UNIVERSIDAD DE GRANADA

Práctica 2

Algoritmos Divide y Vencerás

Víctor José Rubia López
B3

Fecha de entrega 05/04/2020

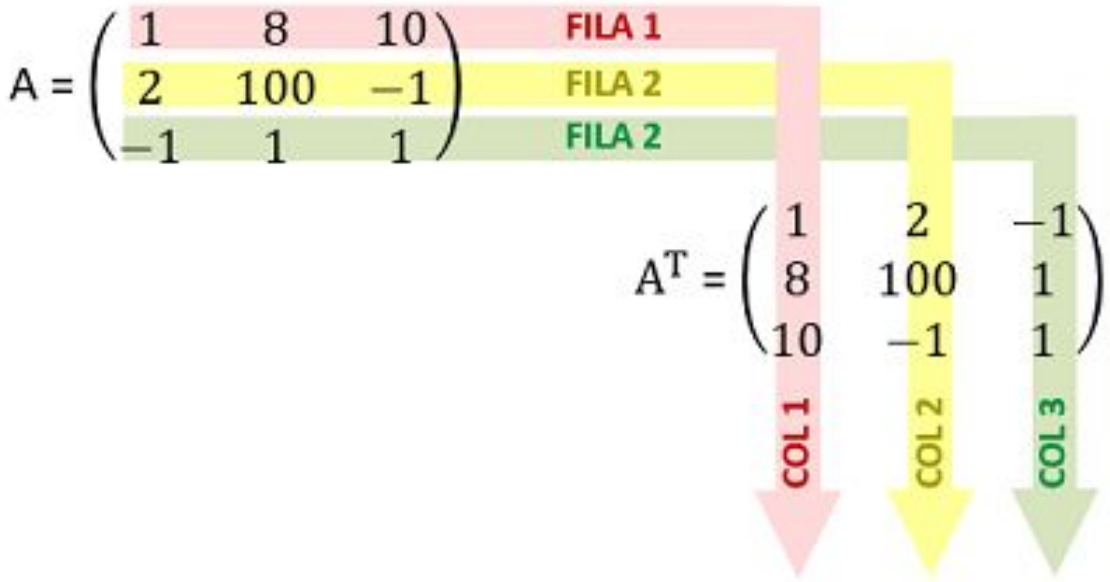
Contenido

Capítulo 1: Traspuesta de una matriz

1.1. Versión sin Divide y Vencerás

1.1.1. Eficiencia teórica

Para la realización de la traspuesta de una matriz he usado dos bucles *for* anidados, por lo que la eficiencia de este programa es $O(n^2)$ (función cuadrática). Lo podemos comprobar cuando al tomar valores cada vez más grandes, el tiempo que tarda es mayor.



1.1.2. Eficiencia empírica

Para calcularla, hemos simulado dos situaciones:

1) **Potencias de 2**: He calculado todas las potencias de dos hasta el 2^{14}. Lo he ejecutado un total de 20 veces y calculado la media de tiempo de sus ejecuciones.

TAMAÑO	TIEMPO uS
2	0,00001
4	0,0012
8	0,02
16	0,5
32	1,2
64	7,05
128	32,9
256	371,3
512	2191,6
1024	11451,7
2048	51820,1
4096	209090
8192	863436
16384	3,70E+11

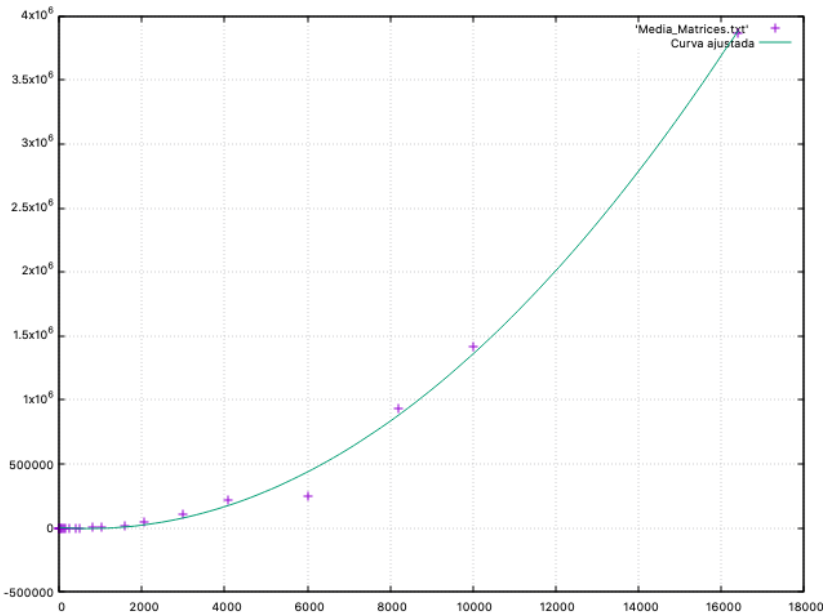
2) **Añadiendo más datos**: Hemos añadido más números hasta un total de 25 números.

TAMAÑO	Tiempo uS
2	0,00012
4	0,00026
8	0,03671
12	0,07012
16	0,95
32	3,65
64	10,85
80	34,20
128	33,20
160	305,05
256	324,20
400	1875,40
512	1850,30
800	10460,10
1024	11031,60
1600	47782,40
2048	49322,20
3000	207972,00
4092	209417,00
6000	867654,00
8192	864645,00
10000	356557000000
16384	357732000000

Para realizar la medición de los tiempos, he optado por dos vías distintas. El algoritmo de transposición de una matriz solamente funciona con matrices cuadradas de tamaño 2^n, ya que, al subdividir la matriz por la mitad nos tiene que quedar la matriz más pequeña 2 X 2. Por ello, antes de realizarlo, mi algoritmo rellena de 0s las filas y las columnas de la matriz dada hasta ser de un tamaño cuadrado potencia de 2. Por ello, he decidido hacer una pasada con valores únicamente de potencias de 2, y otra pasada con dichos valores añadiendo otros, ya que, por ejemplo, al final, una matriz de tamaño 135 se convertirá en una de tamaño 256, y quería ver si esto afectaba o no al tiempo que tarda el algoritmo en trasponerla.

1.1.3. Eficiencia híbrida

Recogiendo los datos obtenidos en la eficiencia empírica y con la eficiencia teórica $O(n^2)$ obtenemos el resultado del parámetro y la gráfica ajustada siguiente:




```
Final set of parameters
=======================
a1              = 0.0158198
a2              = -22.8006
a3              = 5067.03

Asymptotic Standard Error
=========================
+/- 0.0005024    (3.176%)
+/- 7.198        (31.57%)
+/- 1.246e+04    (245.9%)
```

1.2. Versión Divide y Vencerás

1.2.1. Eficiencia teórica

De la misma forma que el caso del programa sin divide y vencerás, la función tendrá un máximo de dos bucles *for* anidados, por tanto, la eficiencia sigue siendo $O(n^2)$.

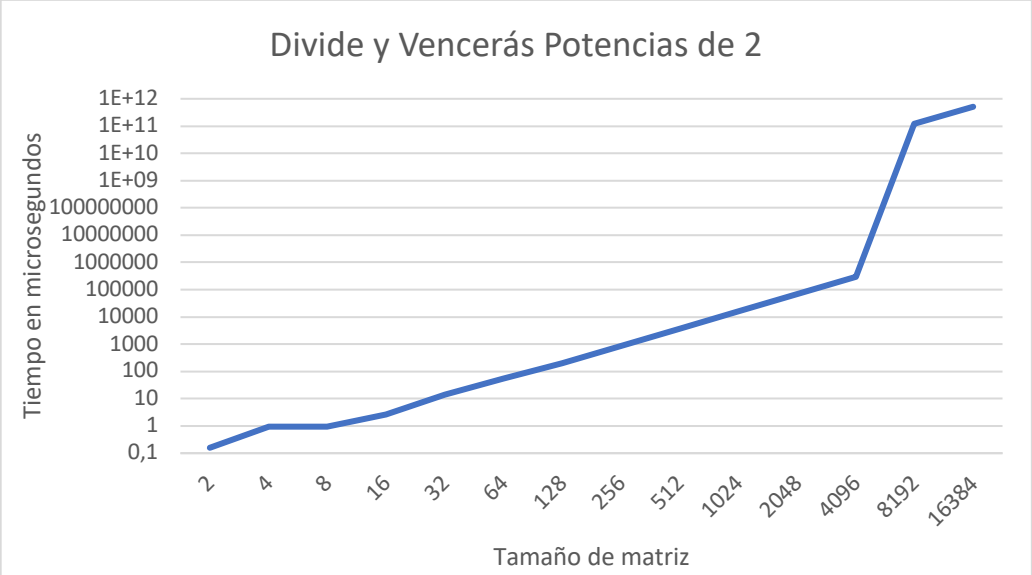
Sin embargo, para este caso, el algoritmo divide y vencerás necesitará de más cambios para poder llegar a la solución, por lo tanto, aunque tenga la misma eficiencia, el tiempo que tardará con los mismos valores será mayor.

1.2.2. Eficiencia empírica

Tomo exactamente los mismos valores tanto para las potencias de 2, como para añadir números entre medias. El tiempo que tarda en ejecutar los números es casi el doble que en el caso sin Divide y Vencerás.

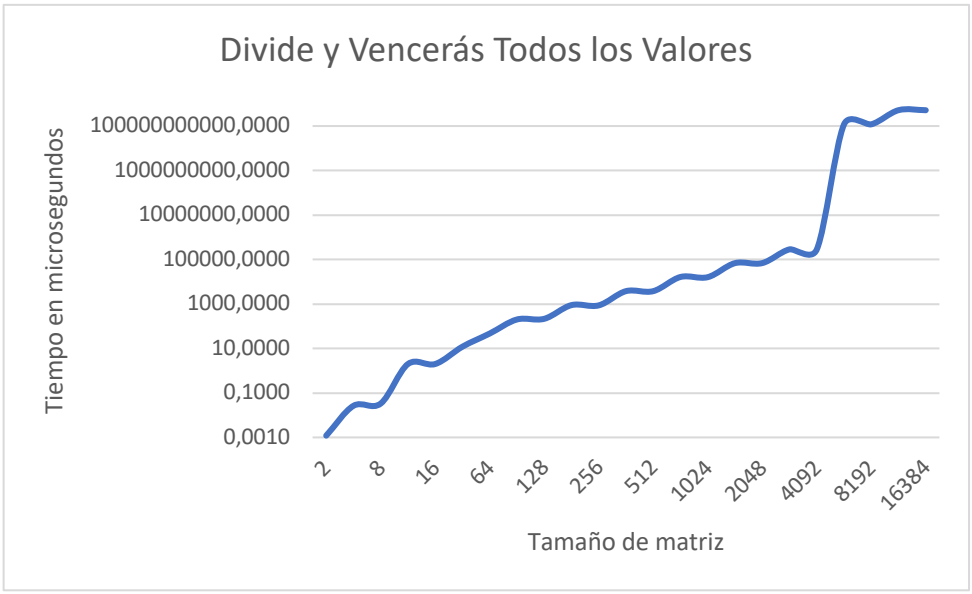
1) **Potencias de 2**

TAMAÑO	TIEMPO EN uS
2	0,158
4	0,95
8	0,95
16	2,65
32	14,05
64	53,4
128	199,35
256	861,35
512	3782,8
1024	16132,7
2048	67816,4
4096	288767
8192	1,22E+11
16384	5,11E+11

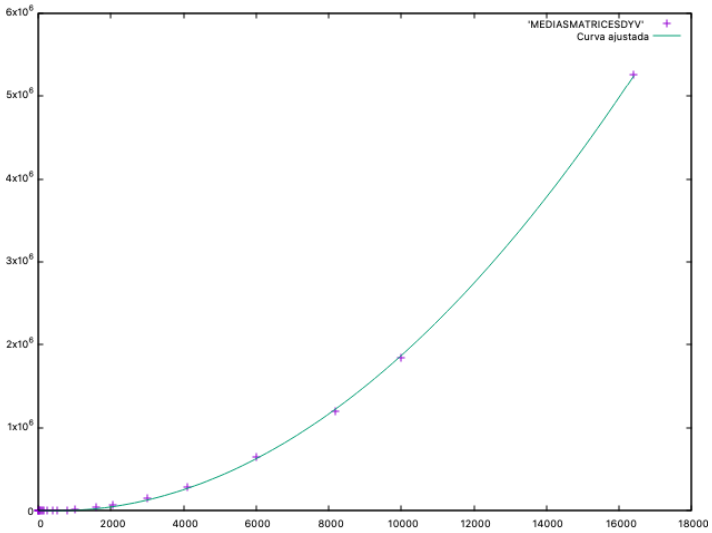


2) **Números añadidos**

TAMAÑO	TIEMPO EN uS
2	0,0012
4	0,0264
8	0,0333
12	2,0000
16	2,0000
32	12,0333
64	46,9333
80	200,9670
128	215,3330
160	888,6330
256	865,1670
400	3776,2300
512	3772,0700
800	16368,1000
1024	16053,8000
1600	68606,7000
2048	68931,0000
3000	285146,0000
4092	280990,0000
6000	1185240000000,0000
8192	1193780000000,0000
10000	5162840000000,0000
16384	5148570000000,0000



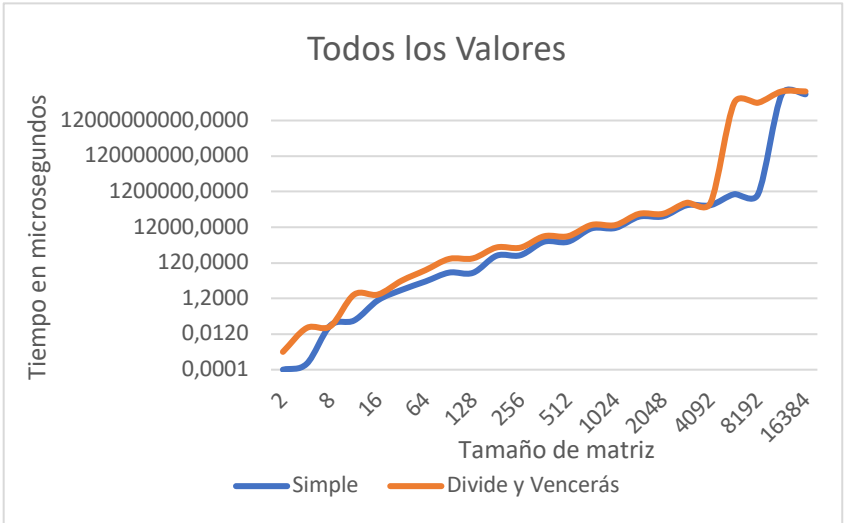
1.2.3. Eficiencia híbrida




```
Final set of parameters
=======================
a1              = 0.0207494
a2              = -20.8997
a3              = 7037.15

Asymptotic Standard Error
=========================
+/- 0.0001659    (0.7994%)
+/- 2.376        (11.37%)
+/- 4114         (58.46%)
```

1.3. Comparando versiones



Podemos observar que la versión Divide y Vencerás está la mayor parte de los tamaños en tiempo por encima de la versión que no aplica esta técnica.

Capítulo 2: Eliminar elementos repetidos

2.1. Versión sin Divide y Vencerás

2.1.1 Eficiencia teórica

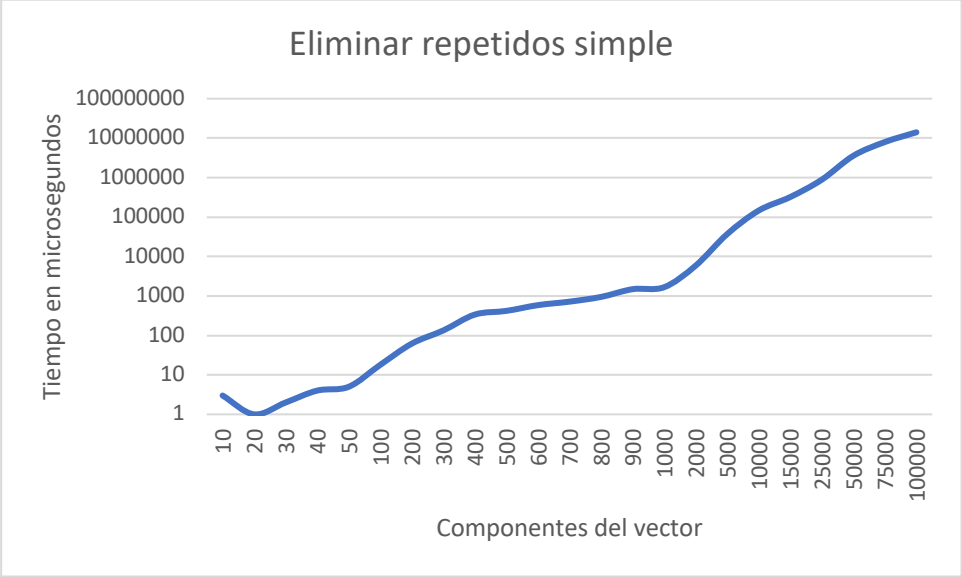
```
1 void eliminarRepSimple( vector<int> &v ){
2
3     for(int i = 0; i < v.size(); i++){
4         for(int j=i+1; j < v.size(); j++){
5             if(v[i] == v[j]){
6                 v.erase(v.begin() + j);
7                 j--;
8             }
9
10 }
```

Como la función está compuesta por dos bucles *for*, siendo el primero desde 0 hasta el tamaño del vector, y el segundo recorrería el vector desde la posición del bucle *for* anterior hasta el final del vector. El condicional *if* es O(1), por lo que la combinación de todos estos sería $O(n^2)$.

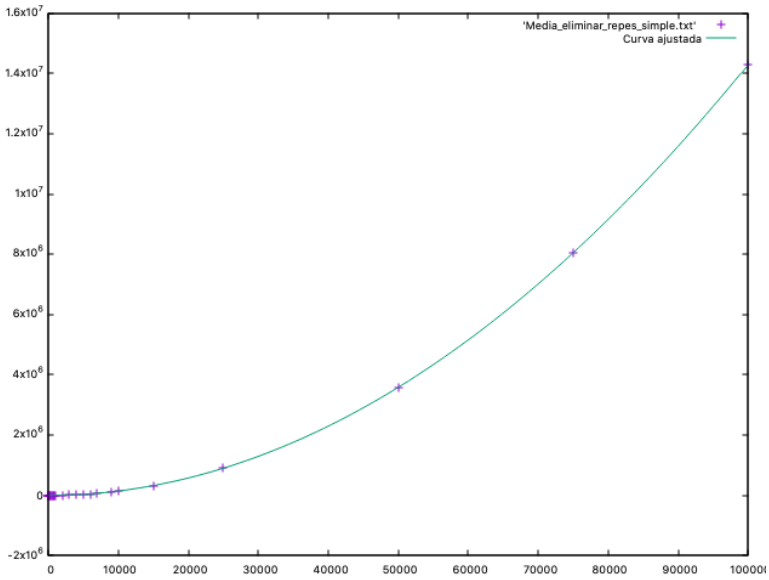
2.1.2 Eficiencia empírica

Para la eficiencia empírica, he tomado un número total de 22 valores y hasta un valor máximo de 100000000.

TAMAÑO	TIEMPO EN uS
10	3
20	1
30	2
40	4
50	5
100	18
200	62
300	134
400	338
500	419
600	586
700	718
800	946
900	1487
1000	1667
2000	5872
5000	37011
10000	146159
15000	321699
25000	885378
50000	3569453
75000	7964171
100000	14068549



2.1.3 Eficiencia híbrida




```
Final set of parameters
=======================
a1              = 0.000957138
a2              = 2.12414
a3              = -5445.65

Asymptotic Standard Error
=========================
+/- 5.816e-06    (0.6077%)
+/- 0.5271       (24.82%)
+/- 4618         (84.8%)
```

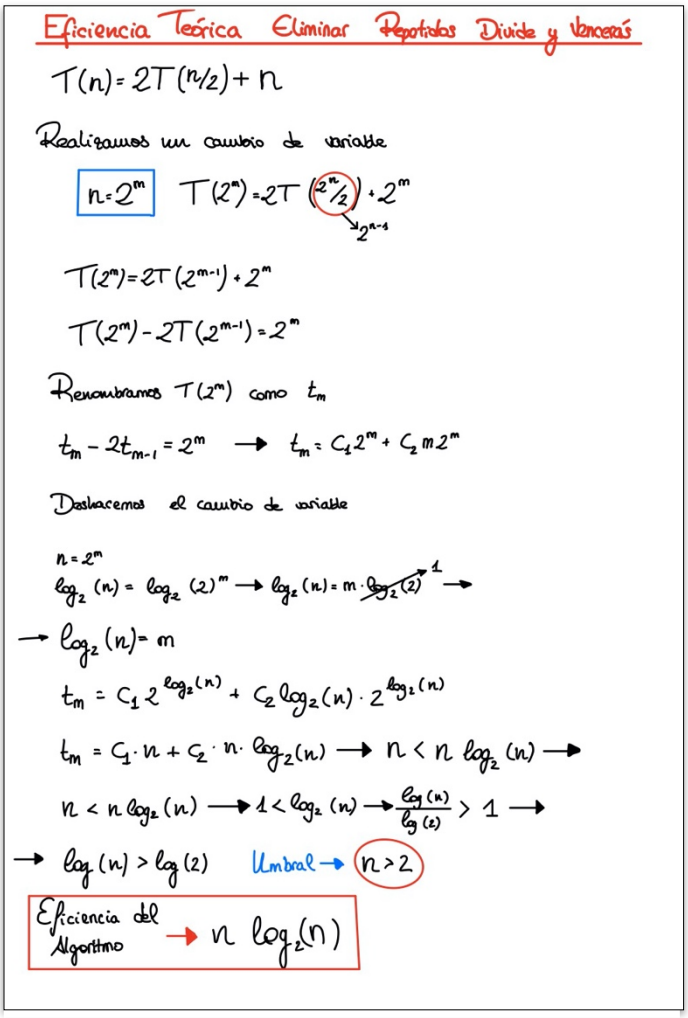
2.2. Versión Divide y Vencerás

2.2.1 Eficiencia teórica

```
1 void eliminarRepDV(vector<int> &v) {
2
3     MergeSort(v, 0, v.size()-1);
4
5     for (int i = 0; i < v.size() - 1; ++i) {
6         if (v[i] == v[i+1]) {
7             v.erase(v.begin()+i+1);
8             i--;
9         }
10    }
11 }
```

De forma distinta a la versión sin divide y vencerás, al usar el algoritmo *MergeSort*, obtendremos una eficiencia distinta, O(n log n).

Implementando este algoritmo, conseguiré una mejora sustancial de la eficiencia, a través de la ordenación del vector usando el algoritmo *MergeSort*, basado en la técnica de Divide y Vencerás.

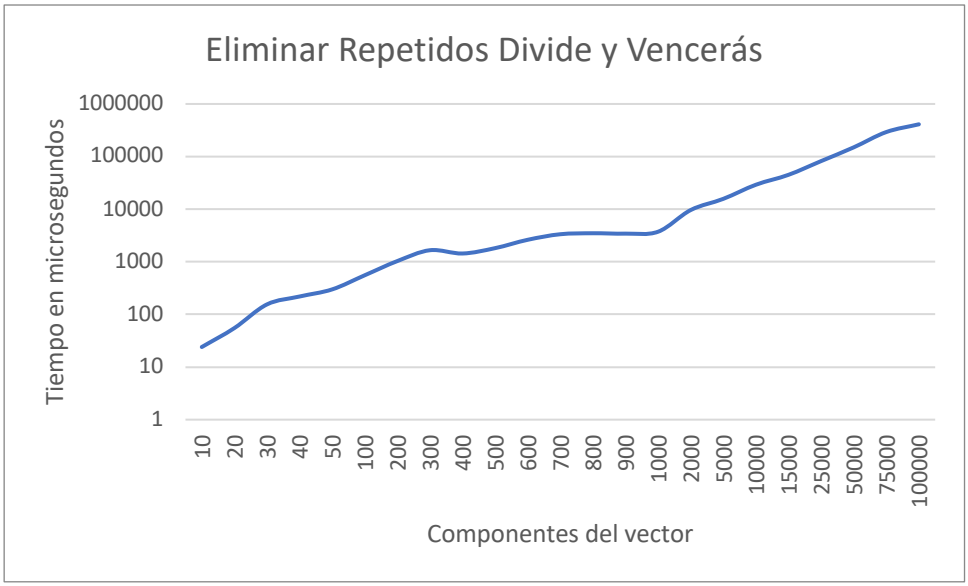


Realizando el cálculo de la recursividad comprobamos que, efectivamente la eficiencia es O(n log n), distinta a la eficiencia en la versión sin Divide y Vencerás. Además, en este caso, dividimos el vector en dos y llamamos a la función recursivamente. De esta manera, obtenemos un algoritmo bastante más eficiente. Aquí tenemos un ejemplo de cómo la técnica Divide y Vencerás es adecuada para este tipo de problemas.

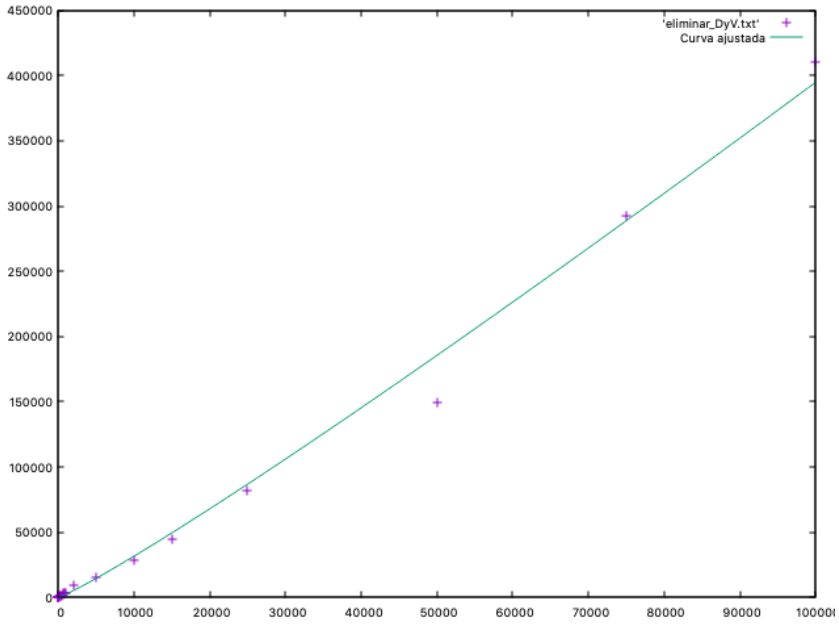
2.2.2 Eficiencia empírica

De la misma forma que en la versión sin Divide y Vencerás, hemos tomado 22 valores.

TAMAÑO	TIEMPO EN uS
10	24
20	55
30	155
40	219
50	299
100	555
200	1033
300	1660
400	1438
500	1819
600	2621
700	3342
800	3489
900	3431
1000	3732
2000	9634
5000	15724
10000	29148
15000	45010
25000	82026
50000	149619
75000	292492
100000	410560



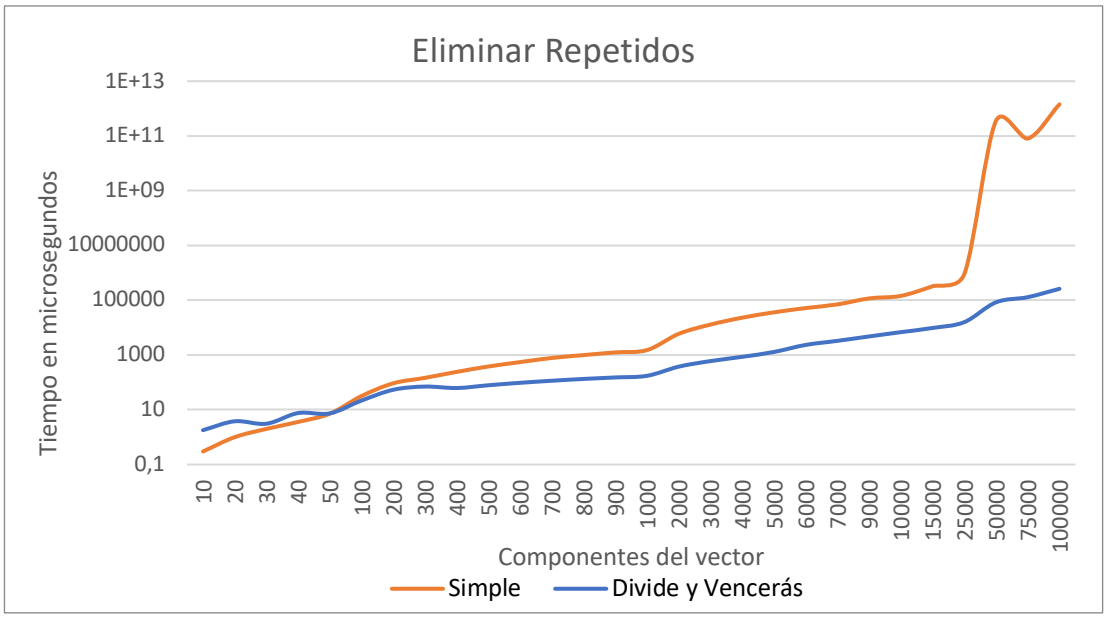
2.2.3 Eficiencia híbrida




```
Final set of parameters
=======================
a0              = 0.789599

Asymptotic Standard Error
=========================
+/- 0.01275      (1.615%)
```

2.3. Comparando versiones



Este es un ejemplo de cómo usando la técnica divide y vencerás nos ayuda en la ejecución de este tipo de problema. Observamos que a partir de 50 componentes, usar la versión Divide y Vencerás es mucho mas eficiente que la que no lo implementa, sin embargo, para tamaños inferiores a 50, comprobamos que es mucho mejor usar la técnica simple.

Capítulo 3: Caso de ejecución

3.1. Vector

Para generar los tiempos y las medidas hemos usado el formato de la primera práctica. Por tanto la forma de ejecución será la siguiente:

```
$ ./archivo_ejecutable salidatxt semilla valor1 valor2 … valor n
```

```
victorjoserubialopez@Mac-mini-de-Victor Individual % g++ -std=c++11 eliminar_repetidossimple.cpp -o eliminar_repetid
ossimple
victorjoserubialopez@Mac-mini-de-Victor Individual % ./eliminar_repetidossimple datos_simple.txt 1 15 1500 15000
        Tiempo de ejec- (us): 3 para tam. caso 15
        Tiempo de ejec- (us): 3139 para tam. caso 1500
        Tiempo de ejec- (us): 324705 para tam. caso 15000

victorjoserubialopez@Mac-mini-de-Victor Individual % g++ -std=c++11 eliminar_repetidosDyV.cpp -o eliminar_repetidosD
yV
victorjoserubialopez@Mac-mini-de-Victor Individual % ./eliminar_repetidosDyV datos_DyV.txt 1 15 1500 15000
        Tiempo de ejec- (us): 12 para tam. caso 15
        Tiempo de ejec- (us): 284 para tam. caso 1500
        Tiempo de ejec- (us): 5308 para tam. caso 15000
```

En este ejemplo hemos puesto que solo haga 2 repeticiones de cada valor, pero para el desarrollo de los datos y gráficas, al igual que de las eficiencias hemos puesto 20 repeticiones.

Capítulo 4: Conclusión

En el problema de la matriz traspuesta, los dos algoritmos tienen la misma eficiencia teórica. Empíricamente, usando la técnica Divide y Vencerás tarda más debido a que "pierde" más tiempo al tener que subdividir la matriz. Este es uno de los casos en los que se observa que el uso de esta técnica es contraproducente.

En el problema individual (grupal) que me han asignado, eliminar los elementos repetidos de un vector, he comprobado cómo Divide y Vencerás acelera la ejecución del algoritmo. La función por tanto tiene un orden de eficiencia menor al simple $O(\mathrm{n}^2) > O(\mathrm{n}\ \log\ \mathrm{n})$ y por ello es mejor usar esta técnica.