PRÁCTICA 2: DIVIDE Y VENCERÁS

Ahmed El Moukhtari Koubaa
Damián Marín Fernández
Jesús Martín Zorrilla
Eduardo Segura Richart

ÍNDICE

DESCRPCIÓN DEL PROBLEMA

Resolución de dos problemas:

- Hallar la traspuesta de una matriz
- Encontrar elementos repetidos en un vector

Con los siguientes algoritmos:

- Fuerza Bruta (fb)
- Divide y Vencerás (dv)

Análisis de la eficiencia:

- Eficiencia Teórica → Análisis código
- Eficiencia Empírica → Ejecución con varios tamaños
- Eficiencia Híbrida → Basada en las anteriores

DESCRPCIÓN DEL PROBLEMA

```
1  void TrasponerMatriz(int ** m, const int dim){
2    for (int i = 0; i < dim; ++i){
3      for (int j = i+1; j < dim; ++j){
4        int intercambia = m[i][j];
5        m[i][j] = m[j][i];
6        m[j][i] = intercambia;
7      }
8    }
9  }
```

Bucle interno

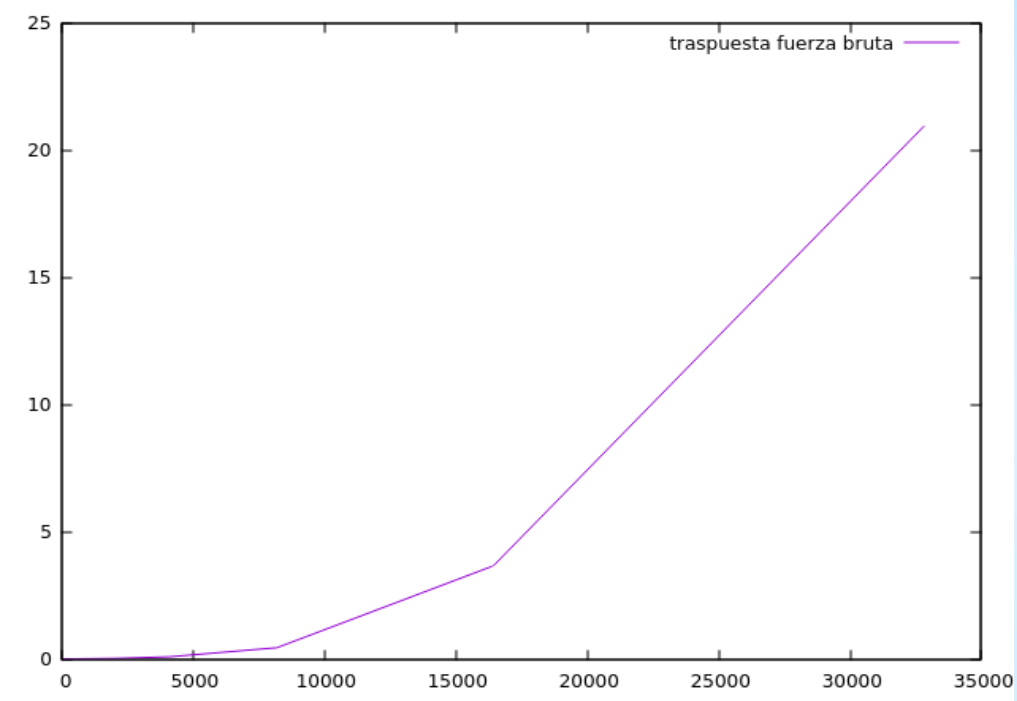
- 3 operaciones de O(1)
- Desde j=i+1 hasta j=dim-1
- n-1 repeticiones

Fórmula →

$$\sum_{i=0}^{n-1} 3i = \frac{3n \times n}{2} = \frac{3}{2}n^2 \in O(n^2)$$

2. MATRIZ TRASPUESTA FB – EFICIENCIA TEÓRICA

Tamaño	tiempo	Opt
4	1.6e-06	1.6e-06
8	1.7e-06	1.7e-06
16	2.2e-06	1.8e-06
32	4.1e-06	2.1e-06
64	1.03e-05	3.8e-06
128	3.7e-05	9.4e-06
256	0.0001402	3.13e-05
512	0.0006555	0.0001807
1024	0.0038022	0.001129
2048	0.0168626	0.0083153
4096	0.0799885	0.0562152
8192	0.437094	0.31406
16384	3.6398	2.1247
32768	20.9105	16.0097



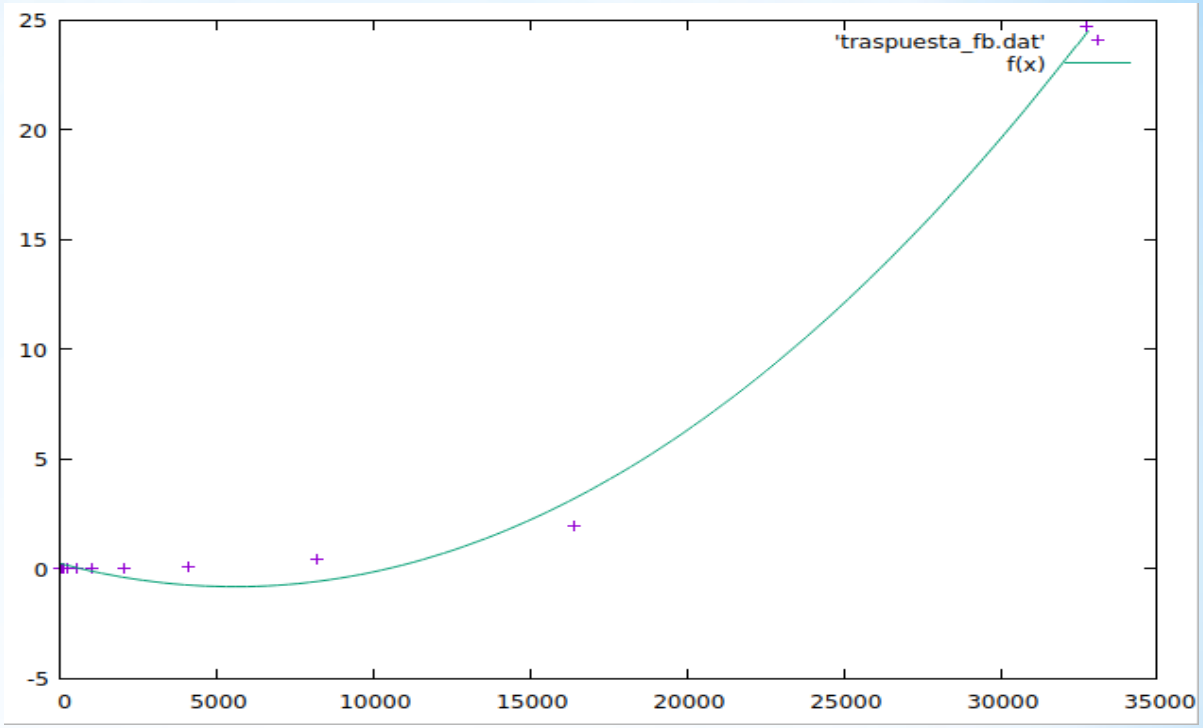
MATRIZ TRASPUESTA FB - EFICIENCIA EMPÍRICA

f(x)=ax2 + bx + c

a = 3,41959e-08

b = - 0,000380884

c = 0,225173



MATRIZ TRASPUESTA FB - EFICIENCIA HÍBRIDA

Para 2 tamaños, 4 y 8:

```
Estado actual de la matriz previa trasposicion
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Estado posterior de la matriz
1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16
```

```
Estado actual de la matriz previa trasposicion
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64

Estado posterior de la matriz
1 9 17 25 33 41 49 57
2 10 18 26 34 42 50 58
3 11 19 27 35 43 51 59
4 12 20 28 36 44 52 60
5 13 21 29 37 45 53 61
6 14 22 30 38 46 54 62
7 15 23 31 39 47 55 63
8 16 24 32 40 48 56 64
```

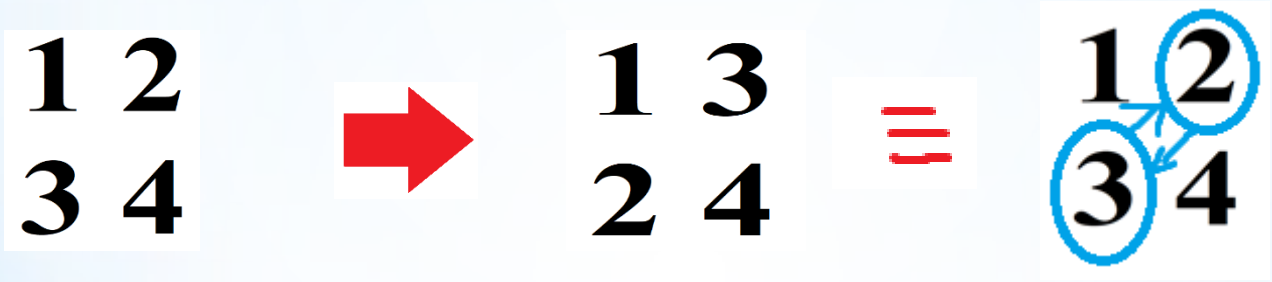
MATRIZ TRASPUESTA FB - EFICIENCIA CASO DE EJECUCIÓN

Descripción del algoritmo:

1.- Dividir en 4 submatrices la matriz original.

2.- Intercambiar las submatrices diagonales (2 y 3).

3.- Repetir desde el paso 1 hasta llegar al caso base (matriz de dimensión 2).



3. MATRIZ TRASPUESTA DyV

```
void TrasponerDV(int ** m, const int fi, const int ff, const int ci, const int cf){
  if (cf-ci > 2){   // Si la dimension de la matriz es mayor que dos
    // Guardar filas y columnas de inicio y final de cada sumbatriz
    int fi1 = fi, ff1 = (fi+ff)/2, ci1 = ci,   cf1 = (cf+ci)/2;  // sb1) Comienza en
    int fi2 = fi, ff2 = ff1,       ci2 = cf1, cf2 = cf;          // sb2) Coincice co
    int fi3 = ff1, ff3 = ff,       ci3 = ci,   cf3 = cf1;        // sb3) Comienza en
    int fi4 = ff1, ff4 = ff,       ci4 = cf1, cf4 = cf;          // sb$) Coincide co

    TrasponerDV(m, fi1, ff1, ci1, cf1);   // Submatriz 1
    TrasponerDV(m, fi2, ff2, ci2, cf2);   // Submatriz 2
    TrasponerDV(m, fi3, ff3, ci3, cf3);   // Submatriz 3
    TrasponerDV(m, fi4, ff4, ci4, cf4);   // Submatriz 4


    const int tam = (cf-ci)/2;              // Numero de elementos a mover en una f
    const size_t tamB = tam*sizeof(int);    // Tamaño de la fila a mover en Bytes
    int * fila_actual = new int[tam];

    // Intercambiar submatrices 2 y 3
    for (int i = 0; i < tam; ++i){                                  // Recorrer todas las fil
      memcpy(fila_actual,      m[i + fi2] + ci2, tamB);             // Mover todos los element
      memcpy(m[i + fi2] + ci2, m[fi3 + i] + ci3, tamB);             // Mover todos los element
      memcpy(m[fi3 + i] + ci3, fila_actual,      tamB);             // Mover todos los element
    }

    delete [] fila_actual;                                          // Liberar espacio reserv
  }
  else {    // Hemos llegado al caso base matriz de dimension 2
    int intercambia = m[fi][cf-1];  // Operaciones elementales básicas
    m[fi][cf-1]     = m[ff-1][ci];  // despreciables
    m[ff-1][ci]     = intercambia;  // Poner el primer elemento en donde estaba el
  }
}
```

Dividir el tamaño en 4

4 llamadas recursivas con 1/4 del tamaño

Unir las particiones (lineal)

Operaciones elementales básicas despreciables

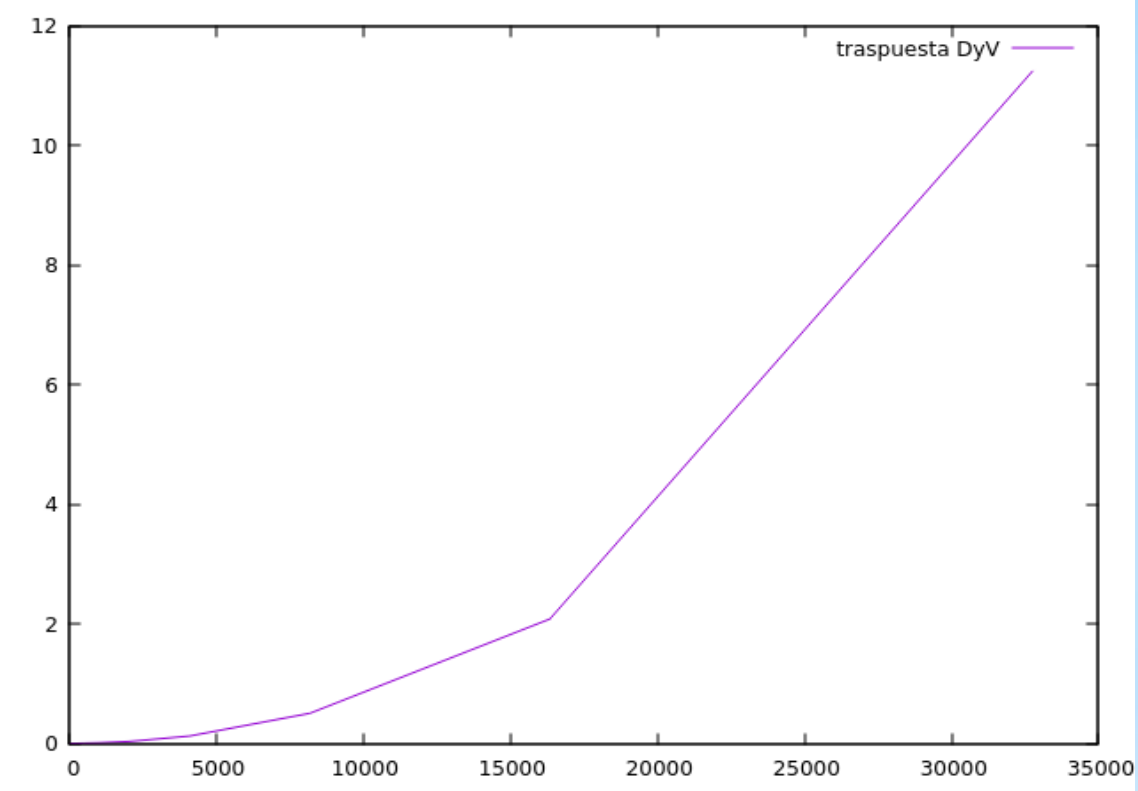
CÓDIGO

T(n) = 4*T(n/4) + n

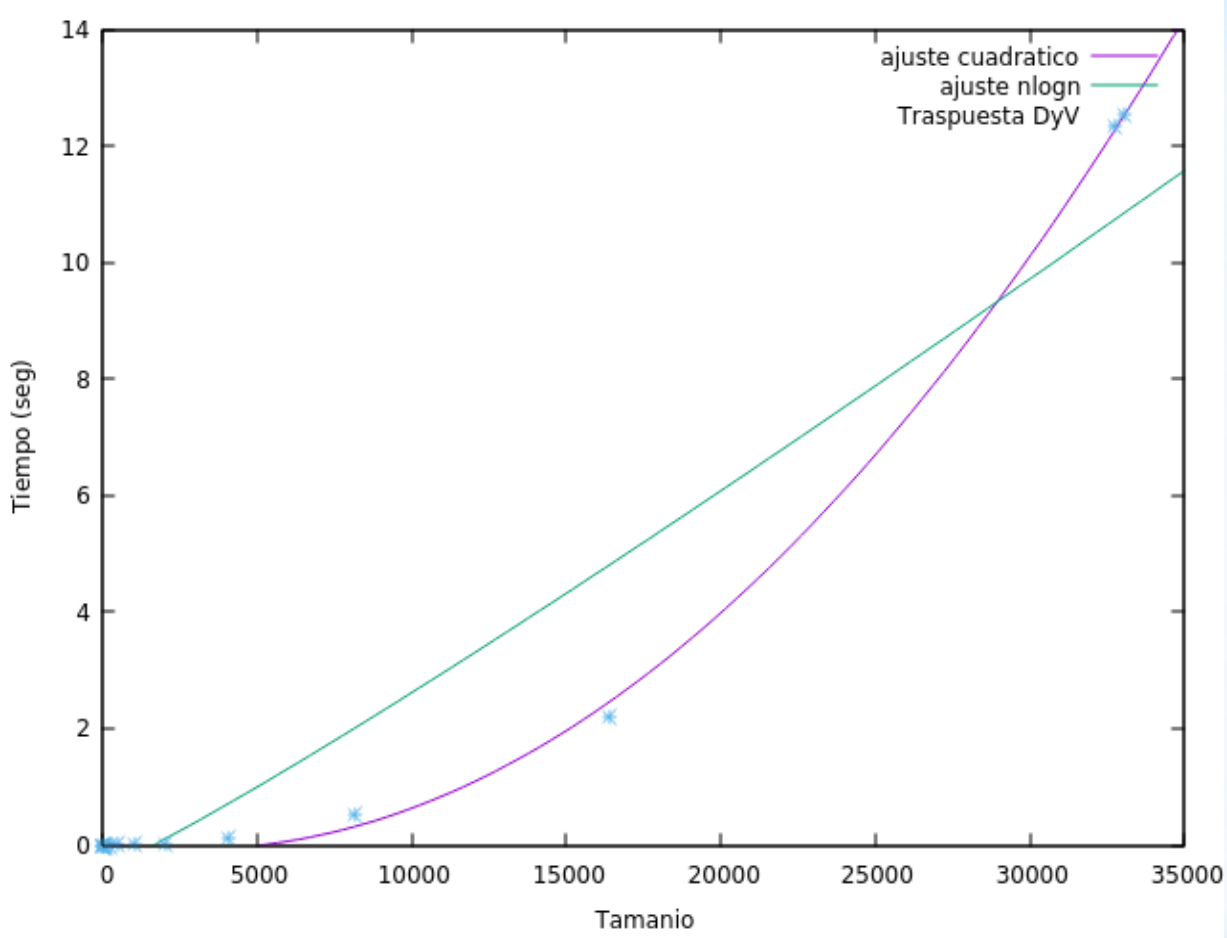
- Hacemos el siguiente cambio de variable para ello ▫ n = 4^k
- Nos queda: T(4^k) = T(4^(k-1)) + 4^k
- Expresamos en función de k: tk = tk-1 + 4^k
- Resolvemos con la ecuación característica (x-4)*(x-4)
- Nos queda c1*4^k + c2*k*4^k
- Deshacemos el cambio c1*n + c2*n*log(n)
- Luego este algoritmo es del orden O(n*log(n)+n)
- Expresamos la eficiencia sin valores despreciables: **O(n*log(n))**

3. MATRIZ TRASPUESTA DyV – EFICIENCIA TEÓRICA

Tamaño	tiempo	Opt
4	3.1e-06	3.3e-06
8	3.4e-06	3.3e-06
16	4.9e-06	6.2e-06
32	1e-05	1.05e-05
64	2.93e-05	2.05e-05
128	0.0001078	7.21e-05
256	0.0004066	0.0002814
512	0.0017361	0.0013057
1024	0.0073086	0.0063488
2048	0.0293867	0.0239823
4096	0.122589	0.0976504
8192	0.502631	0.42082
16384	2.0873	1.73995
32768	11.2342	9.70815



3. MATRIZ TRASPUESTA DyV - EFICIENCIA EMPÍRICA



f(n) = 0.000227427*log(n)*n - 0.000227427*n + 0.000227427

3. MATRIZ TRASPUESTA DyV - EFICIENCIA HÍBRIDA

Dos ejecuciones, para tamaños 4 y 8:

```
Matriz antes:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
4 6.2e-06

Después trasposicion:
1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16
```

```
Matriz antes:
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
8 4.4e-06

Después trasposicion:
1 9 17 25 33 41 49 57
2 10 18 26 34 42 50 58
3 11 19 27 35 43 51 59
4 12 20 28 36 44 52 60
5 13 21 29 37 45 53 61
6 14 22 30 38 46 54 62
7 15 23 31 39 47 55 63
8 16 24 32 40 48 56 64
```

3. MATRIZ TRASPUESTA
DyV – CASO DE EJECUCIÓN

```
//para quitar repetidos en el vector generado
int aux;
    for(int i=0; i<n; i++){
        for(int j = i+1; j < n; j++){
            if(T[i] == T[j]){
                aux = j;
                while(aux < n){
                    T[aux] = T[aux+1];
                    ++aux;
                }
                --n;
                --j;
            }
        }
    }
```

2 bucles anidados

Bucle Interno:

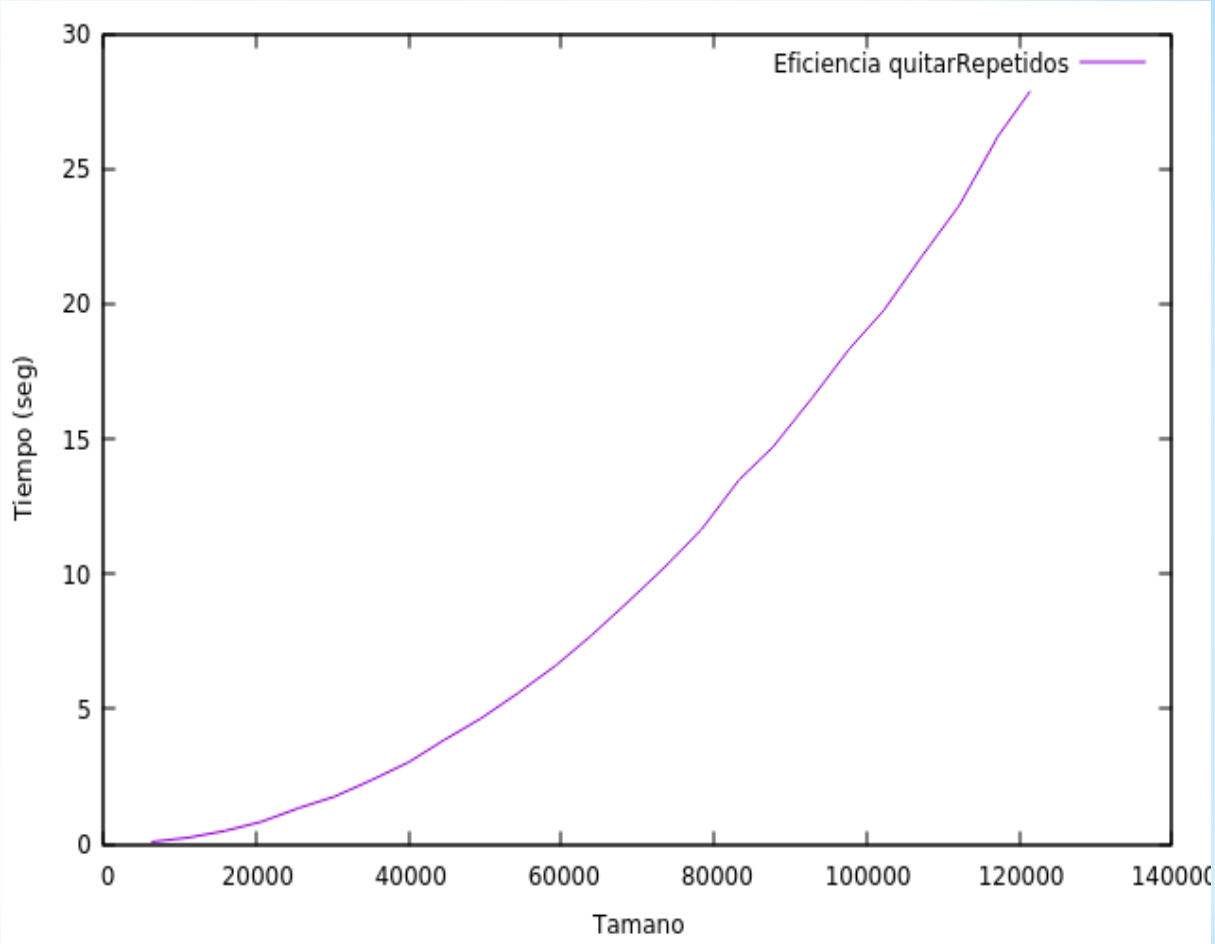
- Comparación
- Desplazamiento n-k elementos restantes hacia la izquierda.

Peor caso: **O(n^2)**

Vector vacío: **O(1)**

4. ELIMINAR REPETIDOS FB - EFICIENCIA TEÓRICA

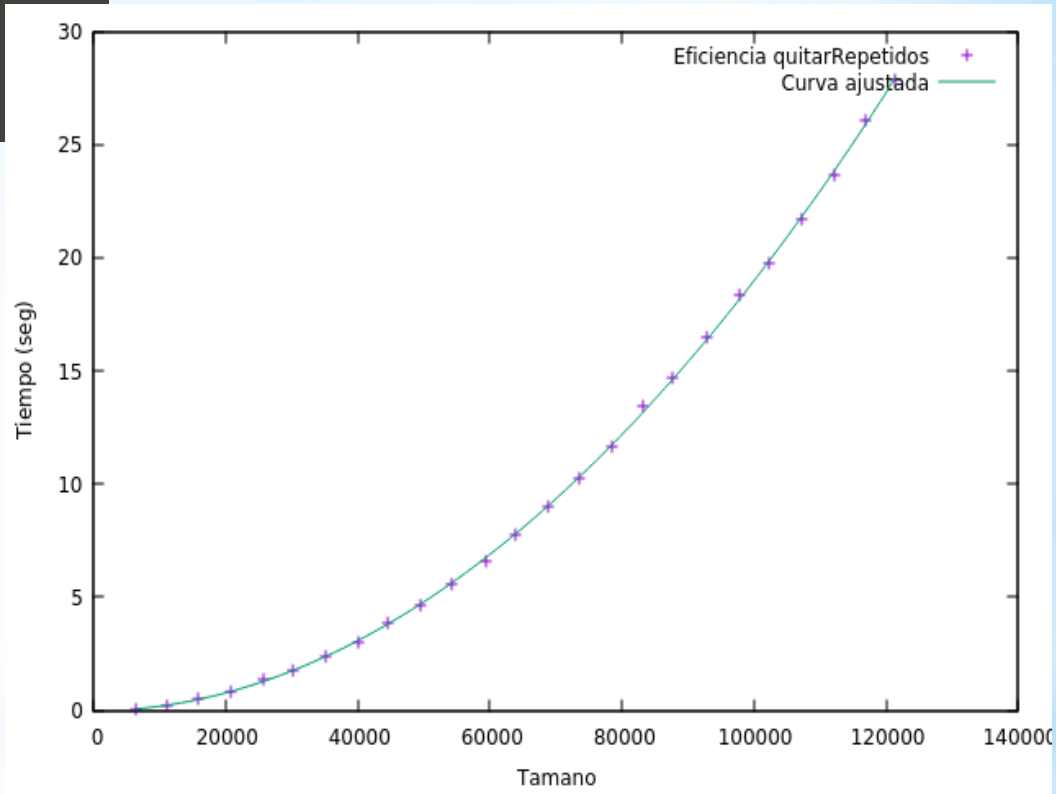
Tamaño	tiempo
6314	0.077351
11154	0.237305
15938	0.484241
20776	0.828906
25648	1.3297
30302	1.7567
35197	2.37045
40079	3.04086
44623	3.84492
49434	4.63525
54266	5.57575
59328	6.61627
63970	7.72621
68845	8.98567
73551	10.2458
78363	11.6415
83296	13.487
87713	14.6903
92846	16.5195
97765	18.3429
102268	19.7659
107122	21.7133
112128	23.6367
116916	26.0533
121436	27.8372



4. ELIMINAR REPETIDOS FB - EFICIENCIA EMPÍRICA

```
Final set of parameters            Asymptotic Sta
=======================            ==============
a0              = 1.87415e-09      +/- 2.064e-11
a1              = 2.8429e-06       +/- 2.716e-06
a2              = -0.0410974       +/- 0.07559

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.972  1.000
a2              0.790 -0.894  1.000
```



4. ELIMINAR REPETIDOS FB – EFICIENCIA HÍBRIDA

Dos ejemplos, uno de tamaño 10 y otro de tamaño 15:

```
jesusmz@jmz:~/Documentos/Universidad/Algoritmica/Practica2$ ./generaDuplicados 10
Vector: 9 2 5 2 6 2 6 7 6 6
Vector con repetidos quitados: 9 2 5 6 7
jesusmz@jmz:~/Documentos/Universidad/Algoritmica/Practica2$ ./generaDuplicados 15
Vector: 5 7 8 1 9 8 0 2 11 14 0 3 3 4 5
Vector con repetidos quitados: 5 7 8 1 9 0 2 11 14 3 4
jesusmz@jmz:~/Documentos/Universidad/Algoritmica/Practica2$
```

4. ELIMINAR REPETIDOS FB - CASO DE EJECUCIÓN

Algoritmo: Eliminar todos los repetidos de UN MISMO vector

No es posible aplicar DyV
ALTERNATIVA

```
void EliminaRepetidosDV(int * v, int & n){
  Ordena(v, 0, n);                    // Ordenar elementos
  EliminaRepetidosOrdenado(v,n);      // Eliminar elemenos repetidos ya ordenados
}
```

5. ELIMINAR REPETIDOS DyV - ANÁLISIS

Algoritmo de ordenación → MERGESORT → **O(n*log(n))**

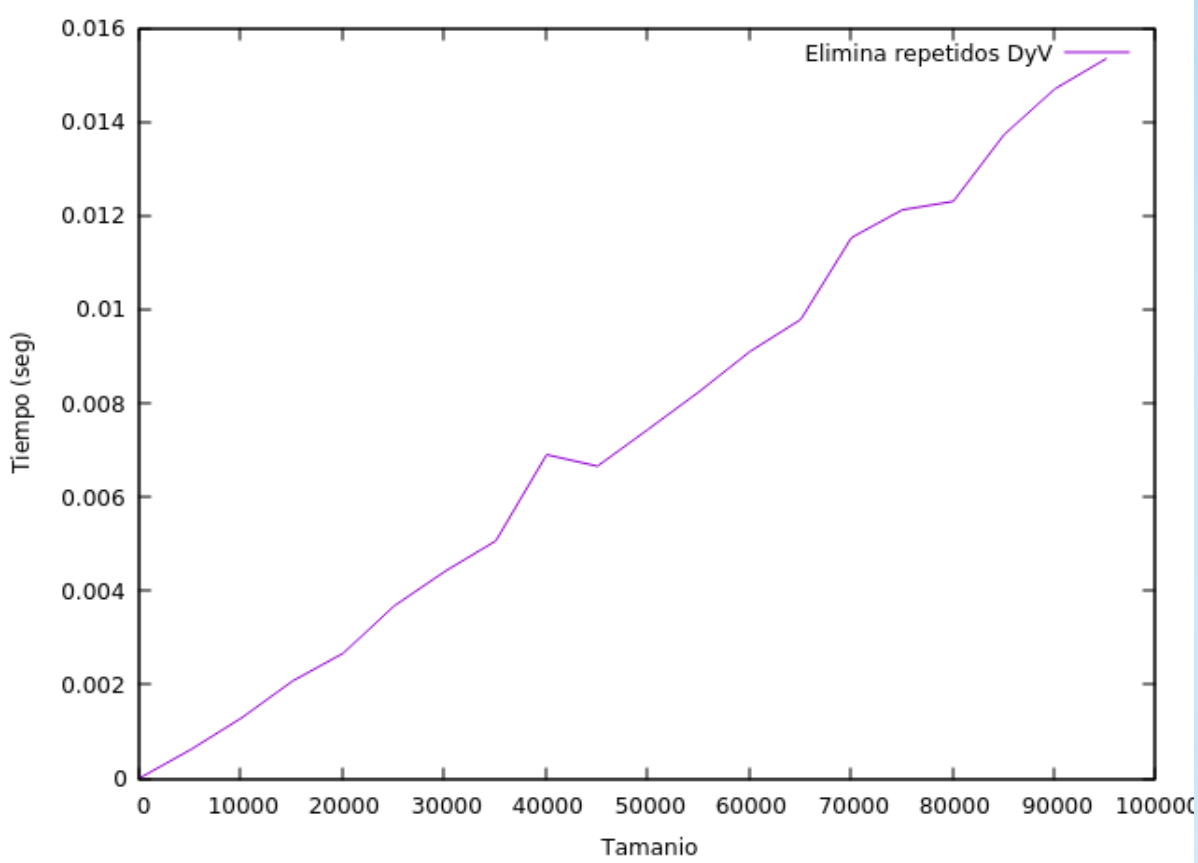
Algoritmo de eliminación de elementos → **O(n)**

TOTAL:

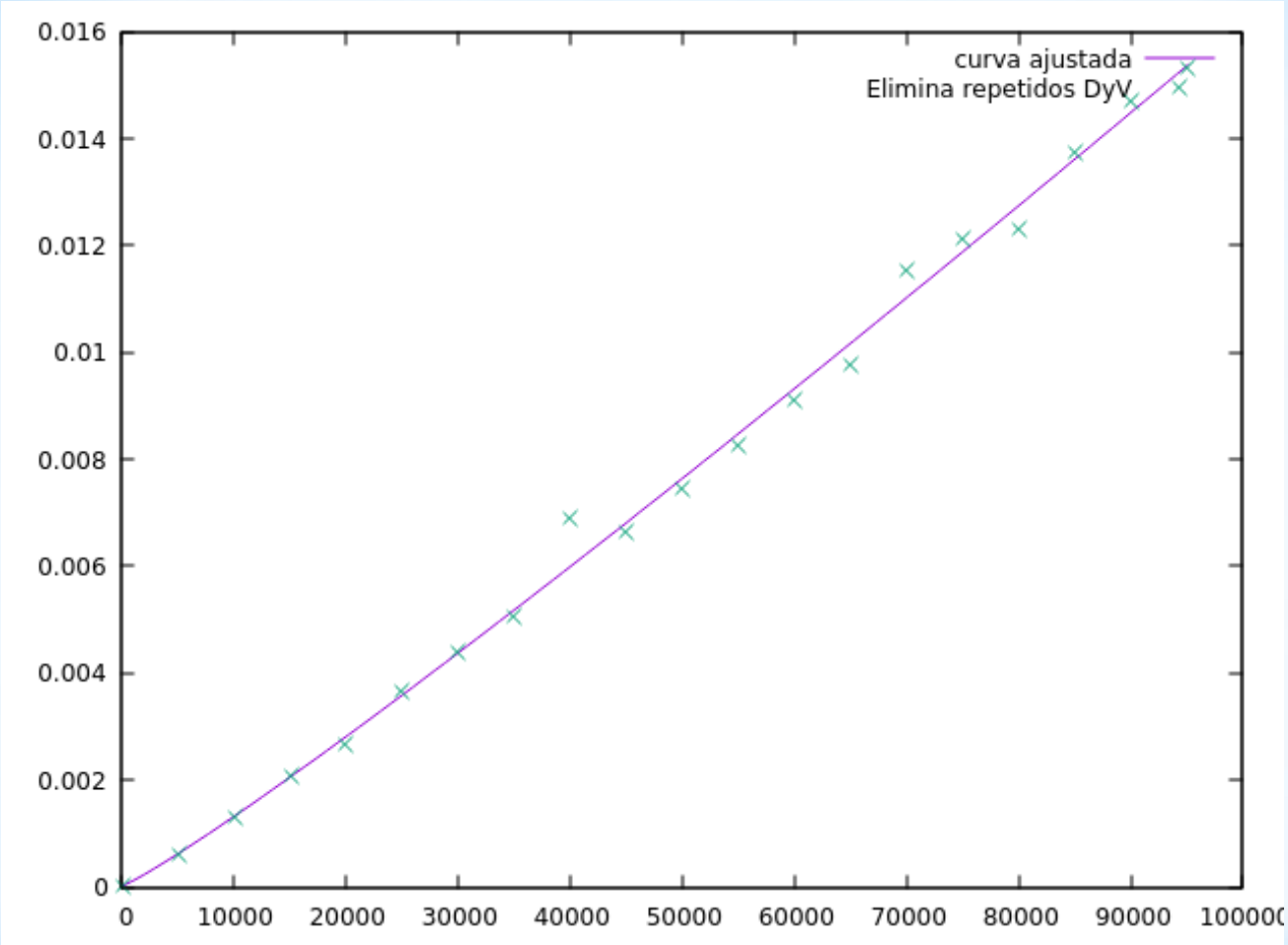
O(n*log(n)) + O(n) = O(n*log(n))

5. ELIMINAR REPETIDOS
DyV – EFICIENCIA TEÓRICA

Tamaño	tiempo
100	1.13e-05
5100	0.0006161
10100	0.0012946
15100	0.0020756
20100	0.0026694
25100	0.0036772
30100	0.0044156
35100	0.0050642
40100	0.0069055
45100	0.0066585
50100	0.0074444
55100	0.0082465
60100	0.0091049
65100	0.0097859
70100	0.0115315
75100	0.0121244
80100	0.0123026
85100	0.0137282
90100	0.0147036
95100	0.0153416



5. ELIMINAR REPETIDOS DyV - EFICIENCIA EMPÍRICA



$f(x) = a0*x*\log(n)$

5. ELIMINAR REPETIDOS DyV – EFICIENCIA HÍBRIDA

Dos ejecuciones, para tamaños 15 y 30:

```
El tamaño del vector es: 15
El vector es: 14 13 0 6 4 8 3 7 1 8 2 4 14 0 0
El tamaño del vector es: 10
El vector es: 0 1 2 3 4 6 7 8 13 14
```

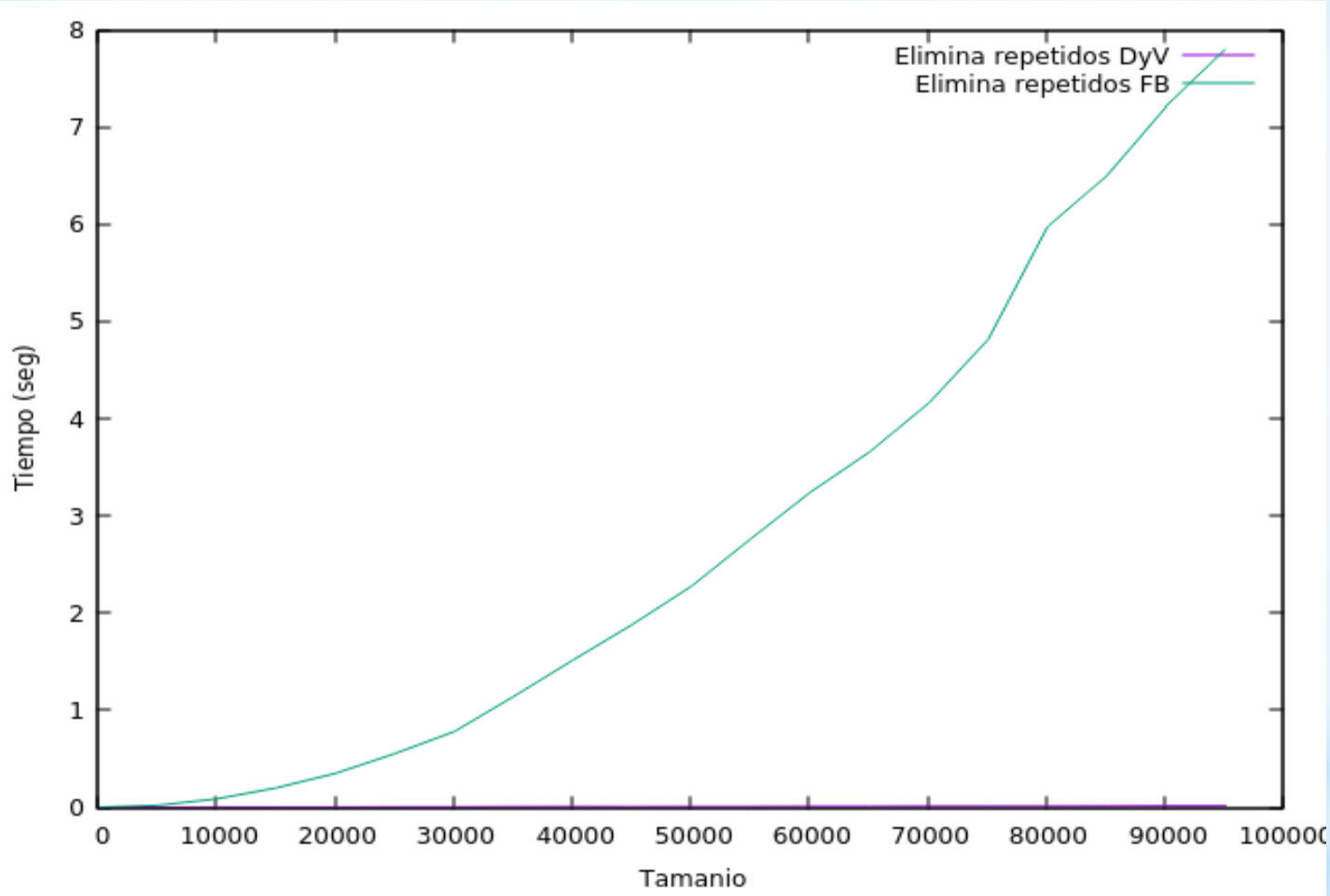
```
El tamaño del vector es: 30
El vector es: 2 5 9 18 6 16 15 22 18 1 26 23 10 0 11 2 3 14 2 28 22 11 8 11 16 23 3 20 5 0
El tamaño del vector es: 19
El vector es: 0 1 2 3 5 6 8 9 10 11 14 15 16 18 20 22 23 26 28
```

5. ELIMINAR REPETIDOS
DyV – CASO DE EJECUCIÓN

Empíricamente tenemos que ambas son cuadráticas, por lo que la versión DyV no es mejor que la versión por fuerza bruta, aunque sí que mejora el tiempo de ejecución a partir de un tamaño de entrada mayor a 4096, siendo hasta 2 veces más rápida.

Luego nos quedaría que el mejor algoritmo en tiempo de ejecución es la versión DyV, aunque no se consigue una mejora de eficiencia.

6. COMPARATIVA TRASPUESTA



7. COMPARATIVA ELIMINA REPETIDOS

Matriz Traspuesta:

- No hay mejora en orden de eficiencia
- Se reduce el tiempo de ejecución en hasta un 50%

Eliminar Repetidos:

- Eficiencia mejorada
- No implementación DyV pura

DyV:

- Mejora en tiempos de ejecución
- No siempre mejora orden eficiencia
- Su implementación no siempre es segura

8. CONCLUSIONES