ALGORÍTMICA

Práctica 2: Divide y vencerás

Grupo Petazetas: Sara Bellarabi El Fazazi, Manuel Villatoro Guevara , Arturo Sánchez Cortés, Sergio Vargas Martin

Índice:

1. Problema común: Traspuesta de una matriz

PSEUDOCODIGO TRASPUESTA.CPP

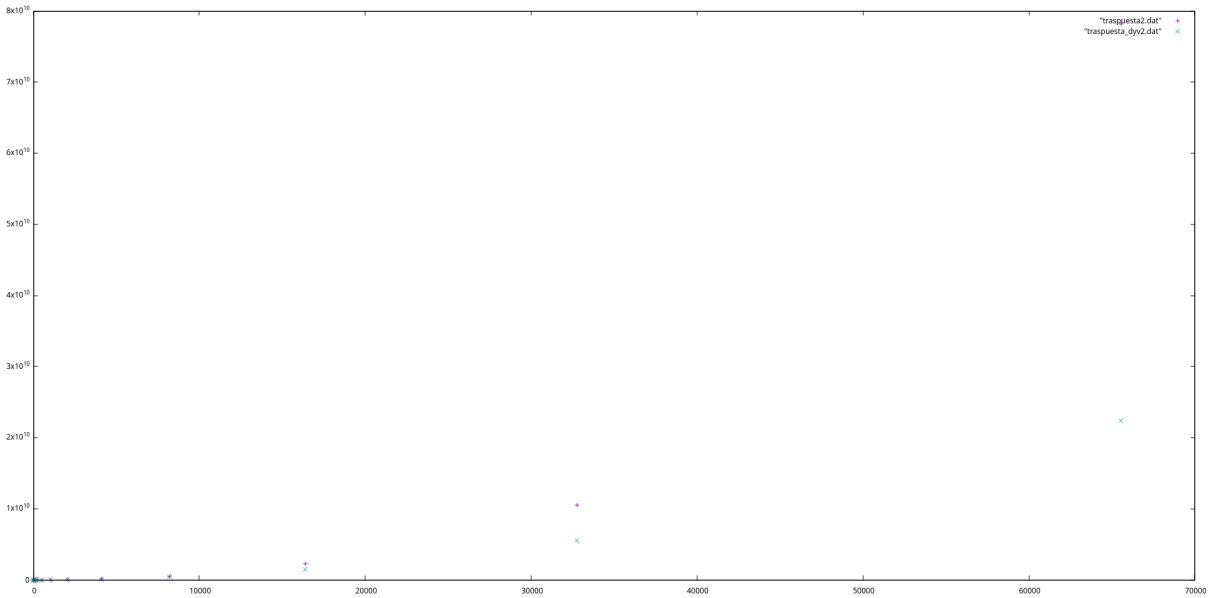
```
01. ALGORITMO Transpuesta;
02. FUNCIÓN transpuesta(matriz &m) : matriz
03.         m2: matriz
04.       FOR(int i=0-; i < m.size(); i++)
05.                 FOR(int j=0-; j < m[0].size(); j++)
06.                           m2[j][i] = m[i][j];
07.         RETURN m2;
08.
09. FUNCIÓN transpuesta_dyv(matriz &entrada, matriz &salida, int x0, int x1, int y0, int y1) : void
10.       x_mid, y_mid : int
11.       IF (abs(y0 - y1) == 0)
12.                  salida[x1][y1] = entrada[y1][x1];
13.                  salida[x0][y0] = entrada[y0][x0];
14.       ELSE
15.                 traspuesta_dyv(entrada, salida, x0, x_mid + 1, y0, y_mid);
16.                 traspuesta_dyv(entrada, salida, x0, x_mid + 1, y_mid + 1, y1);
17.                 traspuesta_dyv(entrada, salida, x_mid, x1, y0, y_mid);
18.                 traspuesta_dyv(entrada, salida, x_mid, x1, y_mid + 1, y1);
19. FUNCIÓN transpuesta_dyv(matriz m) : matriz
20.       m2 : matriz
21.       traspuesta_dyv(m, m2, 0, m[0].size() - 1, 0, m.size() - 1);
22.       RETURN m2;
```

1.1 Traspuesta de una matriz: análisis teórico.

Este algoritmo, dada una matriz de tamaño n= 2^k nos devuelve la traspuesta de dicha matriz teniendo en cuenta que n es el número de elementos de la matriz,
Una usando la técnica “divide y vencerás” y otra versión más sencilla y directa, pero menos eficiente.
En este problema no se ha notado evidente mejoría a la hora de analizar la eficiencia del algoritmo divide venceras al normal.
En el caso del normal n, el primer for empieza en la primera posición del vector y el siguiente for itera en las posiciones restantes del vector haciendo los cambios oportunos para obtener la matriz traspuesta. Sin embargo, la eficiencia no sería cuadrática como cabría pensar al ver los dos bucles sino lineal, ya que el segundo for se va “limitando” debido a que a medida que el primer for va a avanzando en la siguiente posición del vector, el segundo for itera sobre un vector restante que cada vez se va reduciendo más. Esto hace que la eficiencia del algoritmo sea de O(n).

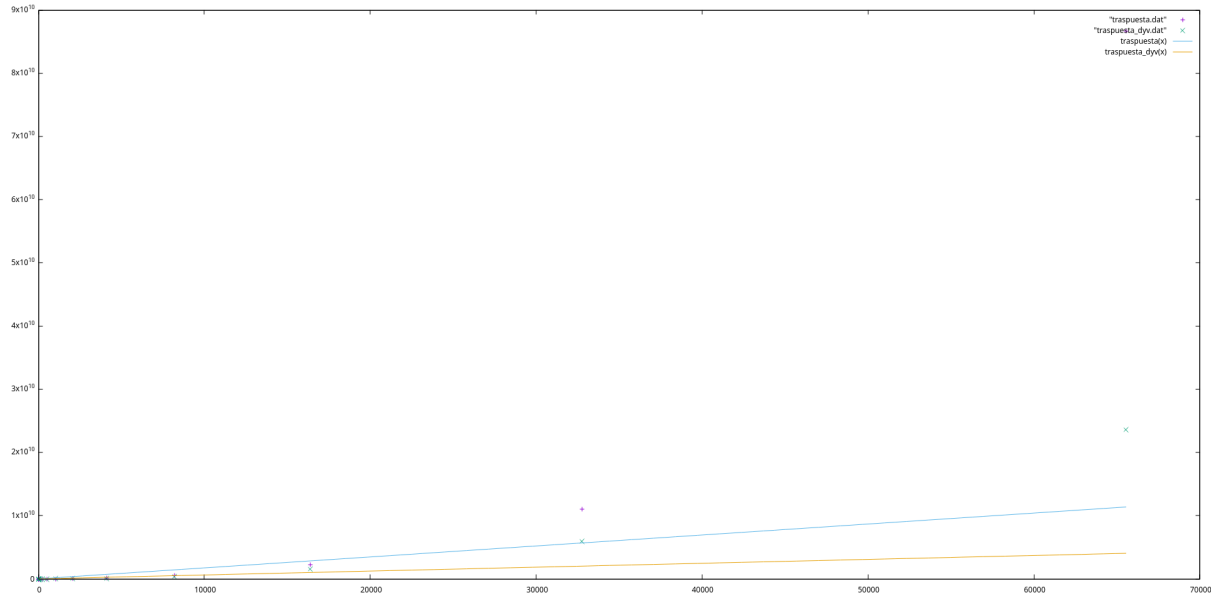
En el caso de divide y vencerás nos fijamos en las condicionales if y else que tenemos.
El primer if tiene un bucle for que va desde i=0 ó i=j+1 (O(n)). A continuación el else llama recursivamente a la función, cuyo orden ya hemos calculado (O(n)). Luego el orden del algoritmo sería el máximo orden del if y del else: O(n)).
El mejor caso sería que la traspuesta de la matriz fuese idéntica a la matriz original, pero al no realizarse en el algoritmo ninguna comprobación previa la eficiencia sería igual a la del peor caso tanto el divide venceras como en el algoritmo de fuerza bruta : Ω(n).
Como podemos observar no existe una mejoría evidente entre la eficiencia del algoritmo usando la técnica divide y vencerás.

1.2 Análisis empírico



1.3 Análisis híbrido

Para calcular la constante oculta hemos usado el mismo programa que en la práctica anterior, está junto al resto de códigos y se llama calcular_k.cpp



traspuesta2.dat K: 173564.245605
traspuesta_dyv2.dat K: 61643.950594

1.4 Estudio del umbral

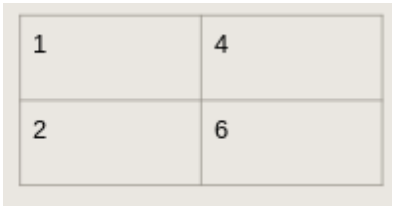
Como podemos observar por el estudio teorico y empirico, nos encontramos con que ambas eficiencias son lineales, de forma que nunca tendrían un punto en común. Para ver el umbral tenemos que fijarnos en la tabla de tiempos y ver en qué momento el divide y vencerás tarda menos para un mismo tamaño y vemos que para 2048 tarda menos.

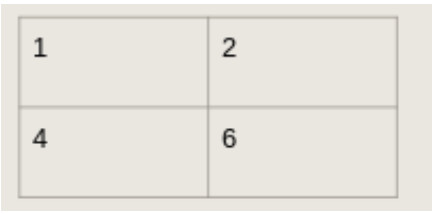
N	Fuerza bruta(t)	Divide y venceras (t)
2	4303	9232
4	1892	2856
8	2248	4348
16	4080	9576
32	9100	31832
64	40528	109166
128	112830	396717
256	425701	1427165
512	2260010	2498899

1024	2043310	5083034
2048	8737627	22465887
4096	111298673	83556186
8192	513980989	369877279
16384	2241478308	1481458946
32768	10534747606	5522607984
65536	78154152038	22438807118

1.5 Descripción caso de ejecución: traspuesta de una matriz

Algoritmo fuerza bruta:





i = 0
k = 1 hasta < a.size() (2)

swap a[0][1] = a[1][0]

FIN

Algoritmo divide y vencerás:

i=0, j=2 (inicio, y fin)

abs(i-j) != 1
por tanto se ejecuta el bloque else:

trasponer_dyv(a, i, (j+i)/2); i=0, j=1 → abs(i-j) ==1 → swap a [0][1] 0 a[1][0]

trasponer_dyv(a, i, (j+i)/2); i=1, j=2 → abs(i-j) ==1 → swap a [1][2] 0 a[2][1]

2. Problema asignado: Comparación de preferencias

PSEUDOCODIGO PROTOTIPO.CPP

```
01. ALGORITMO Transpuesta;
02. FUNCIÓN uniforme() : double
03.       t : int              //random
04.       f : double  //RAND_MAX
05.       RETURN  t/f;
06. FUNCIÓN generador(int n) : vector
07.       T : vector;
08.       FOR (int j = 0; j < n; j++)
09.                 T.push_back(j);
10.       FOR (int j = n - 1; j > 0; j--)
11.                 u : double ←  uniforme();
12.                 k : int ←  j * u
13.                 tmp : int ←  T[j]
14.                 T[j] = T[k];
15.                 T[k] = tmp;
16.       RETURN T;
17. FUNCIÓN fuerza_bruta(vector v) : int
18.       inv : int <-- 0
19.       FOR (int i = 0; i < v.size() - 1; i++)
20.                 FOR (int j = i + 1; j < v.size(); j++)
21.                           IF (v[i] > v[j])
22.                           inv++;
23.       RETURN inv;
24. FUNCIÓN fusion(vector &vec, int iz, int der) : int
25.       aux(der - iz + 1) : vector;
26.       centro ← (iz + der)/2, inv ← 0, i ← iz, j ← centro + 1, k  : int
27.       FOR (k = 0; i <= centro && j <= der; k++)
28.                 IF(vec[i] <= vec[j])
29.                 aux[k] = vec[i];
30.                 i++;
31.                 ELSE
32.                 aux[k] = vec[j];
33.                 j++;
34.                 inv += centro - i + 1;
35.       FOR (; i <= centro; k++, i++)
```

```
36.             aux[k] = vec[i];
37.     FOR (; j <= der; k++, j++)
38.             aux[k] = vec[j];
39.     Copiar los primeros (der-iz +1) elementos del vector aux al vector vec a partir de la
posición iz.
40.     RETURN inv;
41. FUNCIÓN inversiones(vector &vec, int iz, int der) : int
42.     centro ← (iz + der)/2 : int
43.     IF (iz >= der)
44.             RETURN 0;
45.     ELSE
46.             RETURN inversiones(vec, iz, centro) + inversiones(vec, centro + 1, der) +
fusion(vec, iz, der);
```

2.1. Comparación de preferencias: análisis teórico.

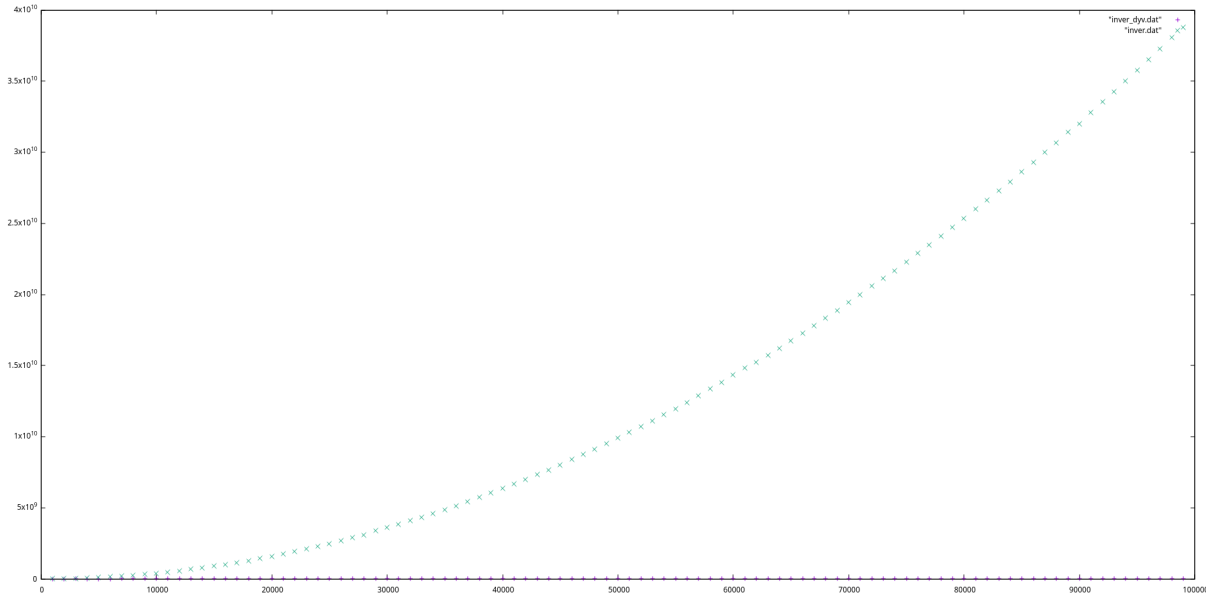
Se trata de un algoritmo para medir la similitud de dos rankings en función de las preferencias de dos usuarios. Para resolver el problema lo que hace el algoritmo de fuerza bruta es ir comprobando todos los pares (i,j) lo cual nos daría un orden cuadrático $O(n^2)$

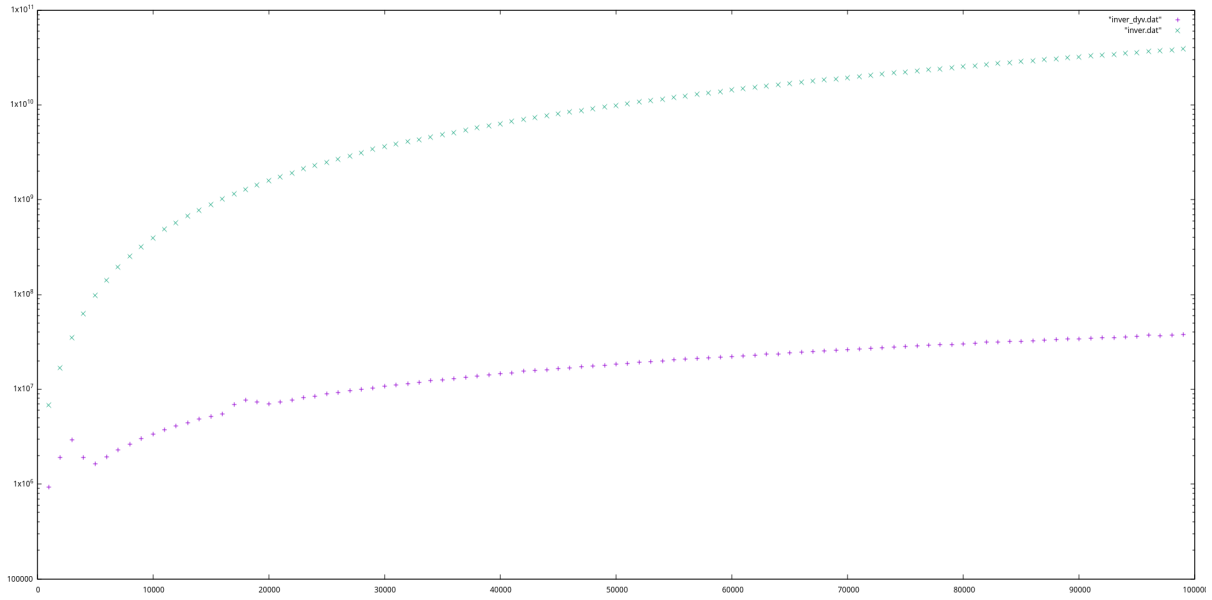
En el caso del algoritmo divide y vencerás, este procederá a dividir la lista de productos en dos mitades y contar recursivamente el número de inversiones en cada mitad. A continuación, asumiendo que cada mitad está ordenada, se contabilizan las inversiones en la que ai y aj están en mitades diferentes. Por último se realiza una mezcla de las dos mitades para devolver un conjunto ordenado.
Conteo: O(n); Mezcla: O(n).
La ecuación del tiempo de ejecución quedaría así:

$T(n) <= T(n/2) + T(n/2) + O(n) \rightarrow T(n) = O(nlogn)$

En el mejor caso ambos usuarios tendrían la misma preferencia de rankings, sin embargo esto no se comprueba en el algoritmo por lo que la eficiencia sería igual a la del peor caso: $\Omega(n^2)$ para la fuerza bruta y $\Omega(nlog(n))$ para divide vencerás.

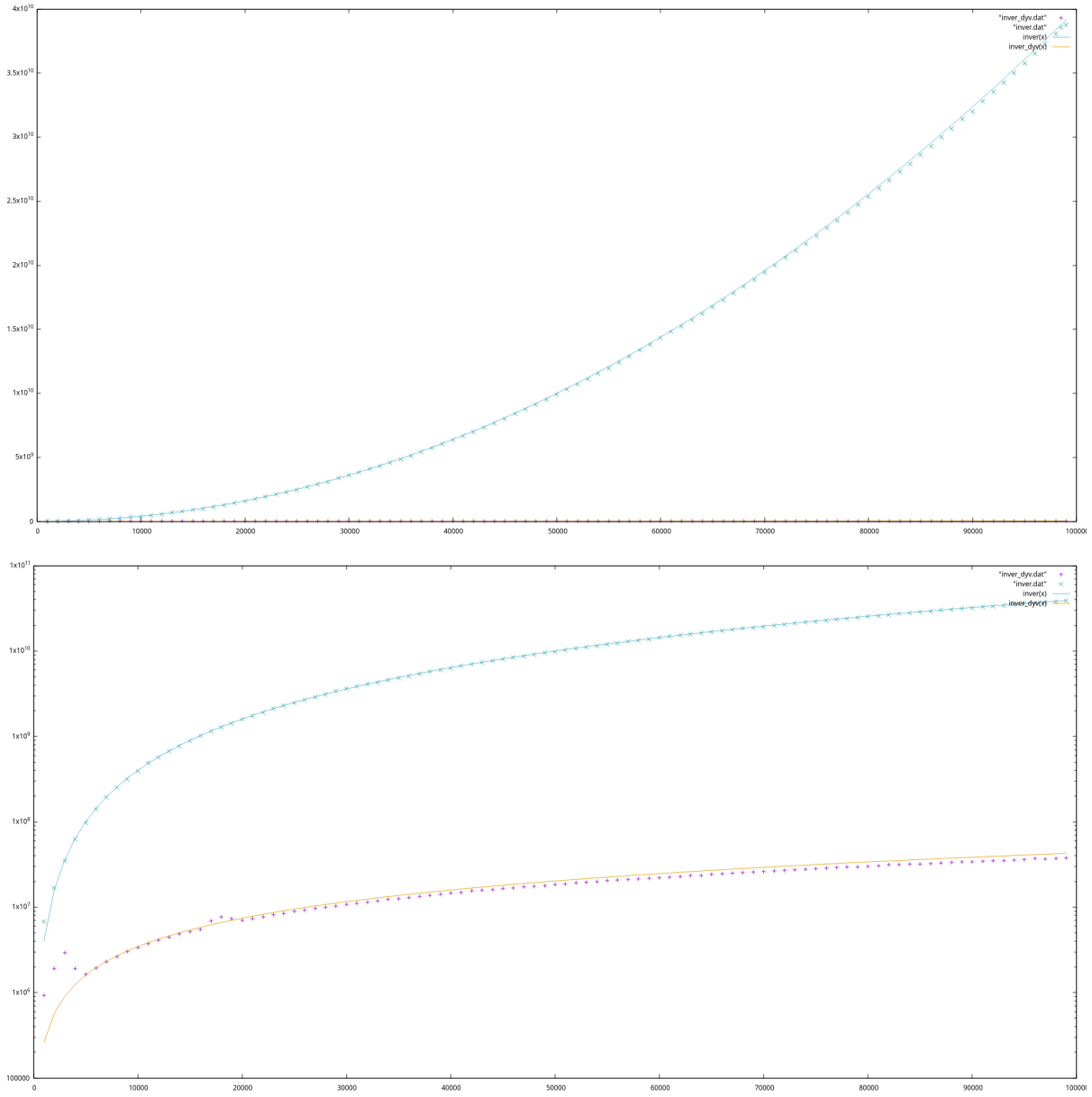
2.2 Análisis empírico





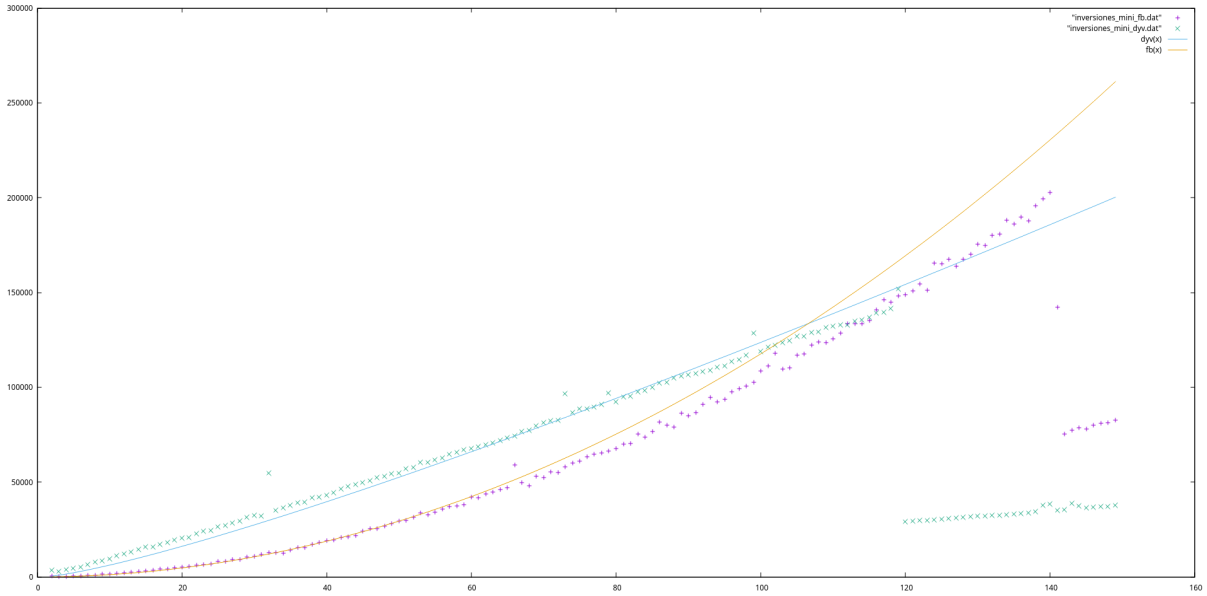
2.3 Análisis híbrido

inver.dat K: 3.996632
inver_dyv.dat K: 37.512211



2.4 Estudio del umbral

Como vemos en las gráficas anteriores, en ningún momento se cruzan las líneas, por lo que necesitamos tamaños de problema menor que 1000.



inversiones_mini_fb.dat K: 11.763392
inversiones_mini_dyv.dat K: 268.567322

Para ver el punto exacto de corte resolvemos un sistema de ecuaciones compuesto por las dos funciones híbridas de los algoritmos:

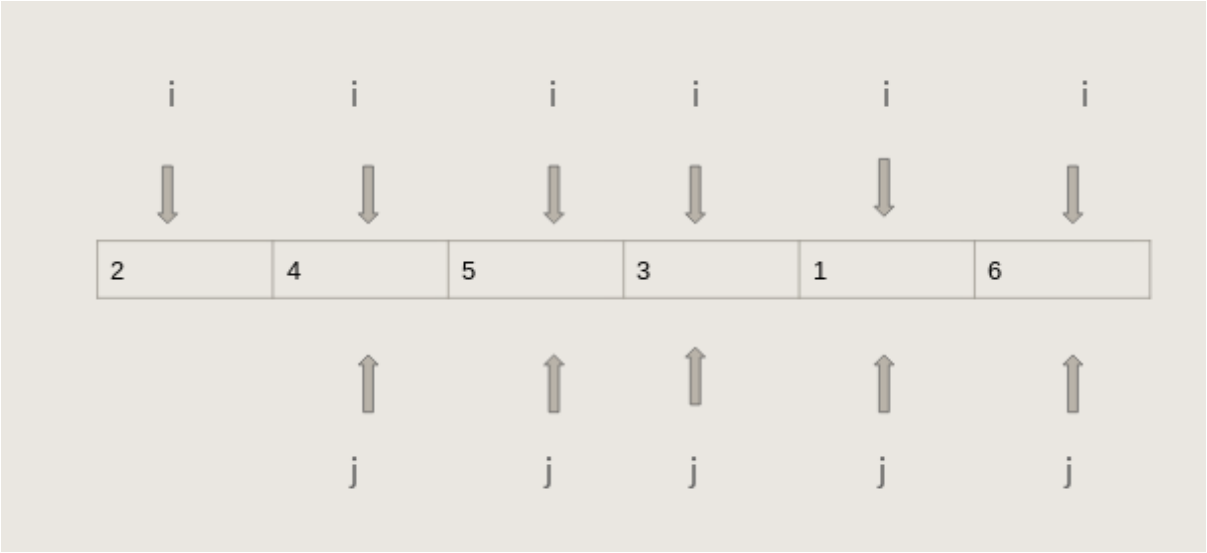
$y=x \cdot \log(x) \cdot 268.567322$
$y=11.763392 \cdot x^2$

Y vemos que queda: x = 106.598, y = 133670

Por tanto para tamaños de problema superiores a 106 es mejor el algoritmo divide y vencerás, para tamaños inferiores es mejor el fuerza bruta

2.5. Descripción caso de ejecución: comparación de preferencias

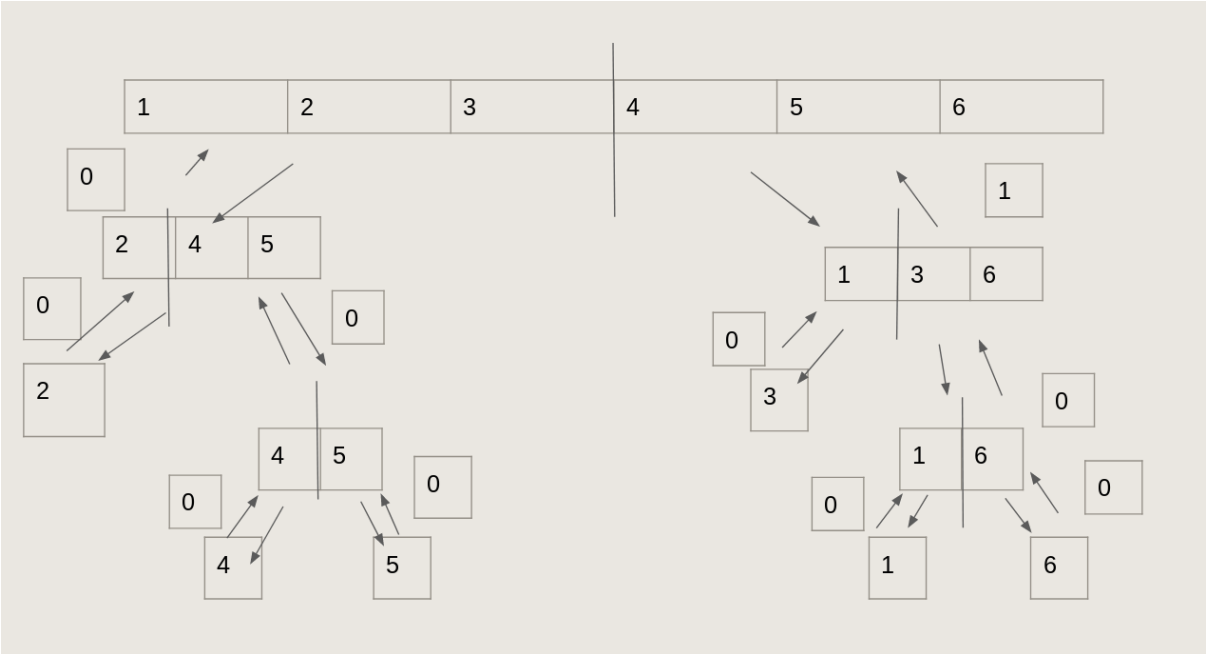
Algoritmo fuerza bruta:



v[i]>v[j]? → No → Inversión = 6

Número total de inversiones= 6

Algoritmo divide y vencerás:



Número de inversiones → 6