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

21 de abril de 2016

Índice

Introducción

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.

Problema

Mezclando k vectores ordenados

Se tienen k vectores ordenados (de menor a mayor), cada uno con n elementos, y queremos combinarlos en un único vector ordenado (con kn elementos)

Cota superior

Es posible imponer una cota superior teórica. Teniendo en cuenta que hay kn elementos, si aplicásemos un algoritmo de ordenación con eficiencia $O(n) = nlog(n)$ deducimos que podemos encontrar un algoritmo de ordenación básica con eficiencia $O(k, n) = nklog(nk)$. Tomar los k vectores como uno solo no aprovecha aún el hecho de que partes del vector están ordenadas.

Scripts

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 910 1
let nelementos=nelementos+100 done
```

Scripts de gnuplot

Gnuplot

Los datos están en "datos.dat". Ejecutamos

$ gnuplot algoritmo.gp

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set xlabel "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

Gnuplot

Los datos están en "datos.dat". Ejecutamos

```
$ gnuplot algoritmo.gp
```

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set xlabel "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

Gnuplot

Los datos están en "datos.dat". Ejecutamos

$ gnuplot algoritmo.gp

algoritmo.gp

```
set terminal pngcairo
set output "grafica.png"
set xlabel "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

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

Fuerza bruta

Algoritmo

En cada paso elegimos el mínimo de los primeros elementos de los k vectores, será el primer elemento del vector creciente resultante. Para el siguiente paso descartamos ese elemento y calculamos otra vez el mínimo, lo insertamos al final del vector resultante y así sucesivamente.
Buscar el mínimo es $O(k) = k$ ya que el vector de índices tiene k elementos, y lo repetimos kn veces.

Eficiencia

$$\sum_{i=1}^{kn} k = nk^2 \implies O(k, n) = nk^2$$

Divide y vencerás

Algoritmo

Usaremos mergesort, pero con los primeros montículos ya creados, por tanto tendrá una constante oculta menor que usar mergesort para kn datos arbitrarios.
En el proceso lo que haremos es ir mezclando las partes de dos en dos. El algoritmo que mezcla dos vectores en un único tiene eficiencia $O(n) = n$.

Código

```
void MergeKPartitions(int* &vector, int n_elem, int ini, int fin){
  int size = fin - ini + 1, partitions = size/n_elem;
  if(partitions == 2) // Caso base
    Merge(vector, ini, ini+n_elem, fin);
  else if(partitions > 2){
    int division = ini + (partitions/2)*n_elem; // Cálculo de la división
    MergeKPartitions(vector, n_elem, ini, division-1);  // Primer vector ordenado
    MergeKPartitions(vector, n_elem, division, fin);    // Segundo vector ordenado
    Merge(vector, ini, division, fin);  // Mezclamos los dos conjuntos
  }
}
```

Eficiencia

Donde k es el número de vectores y n el número de elementos de cada vector:

$$T(k,n) = \begin{cases} 2n & \text{si } k = 2 \\ 2T(k/2,n) + kn & \text{si } k > 2 \end{cases}$$

Desarrollo

Sustituyendo $k = 2^m \implies T(2^m, n) = 2T(2^{m-1}, n) + 2^m n$

$$T(2^m, n) = 2\left[T(2^{m-2}, n) + 2^{m-1}n\right] + 2^m n$$

Para el caso genérico, con $j \in [0, m-1] \cap \mathbb{N}$ y desarrollando:

$$T(2^m, n) = 2^j T(2^{m-j}, n) + \sum_{i=1}^{m-1} 2^m n$$

$$T(2^m, n) = 2^{m-1} T(2, n) + \sum_{i=1}^{m-1} 2^m n$$

$$T(2^m, n) = 2^m n + (m-1)2^m n = 2^m n[1 + (m-1)] = 2^m nm$$

Eficiencia final

Solución

Deshacemos el cambio de variable, $k = 2^m \implies log_2(k) = m$:

$$T(k, n) = knlog_2 k$$

Ajuste fuerza bruta

Vamos a variar el número de vectores, la función que debemos ajustar es $f(x) = ax^2$
$a = 1{,}77962 \cdot 10^{-6}$
Para calcular los coeficientes de correlación hemos usado la función stats de gnuplot:

Variando vectores

Imagen

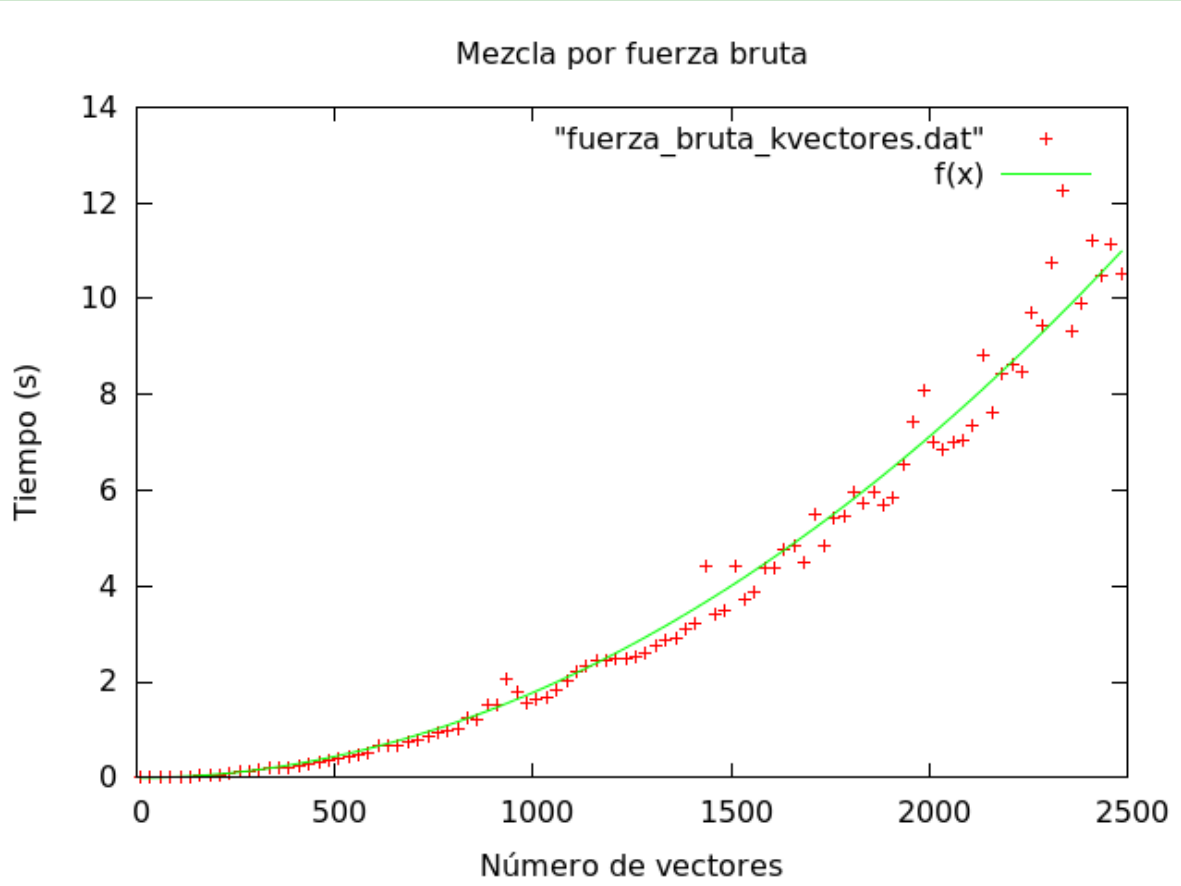


Figura : Fuerza bruta con 200 elementos cada vector

Para la parte en la que cambiamos el número de elementos ajustamos la función $f(x) = ax$ ya que en $T(k, n) = nklog_2 k$, $klog_2 k$ es una constante, concretamente $200 \cdot log_2(200)$.

$$a = 0{,}00031202$$

Correlation: $r = 0{,}9934$

Imagen

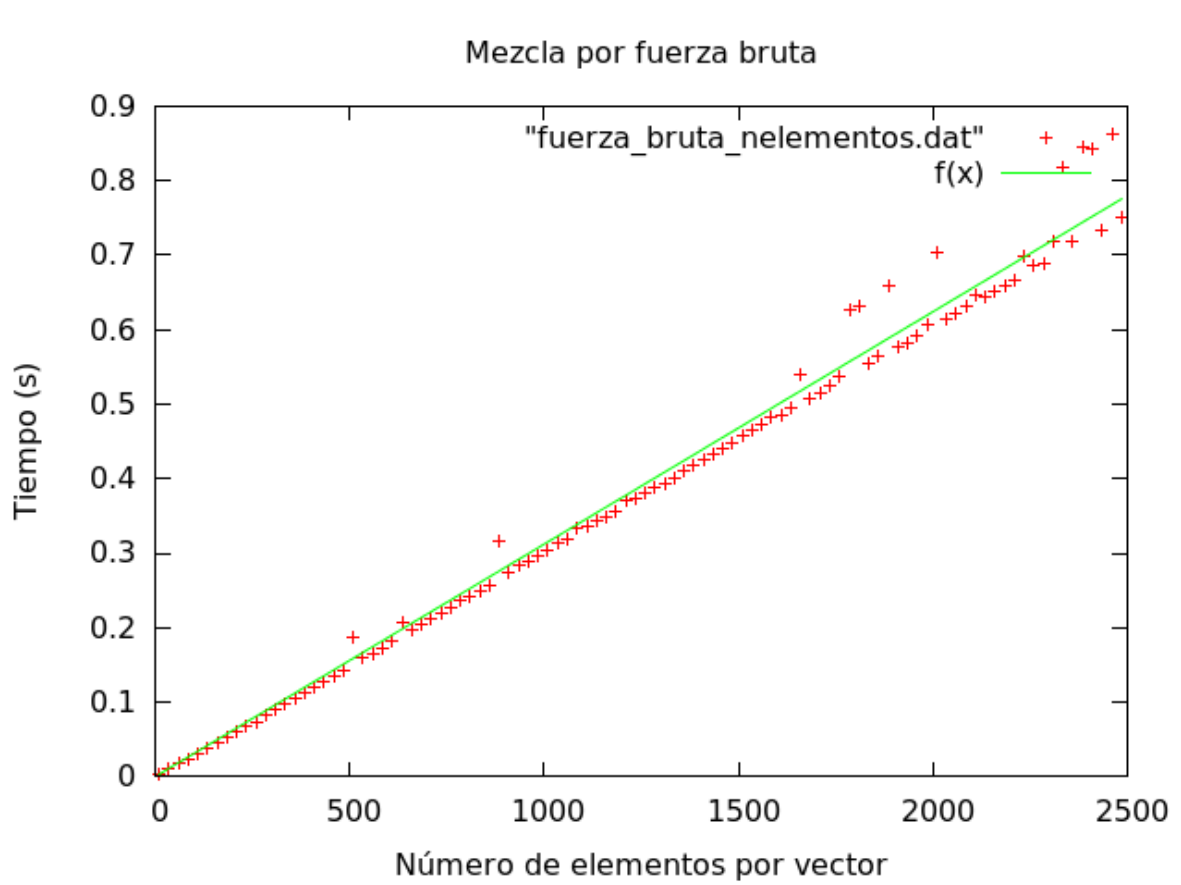


Figura : Fuerza bruta con 200 vectores

A

hora usaremos el algoritmo divide y vencerás. Ahora el eje de abscisas serán los vectores usados. La función ajustada ha sido $f(x) = ax(log(x)/log(2))$

$$a = 4{,}52594 \cdot 10^{-6} \text{ Correlation: } r = 0{,}9863$$

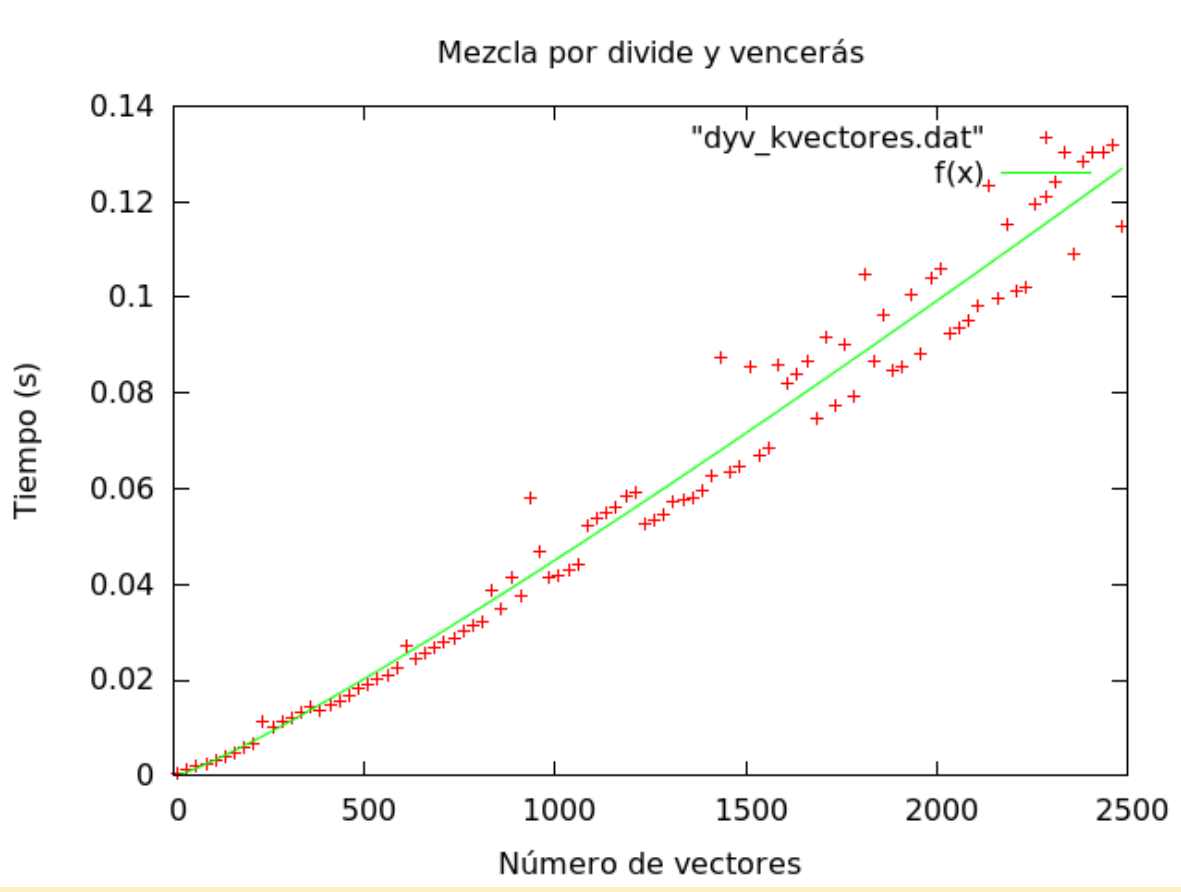


Figura : Divide y vencerás con 200 elementos cada vector

Si ahora fijamos $k = 200$ y hacemos variable el número de elementos debemos ajustar la función $f(x) = ax(log(200)/log(2))$

$$a = 3{,}03036 \cdot 10^{-6} \text{ Correlation: } r = 0{,}9933$$

Imagen

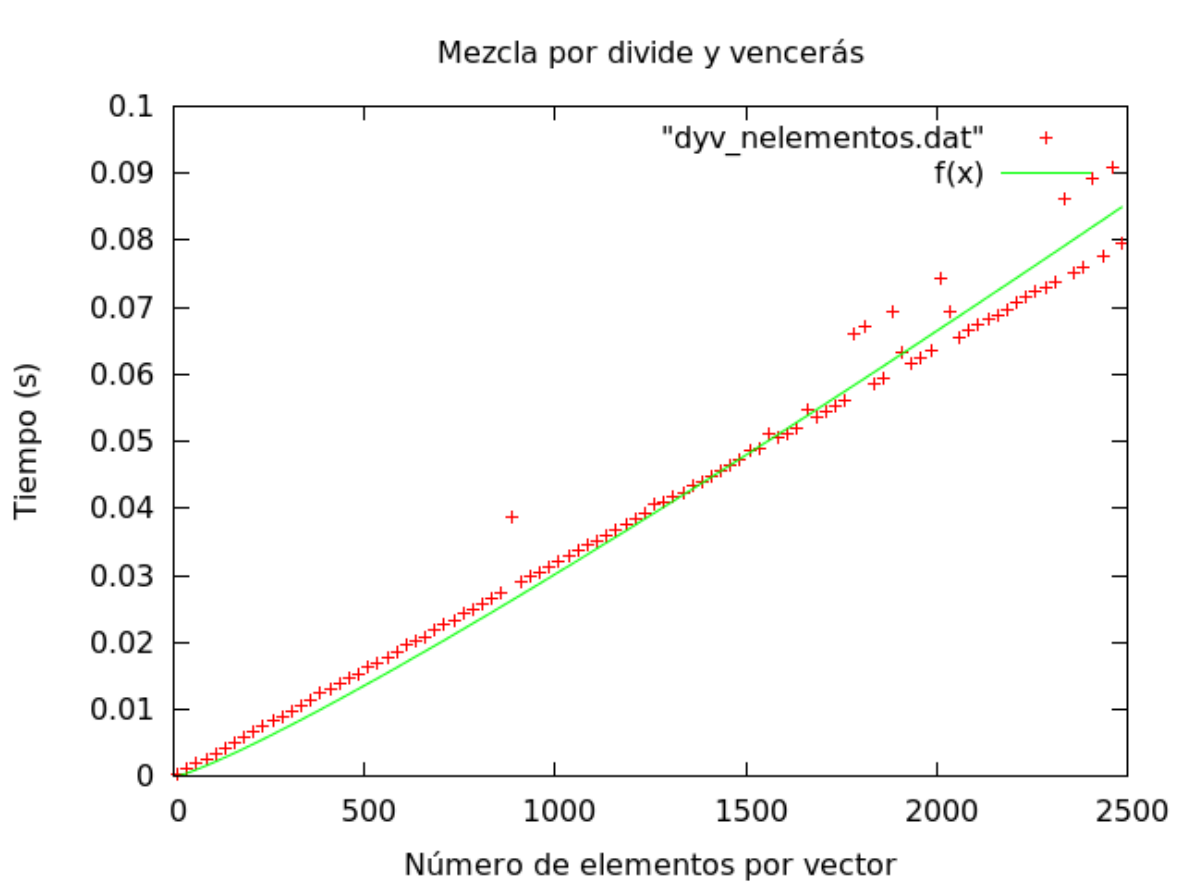


Figura : Divide y venceras con 200 vectores

Según el número de vectores

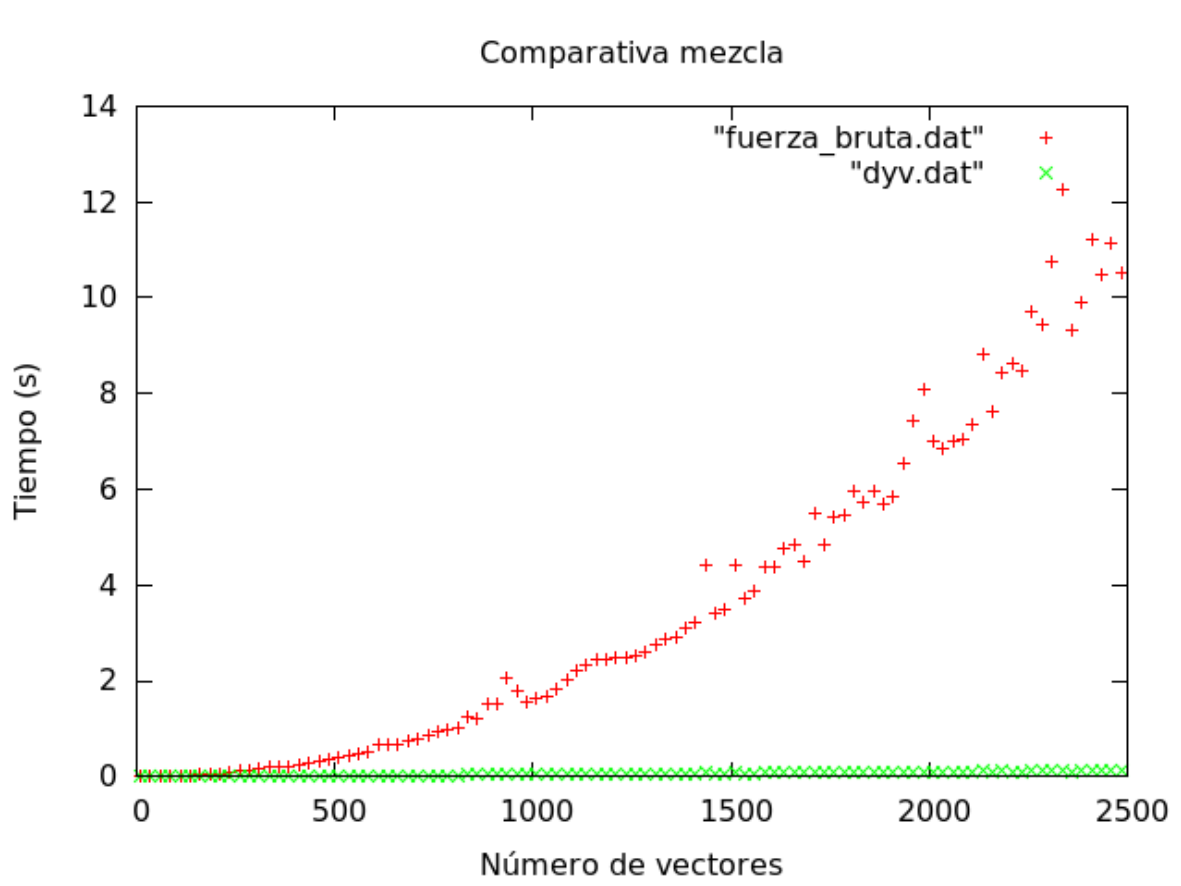


Figura : Comparativa con 200 elementos cada vector

Según el número de elementos de cada vector

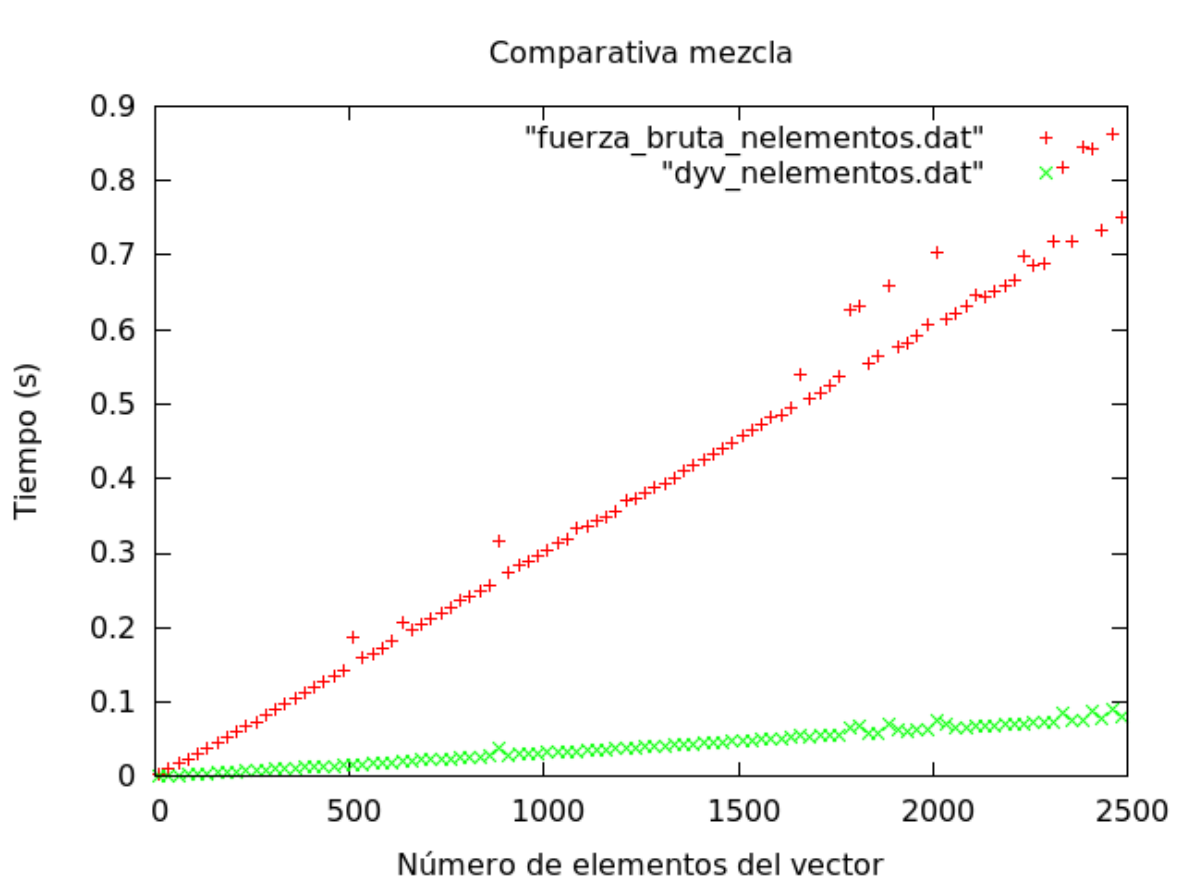


Figura : Comparativa con 200 vectores

Si queremos combinar los resultados podemos variar simultáneamente el número de vectores y los elementos del vector tenemos que crear gráficas en 3 dimensiones. De esta forma podemos comprobar que el número de vectores es un factor más influyente que el número de elementos de cada vector. Sin embargo este es un dato que se aprecia en el algoritmo por fuerza bruta, en el divide y vencerás apenas se nota diferencia.

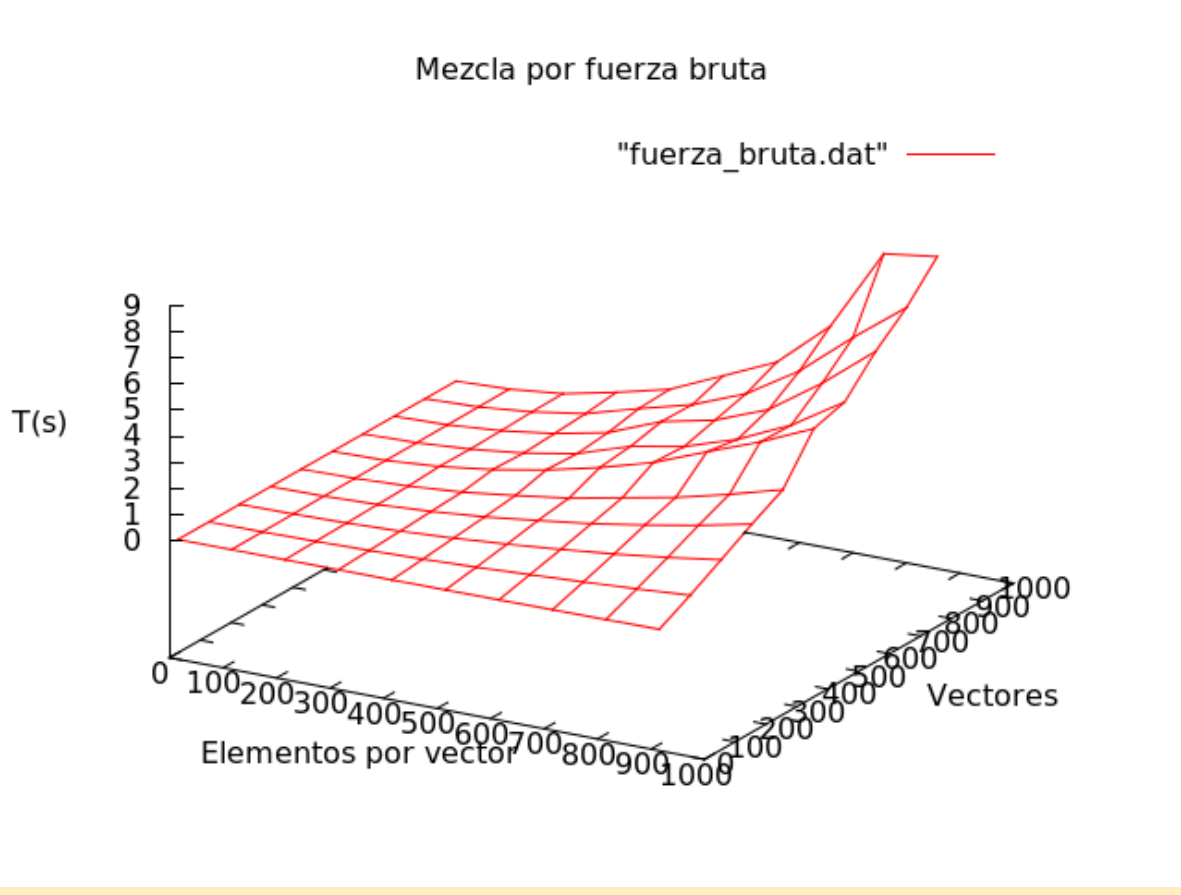


Figura : 3d fuerza bruta

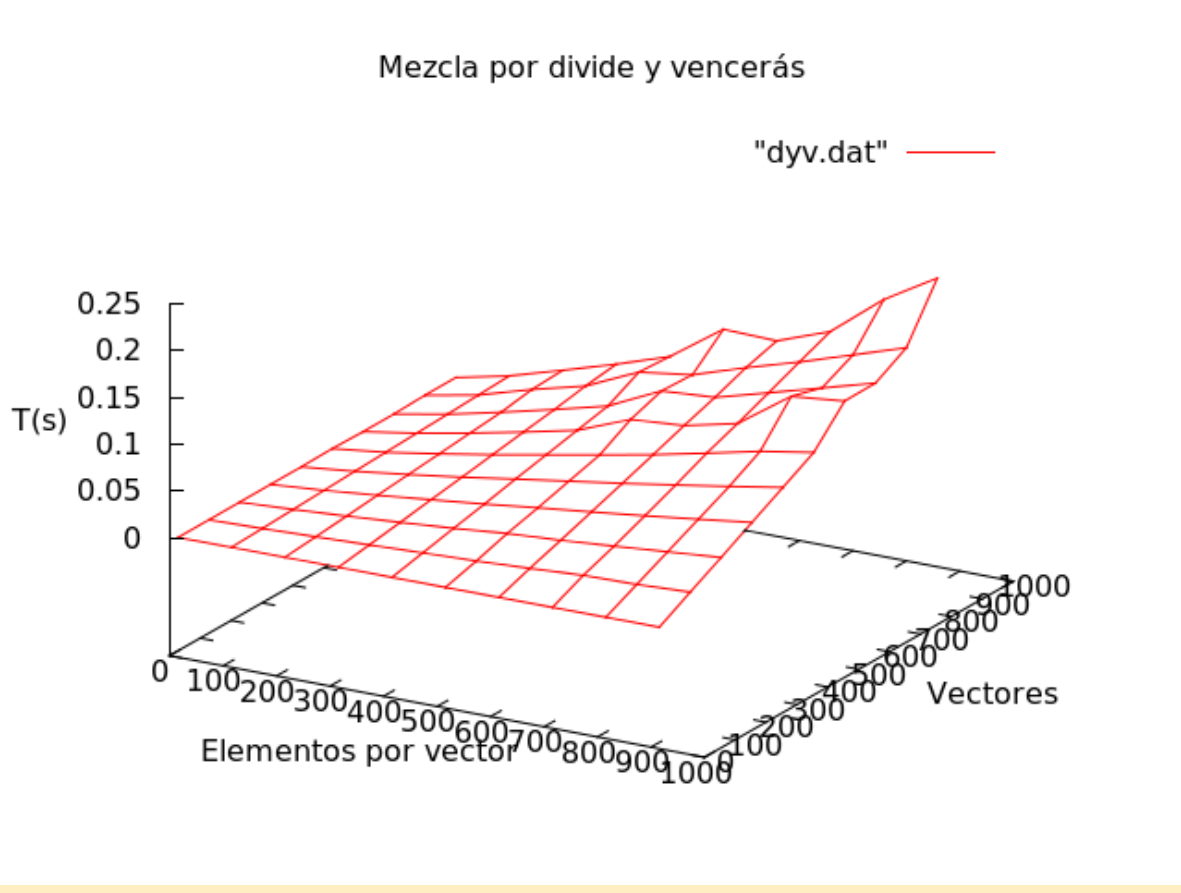


Figura : 3d divide y vencerás

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
1	3
2	1
3	2

v
3
1
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

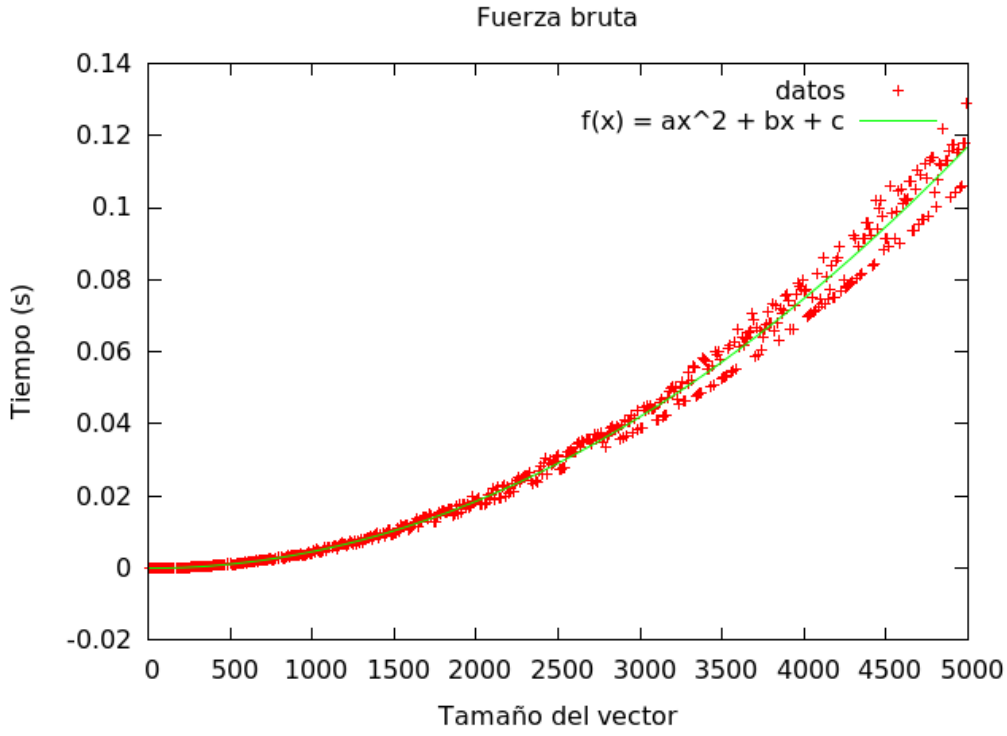
Eficiencia teórica

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

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

Ajuste

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



Divide y vencerás

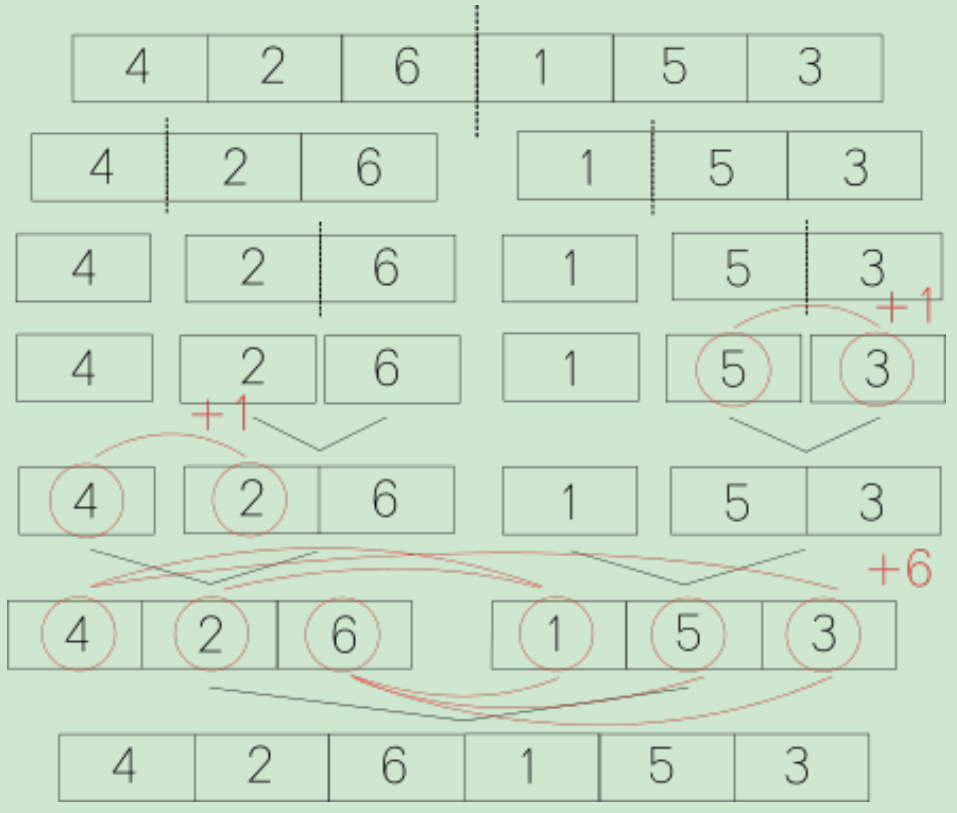
Algoritmo

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

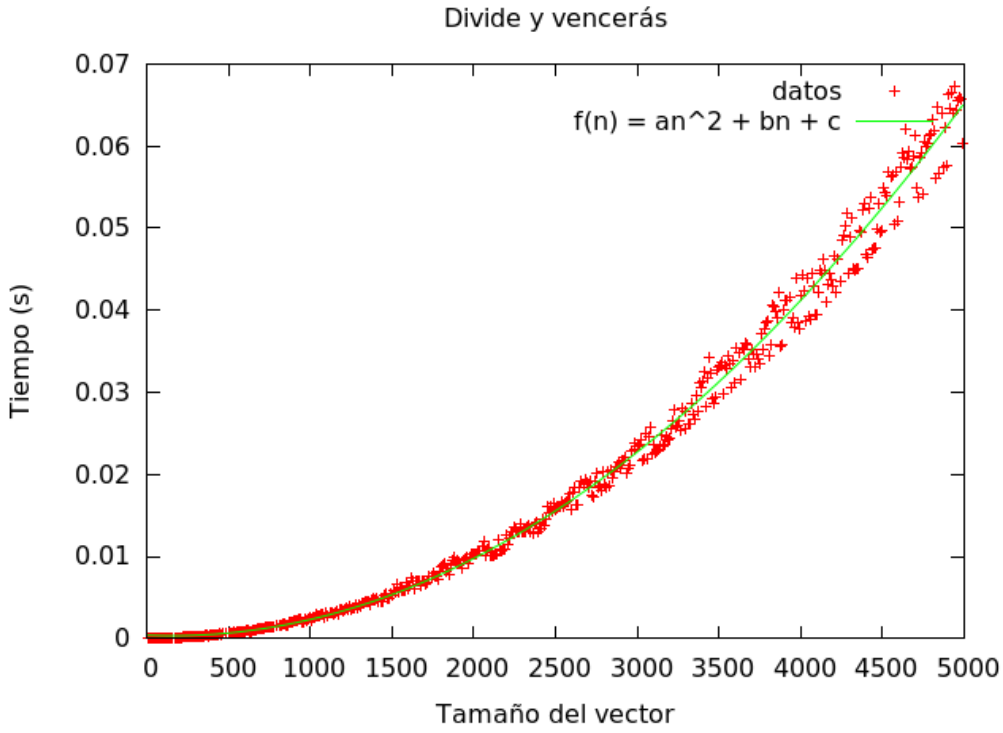
$$n = 2^k \Rightarrow k = \log_2 n$$

$$T(k) = 2^k T(1) + 2^k n$$

$$T(n) = n \cdot 0 + n^2$$

$$O(n^2)$$

Eficiencia empírica



Ajuste

$$a = 2{,}6999 \cdot 10^{-9} \; b = -8{,}47849 \cdot 10^{-7} \; c = 0{,}000424603$$

Divide y vencerás con mergesort

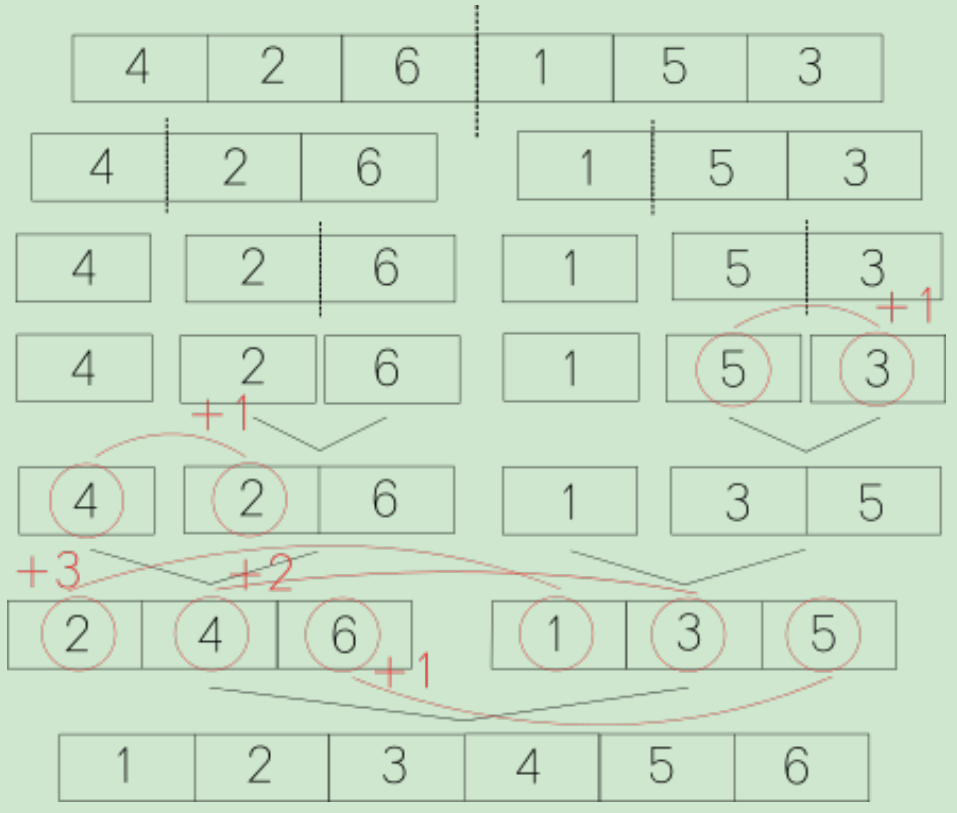
Algoritmo

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;
}
```

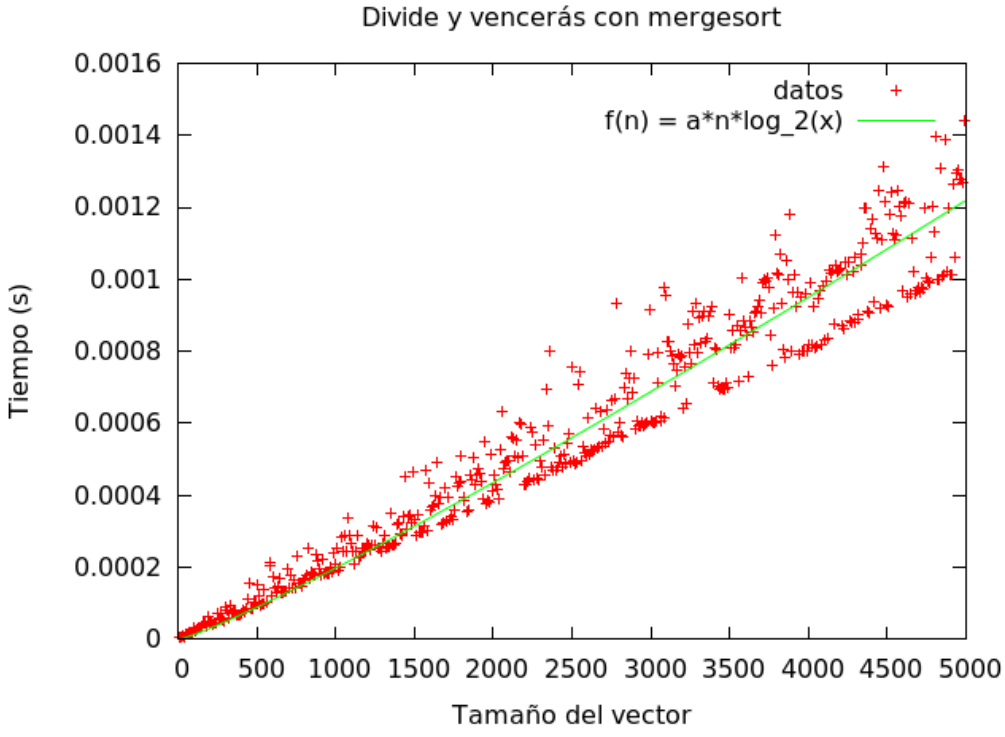
Eficiencia

Eficiencia teórica

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

Ajuste

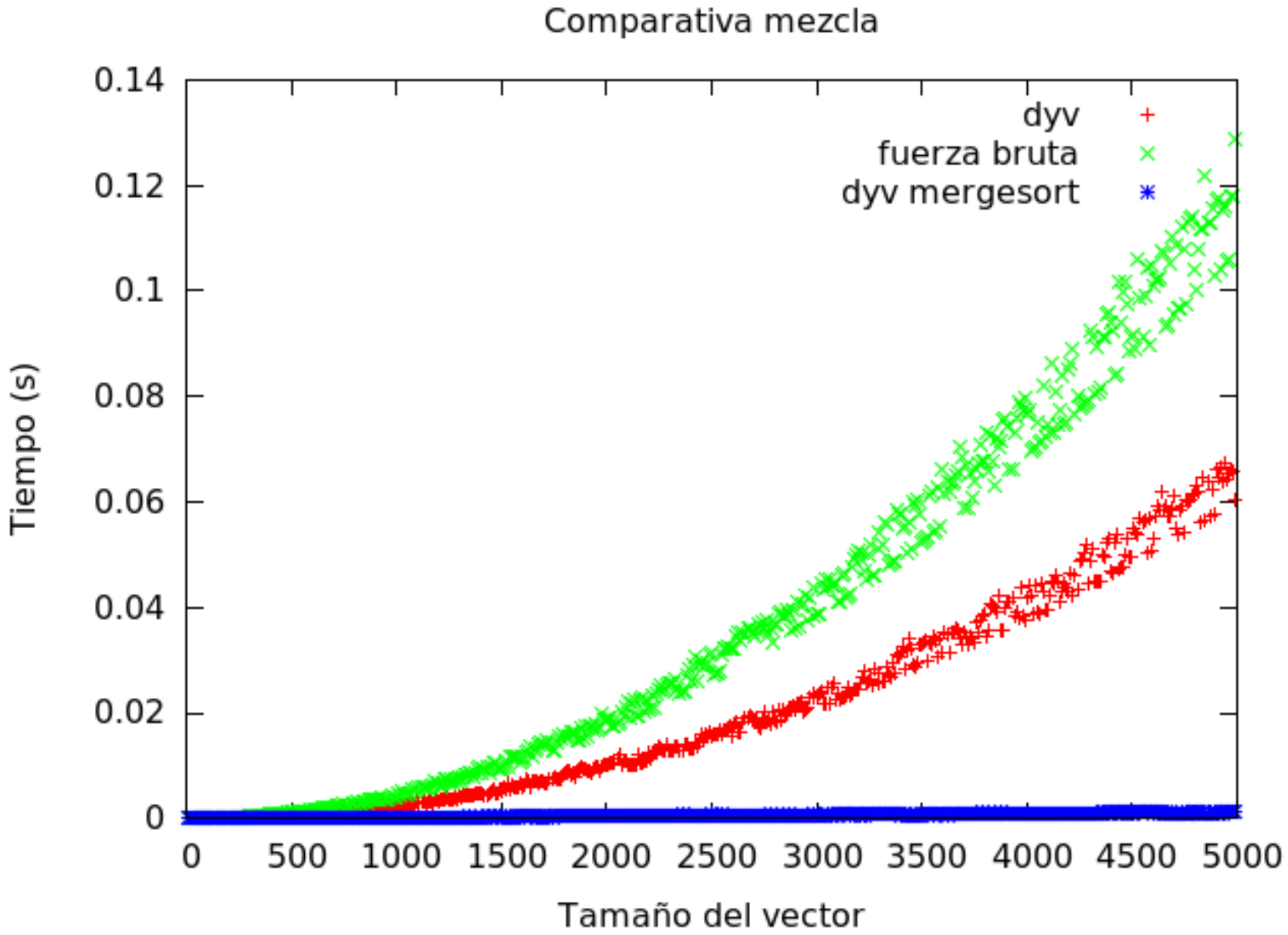
$a = 1{,}99872 \cdot 10^{-8}$



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