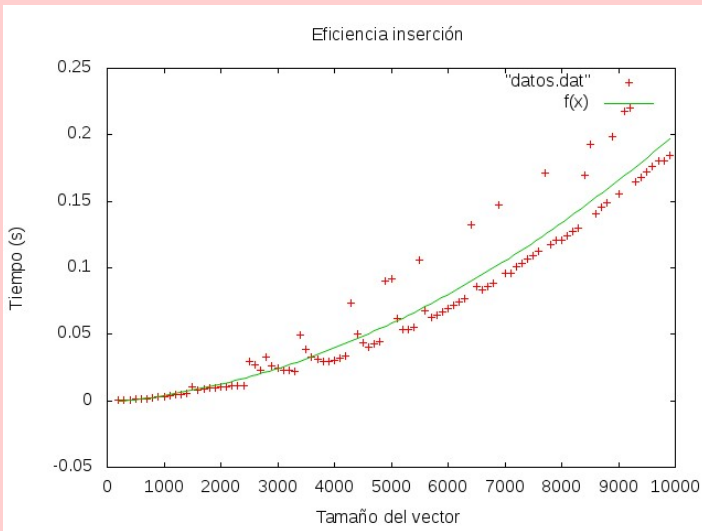
Aunque el algoritmo de inserción tenga una eficiencia  $O(n^2)$  tiene una constante multiplicativa menor que el burbuja, y similar al selección.

## Ajuste

$$f(x) = a \cdot x^2 + b \cdot x + c$$

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

# Imagen



# Selección

## Función

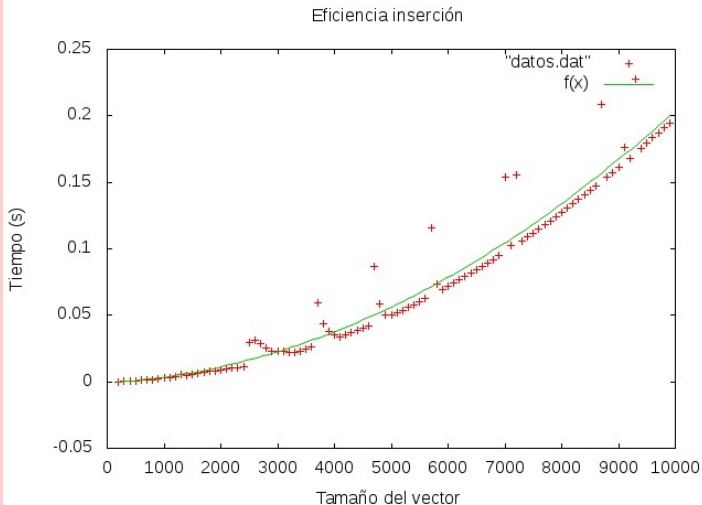
Este algoritmo tiene dos partes una ordenada y otra no, en cada iteración coge el máximo/mínimo de los elementos no ordenados y los inserta en los ordenados.

## Ajuste

$$f(x) = a \cdot x^2 + b \cdot x + c$$

$$a = 2,36327 \cdot 10^{-9} \pm 3,232 \cdot 10^{-11} (1,368 \%)$$

# Imagen



# Mergesort

## Función

Siguiente algoritmo de ordenación: mergesort

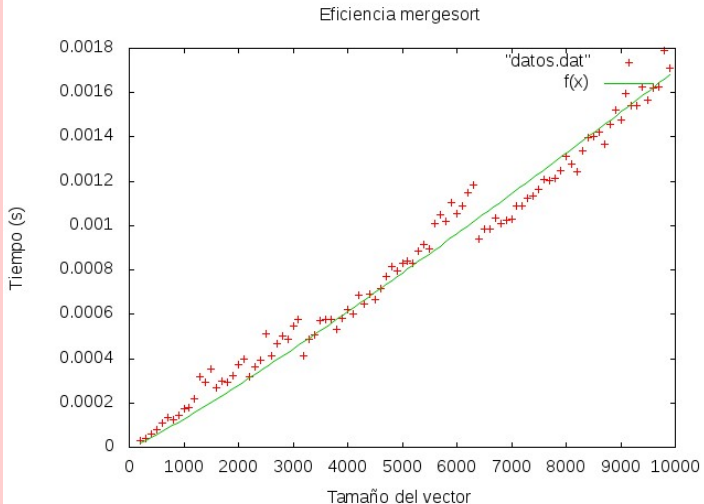
## Ajuste

$$f(x) = a \cdot x \cdot \log_2(x)$$

$$a = 3,5231 \cdot 10^{-8} \pm 1,191 \cdot 10^{-9} (3,382 \%)$$



# Imagen



# Quicksort

## Función

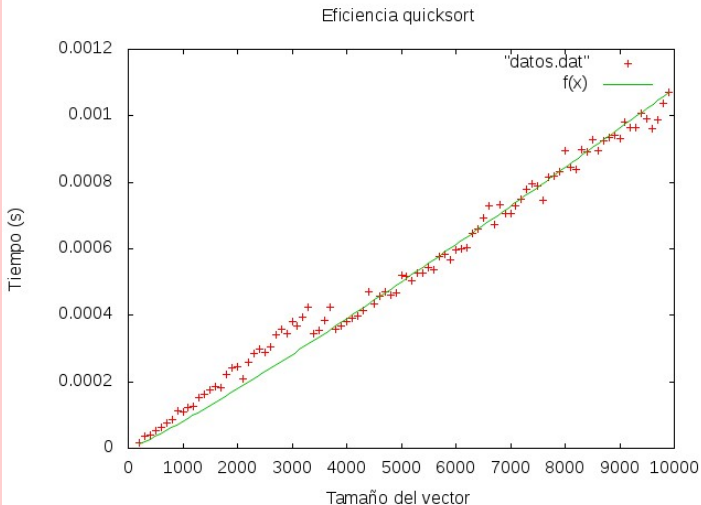
El algoritmo de ordenación más rápido en término medio: quicksort

## Ajuste

$$f(x) = a \cdot x \cdot \log_2(x)$$

$$a = 2,3704 \cdot 10^{-8} \pm 5,497 \cdot 10^{-10} (2,319 \%)$$

# Imagen

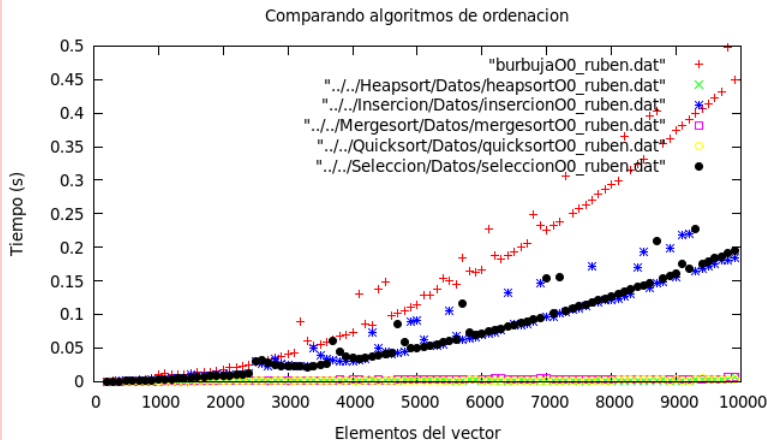


# Algoritmos de ordenación

## Función

Se observa una diferencia notable entre los algoritmos  $O(n \log_2(n))$  y los  $O(n^2)$ , casi no se aprecian los primeros.

# Imagen



# Fibonacci

## Función

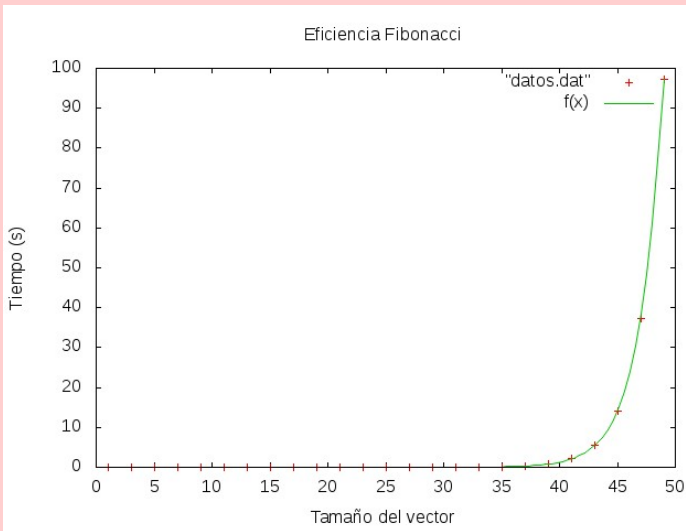
Fibonacciiii

## Ajuste

$$f(x) = a \cdot \left( \frac{1 + \sqrt{5}}{2} \right)^x$$

$$a = 5,59738 \cdot 10^{-9} \pm 2,093 \cdot 10^{-12} (0,0374 \%)$$

# Imagen



# Hanoi

## Función

Hanoi

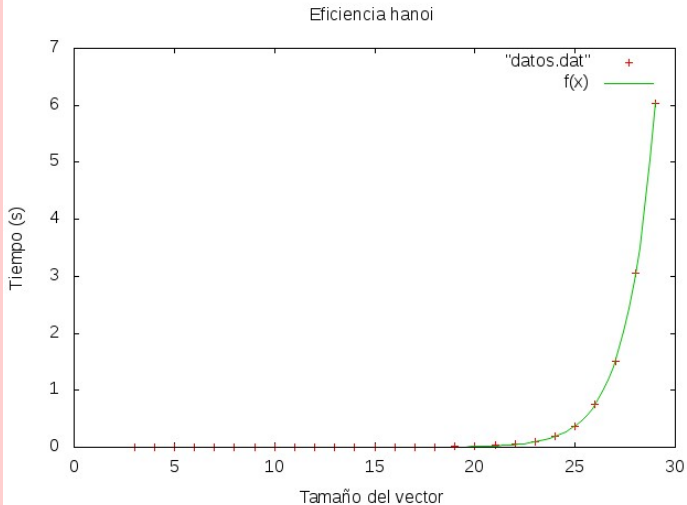
## Ajuste

$$f(x) = a \cdot (2^x)$$

$$a = 1,12636 \cdot 10^{-8} \pm 1,391 \cdot 10^{-11} (0,1235 \%)$$



# Imagen



# Floyd

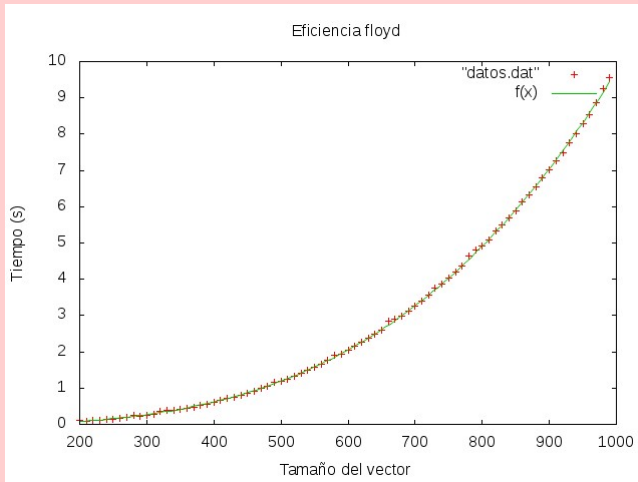
## Función

Tipo de algoritmo con programación dinámica para encontrar el camino mínimo en grafos ponderados.

## Ajuste

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

# Imagen



# Optimizando algoritmos

## Algoritmos

En este apartado optimizaremos diferentes algoritmos.

- ▶ Burbuja
- ▶ Quicksort
- ▶ Floyd

## Observación

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).

# Optimizando algoritmos

## Algoritmos

En este apartado optimizaremos diferentes algoritmos.

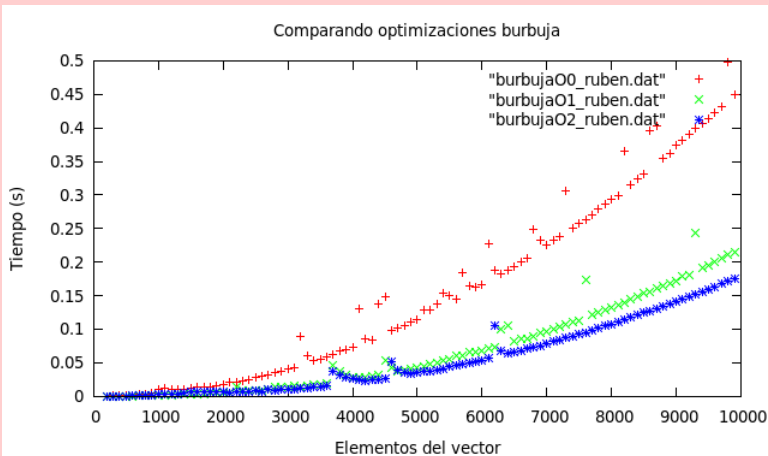
- ▶ Burbuja
- ▶ Quicksort
- ▶ Floyd

## Observación

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).

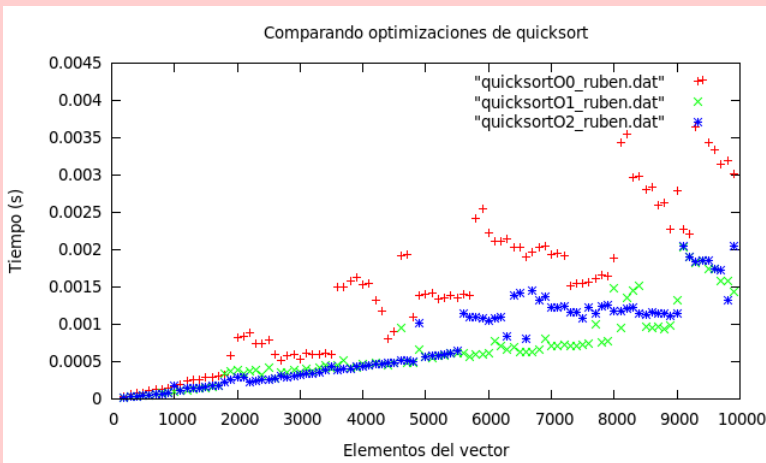
# Burbuja optimizado

## Imagen



# Quicksort optimizado

## Imagen



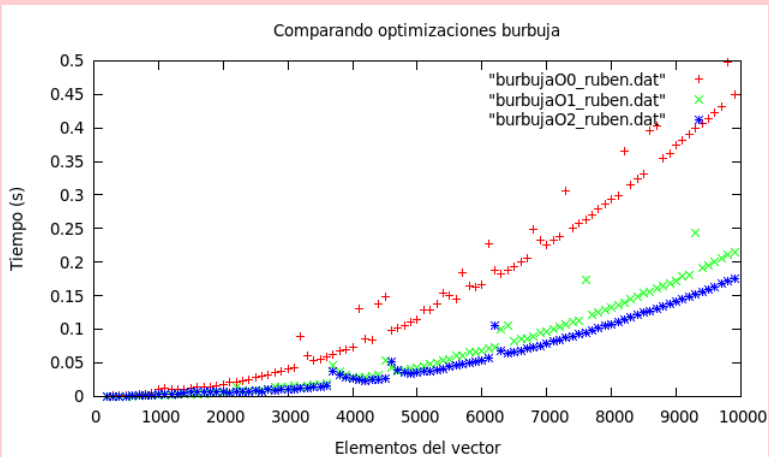
## Conclusión

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.



# Burbuja optimizado

## Imagen



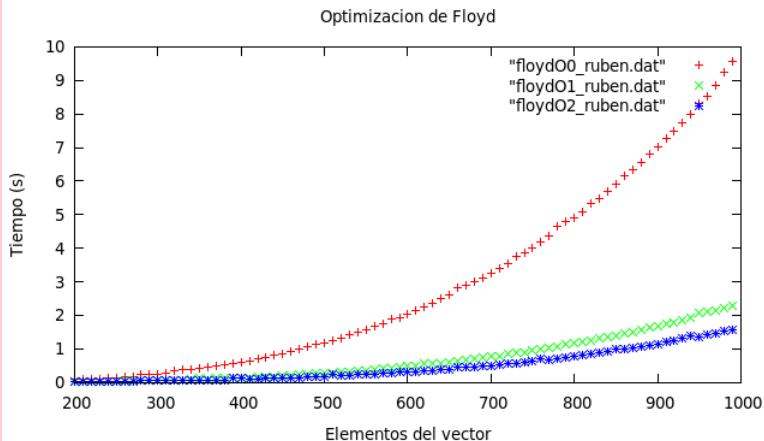
# Floyd

## Algoritmo

El algoritmo floyd, tipo de algoritmo con programación dinámica para encontrar el camino mínimo en grafos ponderados.

# Floyd

## Imagen



# Diferentes ordenadores

## Imagen

