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

13 de abril de 2016

Índice

Eficiencia

Divide y vencerás es una técnica algorítmica que consiste en resolver un problema diviéndolo en problemas más pequeños y combinando las soluciones. El proceso de división continúa hasta que los subproblemas llegan a ser lo suficientemente sencillos como para una resolución directa. El hecho de que el tamaño de los subproblemas sea estrictamente menor que el tamaño original del problema nos garantiza la convergencia hacia los casos elementales.

Script

Podemos obtener los datos fijando el número de vectores usados.

script.sh

```
g++ -std=c++11 ../src/mezcla.cpp
nelementos=10
while [ $nelementos -lt 2500 ]; do
./a.out $nelementos 200 3
let nelementos=nelementos+25
done
```

Script

Si queremos fijar el número de vectores usaremos

script.sh

```
kvectores=10
while [ $kvectores -lt 2500 ]; do
./a.out 200 $kvectores 2
let kvectores=kvectores+25
done
```

Script

Datos en 3 dimensiones, número de vectores, elementos del vector, y tiempo del algoritmo

script.sh

```
nelementos=10
nvectores=10
while [ $nelementos -lt 1000 ]; do
./a.out $nelementos 10 1
./a.out $nelementos 110 1
./a.out $nelementos 210 1
./a.out $nelementos 310 1
./a.out $nelementos 410 1
./a.out $nelementos 510 1
./a.out $nelementos 610 1
./a.out $nelementos 710 1
./a.out $nelementos 810 1
```

Scripts de gnuplot

Podemos hacer que gnuplot automatice su trabajo.
Suponemos que el fichero donde están los datos es "datos.dat", los scripts de gnuplot se ejecutan:

$ gnuplot algoritmo.gp

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set title "..."
set xlabel "Numero de
vectores/elementos del vector"
set ylabel "Tiempo (s)"
set fit quiet
f(x) = ...
fit f(x) "datos.dat"via a
plot "datos.dat", f(x)
```

Funciones ajustadas

$$f(x) = a * x$$

$$g(x) = a * x * x$$

$$h(x) = a * x * (log(x)/log(2))$$

Scripts de gnuplot

Podemos hacer que gnuplot automatice su trabajo.
Suponemos que el fichero donde están los datos es "datos.dat", los scripts de gnuplot se ejecutan:

$ gnuplot algoritmo.gp

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set title "..."
set xlabel "Numero de
vectores/elementos del vector"
set ylabel "Tiempo (s)"
set fit quiet
f(x) = ...
fit f(x) "datos.dat"via a
plot "datos.dat", f(x)
```

Funciones ajustadas

$$f(x) = a * x$$

$$g(x) = a * x * x$$

$$h(x) = a * x * (log(x)/log(2))$$

Scripts de gnuplot

Podemos hacer que gnuplot automatice su trabajo.
Suponemos que el fichero donde están los datos es "datos.dat", los scripts de gnuplot se ejecutan:

$ gnuplot algoritmo.gp

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set title "..."
set xlabel "Numero de
vectores/elementos del vector"
set ylabel "Tiempo (s)"
set fit quiet
f(x) = ...
fit f(x) "datos.dat"via a
plot "datos.dat", f(x)
```

Funciones ajustadas

$$f(x) = a * x$$

$$g(x) = a * x * x$$

$$h(x) = a * x * (log(x)/log(2))$$

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

Comparación de preferencias

Ordenador usado para la ejecución

Asus N56VJ
Sistema operativo: Linux mint (Rosa)
Memoria: 8GB
Procesador: Inter Core i7-4710HQ x 8
Gráficos: Nvidia geforce 750M
Tipo de SO: 64 bits
Disco: 1TB

Problema

Explicación

Queremos comparar las preferencias de dos personas sobre un número n de productos (películas, musica, ...). Para ello contaremos el número de inversiones en su valoración de los productos.
Consideramos que una valoración está invertida cuando A prefiere el producto j antes que i y B prefiere el producto i antes que j.

Simplificación

Obj	A	B
1	3	2
2	1	3
3	2	1

Obj	A	B
3	3	2
1	1	3
2	2	1

A	B	v
1	3	3
2	1	1
3	2	2

Fuerza bruta

Algoritmo

La aproximación más básica al problema es comprobar en cada elemento si los siguientes en el vector son menores.

Criterio

Dos elementos están invertidos si $i < j$ pero $v[i] > v[j]$

Implementación

```
int inversiones(vector<int> v){
  int count = 0;
  int size = v.size();
  for (int i=0; i < size; i++)
    for (int j=i+1; j < size; j
        ++)
      if (v[i] > v[j])
        count++;

  return count;
}
```

Fuerza bruta

Algoritmo

La aproximación más básica al problema es comprobar en cada elemento si los siguientes en el vector son menores.

Criterio

Dos elementos están invertidos si $i < j$ pero $v[i] > v[j]$

Implementación

```
int inversiones(vector<int> v){
  int count = 0;
  int size = v.size();
  for (int i=0; i < size; i++)
    for (int j=i+1; j < size; j
        ++)
      if (v[i] > v[j])
        count++;

  return count;
}
```

Eficiencia

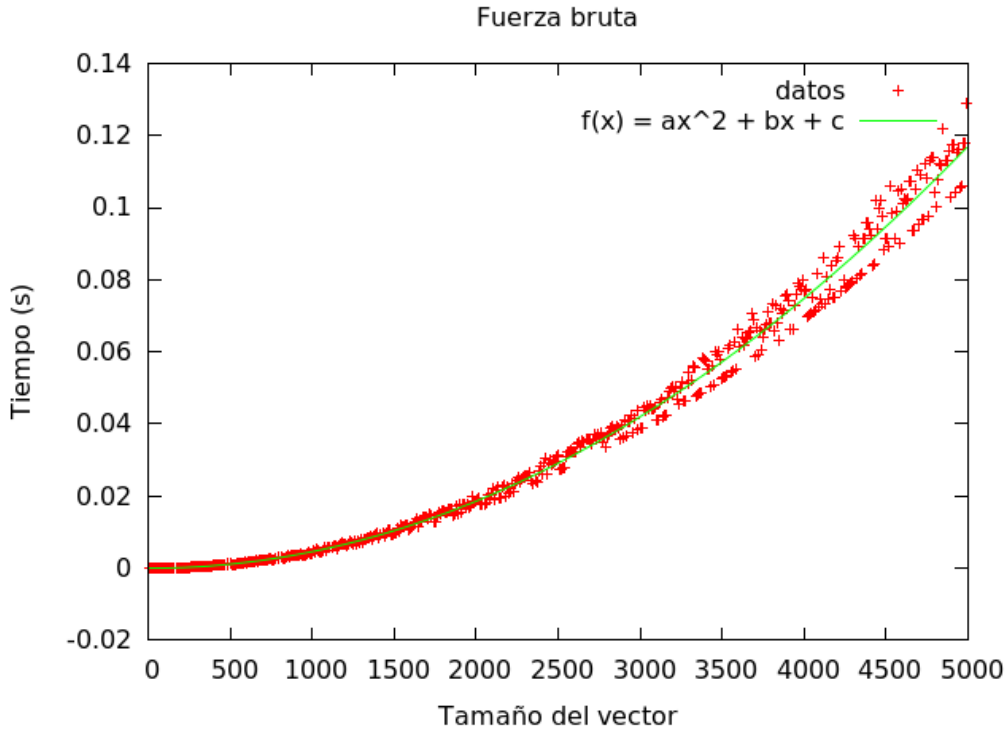
Eficiencia teórica

$$T(n) = \sum_{i=1}^{n-1} n - i$$

$$T(n) = \frac{n^2 - n}{2} \longrightarrow O(n^2)$$

Ajuste

$a = 4{,}62037 \cdot 10^{-9}$
$b = 7{,}56764 \cdot 10^{-9}$
$c = -3{,}85366 \cdot 10^{-5}$



Divide y vencerás

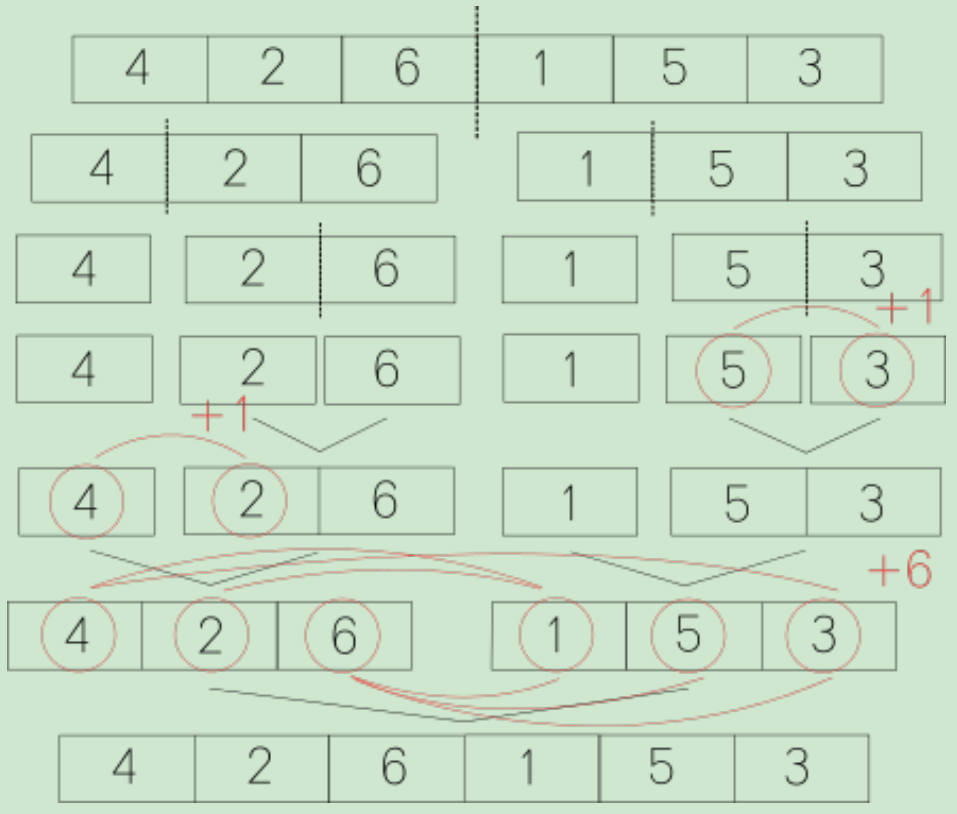
Algoritmo

Utilizamos el método de mezcla para contar el número de inversiones en cada sección.

Criterio

Dos elementos están invertidos si $i < j$ pero $v[i] > v[j]$

Ejemplo



Implementación

```
int fusion(int T[], int inicial, int final, int U[],
    int V[])
{
  int n_desordenados = 0;
  int k = (final-inicial)/2;
  for (int i =0; i < k-inicial; i++)
    for (int j=0; j < final-k; j++)
      if (U[i] > V[j])
        n_desordenados++;

  int i, j;
  for (i =0; i < k-inicial; i++)
    T[i] = U[i];
  for (i=0, j=k-inicial; j < final-k; i++, j++)
    T[j] = V[i];

  return n_desordenados;
}
```

Eficiencia teórica

Recurrencia

$$\begin{cases} T(n) = \frac{n^2-n}{2} \ si \ n \leq 2 \\ T(n) = 2T(\frac{n}{2}) + 2n \ si \ n > 2 \end{cases}$$

Solución

$$n = 2^k \Rightarrow k = \log_2 n$$

$$T(2^k) = 2T(2^{k-1}) + 2^{k+1}$$

$$T(k) = c_1 2^k + c_2 k 2^k \Rightarrow T(n) = n^2(c_1 + c_2 \log_2 n)$$

$$O(n^2 \log_2 n)$$

Eficiencia teórica

Recurrencia

$$\begin{cases} T(n) = \frac{n^2-n}{2} \ si \ n \leq 2 \\ T(n) = 2T(\frac{n}{2}) + 2n \ si \ n > 2 \end{cases}$$

Solución

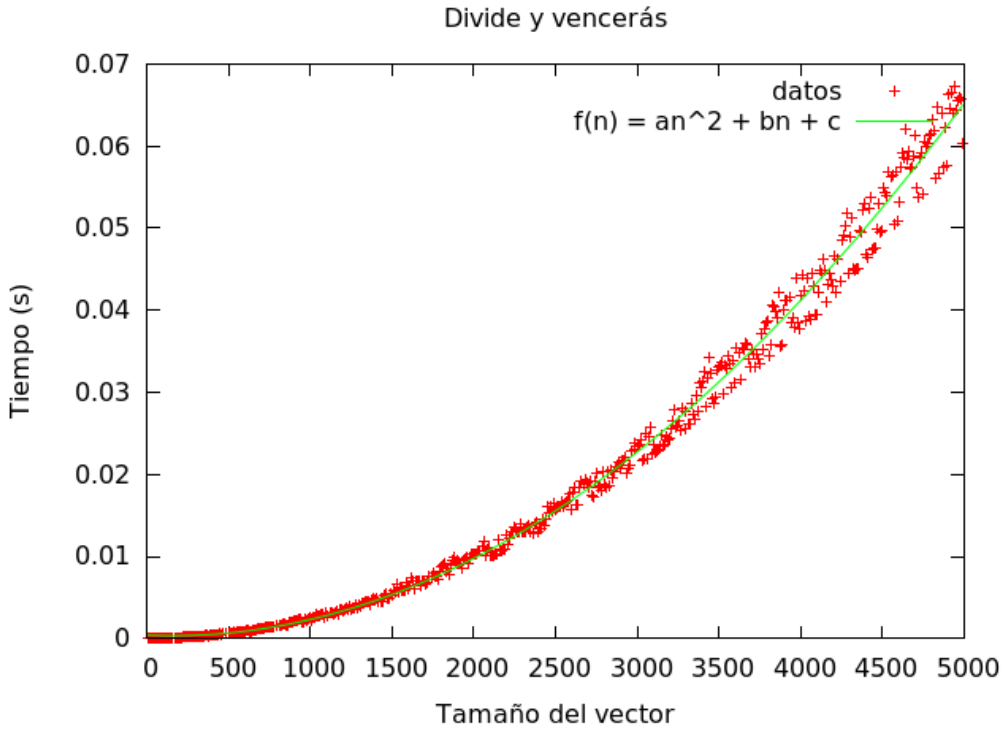
$$n = 2^k \Rightarrow k = \log_2 n$$

$$T(2^k) = 2T(2^{k-1}) + 2^{k+1}$$

$$T(k) = c_1 2^k + c_2 k 2^k \Rightarrow T(n) = n^2(c_1 + c_2 \log_2 n)$$

$$O(n^2 \log_2 n)$$

Eficiencia empírica



Ajuste

$$a = 2{,}6999 \cdot 10^{-9}$$

Divide y vencerás con mergesort

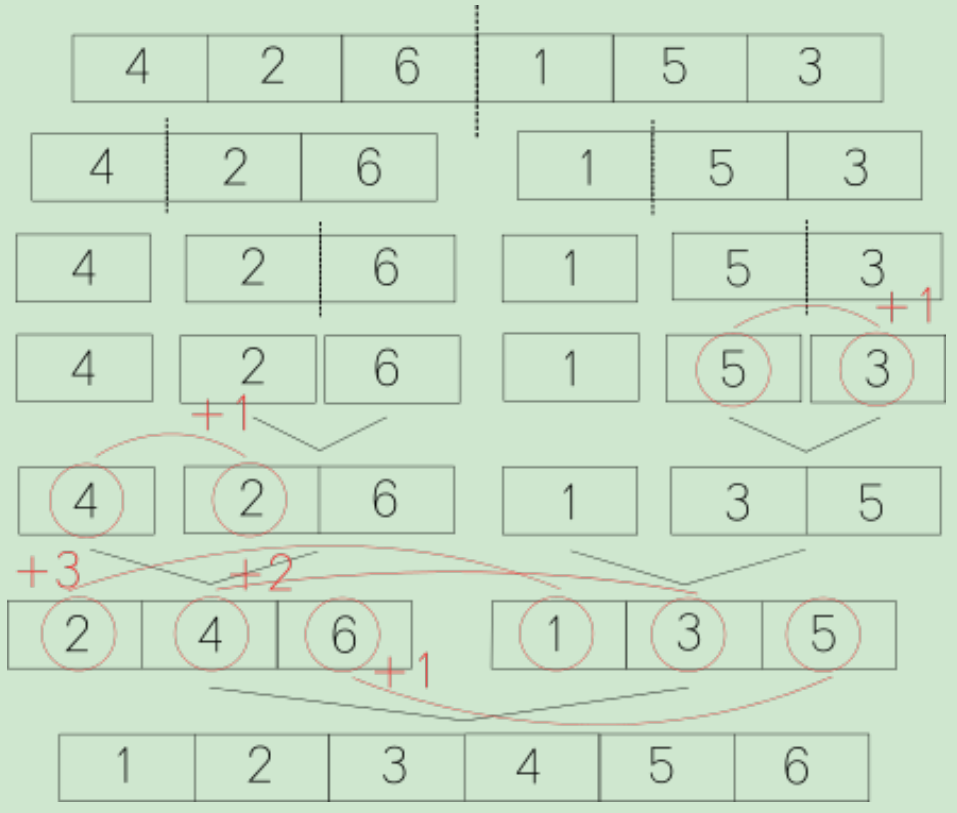
Algoritmo

Utilizamos el algoritmo de ordenación mergesort para contar el número de inversiones.

Criterio

Dos elementos están invertidos si $i < j$ pero $v[i] > v[j]$

Ejemplo



Implementación

```
int fusion(int T[], int inicial, int final, int U[],
    int V[]){
  int j = 0;
  int k = 0;
  int num_inversiones = 0;
  for (int i = inicial; i < final; i++){
    if (U[j] < V[k]){
       T[i] = U[j];
        j++;
    } else{
       T[i] = V[k];
        k++;
        if (U[j] < INT_MAX)
          num_inversiones += (final - inicial)/2 - j;
    }
  }

  return num_inversiones;
}
```

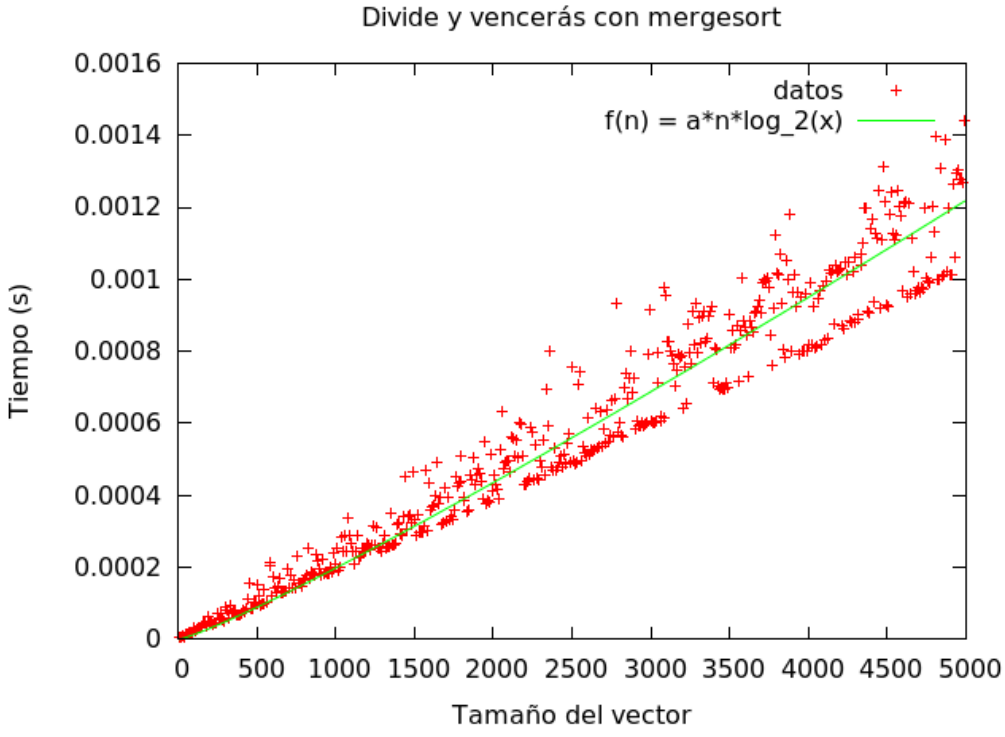
Eficiencia

Eficiencia teórica

Tiene la misma eficiencia que mergesort, $O(n \log n)$

Ajuste

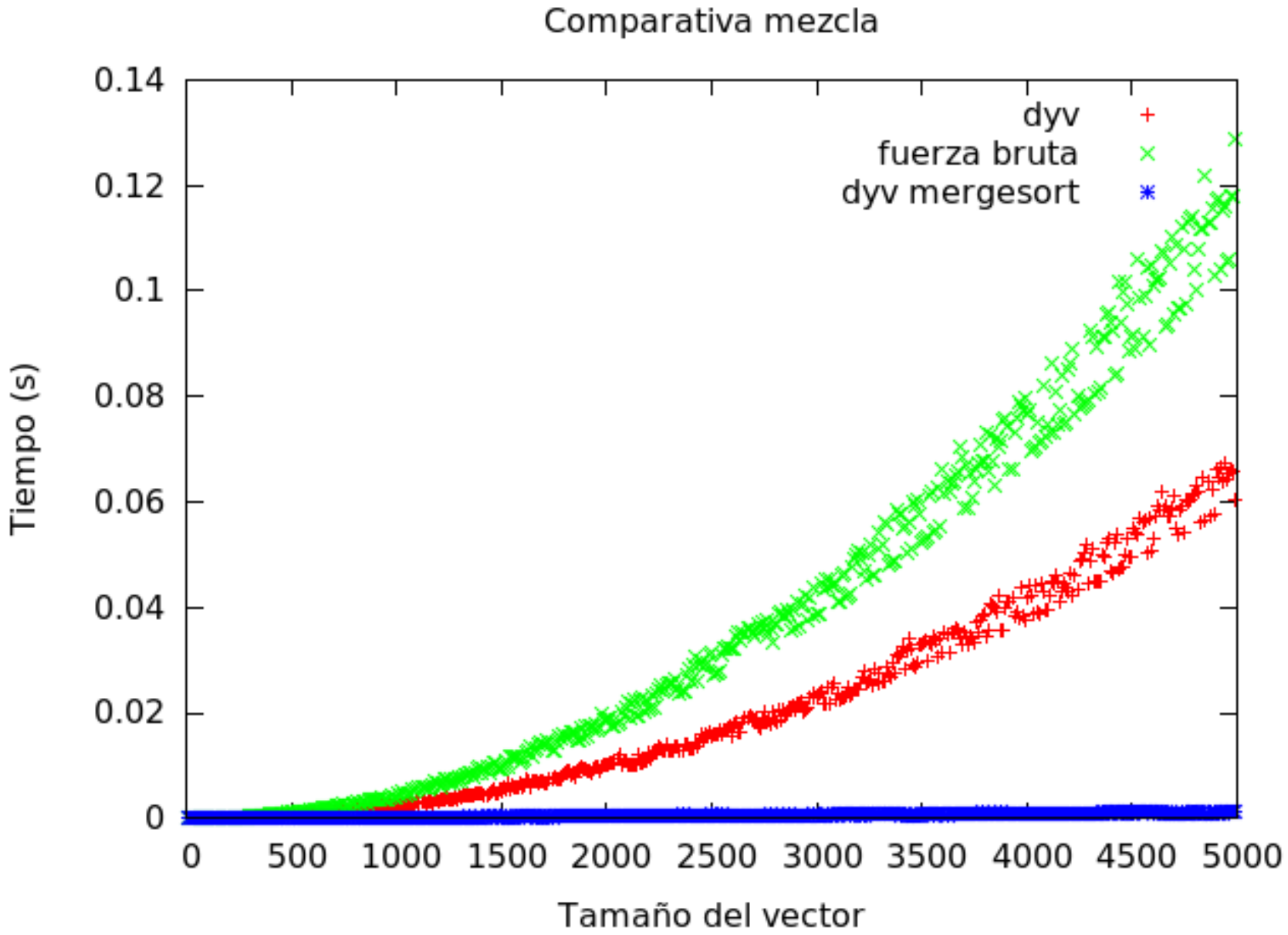
$a = 1{,}99872 \cdot 10^{-8}$



Comparación

Tabla comparativa

N	FUERZA BRUTA	DyV	MERGESORT
10	5.372e-06	5.01e-06	4.475e-06
100	4.3868e-05	4.5584e-05	1.7295e-05
500	0.00113	0.000778726	8.8503e-05
1000	0.0045925	0.00247251	0.000218733
1500	0.0106898	0.00567031	0.000289624
2000	0.0171023	0.0094678	0.000382864


Comparativa mezcla
dyv
fuerza bruta
dyv mergesort
Tiempo (s)
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Tamaño del vector