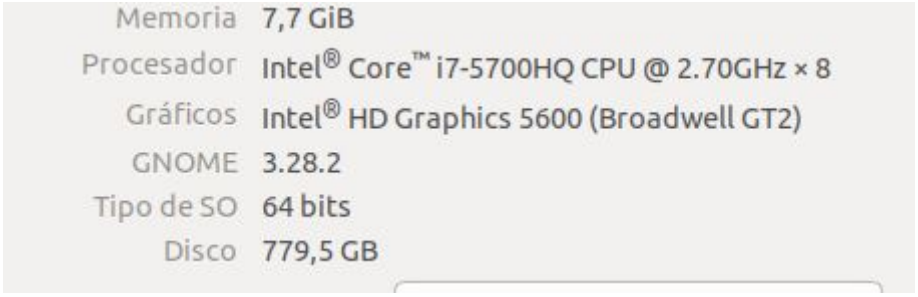
ALGORÍTMICA (2018-2019)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Práctica 2. Grupal
Grupo Ciri

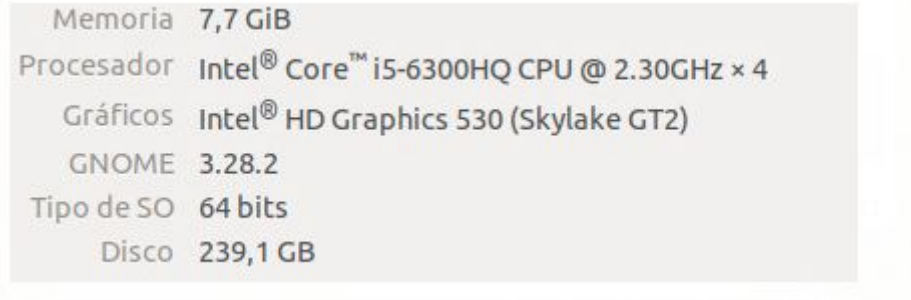
MARTÍN QUIRÓS, JUAN ANTONIO
MARTÍNEZ IÁÑEZ, GONZALO
POZO RAMÍREZ, RAFAEL
SANTOS SALVADOR, JOSÉ
SORIA GONZÁLEZ, RAÚL

Características de los Ordenadores.

Juan Antonio Martín Quirós:

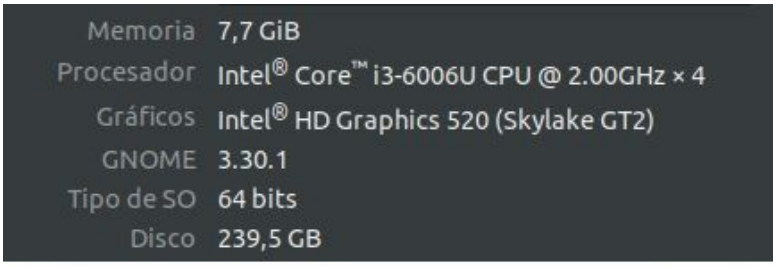


José Santos Salvador:



Rafael Pozo Ramírez:

Gonzalo Martínez láñez:



Raúl Soria González:

Objetivos

- Convertir un algoritmo en divide y vencerás.
- Calcular la matriz transpuesta en c++ mediante un algoritmo de divide y vencerás.
- Realizar un algoritmo de divide y vencerás para el problema de Comparación de preferencias.

Matriz traspuesta

Algoritmo para calcular la matriz transpuesta en c++.

```
void traspuesta(int **m, int **res, int inif, int finf, int inic, int finc){
        if(inif == finf)
                res[inif][inic] = m[inic][inif];
        else
        {
                int centrof = (inif + finf)/2;
                int centroc = (inic + finc)/2;

                traspuesta(m, res, inif, centrof, inic, centroc);
                traspuesta(m, res, inif, centrof, centroc + 1, finc);
                traspuesta(m, res, centrof + 1, finf, inic, centroc);
                traspuesta(m, res, centrof + 1, finf, centroc + 1, finc);
        }
}
```

A este algoritmo se le pasan dos punteros de punteros (matrices), la primera está cargada con la matriz original y la segunda está vacía. Lo que se busca mediante este algoritmo es reducir el tamaño de matriz en cuatro partes iguales y más fáciles de resolver. Para ello se tienen en cuenta cuatro índices, el que indica el principio de una fila, otro que indica el final y del mismo modo con las columnas.

En la primera iteración, se les pasan las índices que delimitan el tamaño completo de matriz, si no concuerdan el principio de las filas con el fin de las filas se marca el centro del tamaño de las filas y las columnas y se realiza una recurrencia llamando a esta misma función cuatro veces. El proceso se repite hasta llegar a matrices de tamaño 1x1 donde se asignará el valor de la posición de la matriz original[i][j] a la matriz traspuesta[j][i].

Con este algoritmo solo se permite el cálculo de matrices cuadradas y además siendo n el tamaño, $n = 2^k$ siendo k un número natural. En nuestro caso, para calcular matrices cuadradas que el tamaño no sea una potencia de dos, creamos una mayor y rellenamos los que sobra con ceros, por tanto el algoritmo se puede ejecutar sin que genere un core.

Eficiencia teórica

Para calcular la traspuesta sin usar un método sin divide y vencerás se deben anidar dos bucles que recorren la matriz entera y asignar el valor de la posición [i][j] a la posición de la nueva matriz [j][i].

La eficiencia de este algoritmo es de $O(n^2)$.

Para calcular la eficiencia del algoritmo con divide y vencerás hay que tener en cuenta que se realiza una función recursiva que crea cuatro nuevos problemas de un tamaño cuatro veces menor, por tanto se puede resolver la ecuación por recurrencia: $T(n) = 4T(n/4) + cn$

Cambio de variable, $n = 4^k$

$T(4^k) = 4T(4^k/4) + c4^k$; $T(4^k) = 4T(4^{k-1}) + c4^k$

Cambio de variable, $T(4^k) = t_k$

$t_k = 4t_{k-1} + c4^k$; $t_k - 4t_{k-1} = c4^k$

Ecuación característica: $(x - 4)(x - 4) = 0$; $(x - 4)^2 = 0$

$t_k = c_1 4^k + c_2 k 4^k$

Deshacer el cambio $k = \log_4 n$

$t_k = c_1 4\log_4 n + c_2(\log_4 n)4^{\log 4n}$

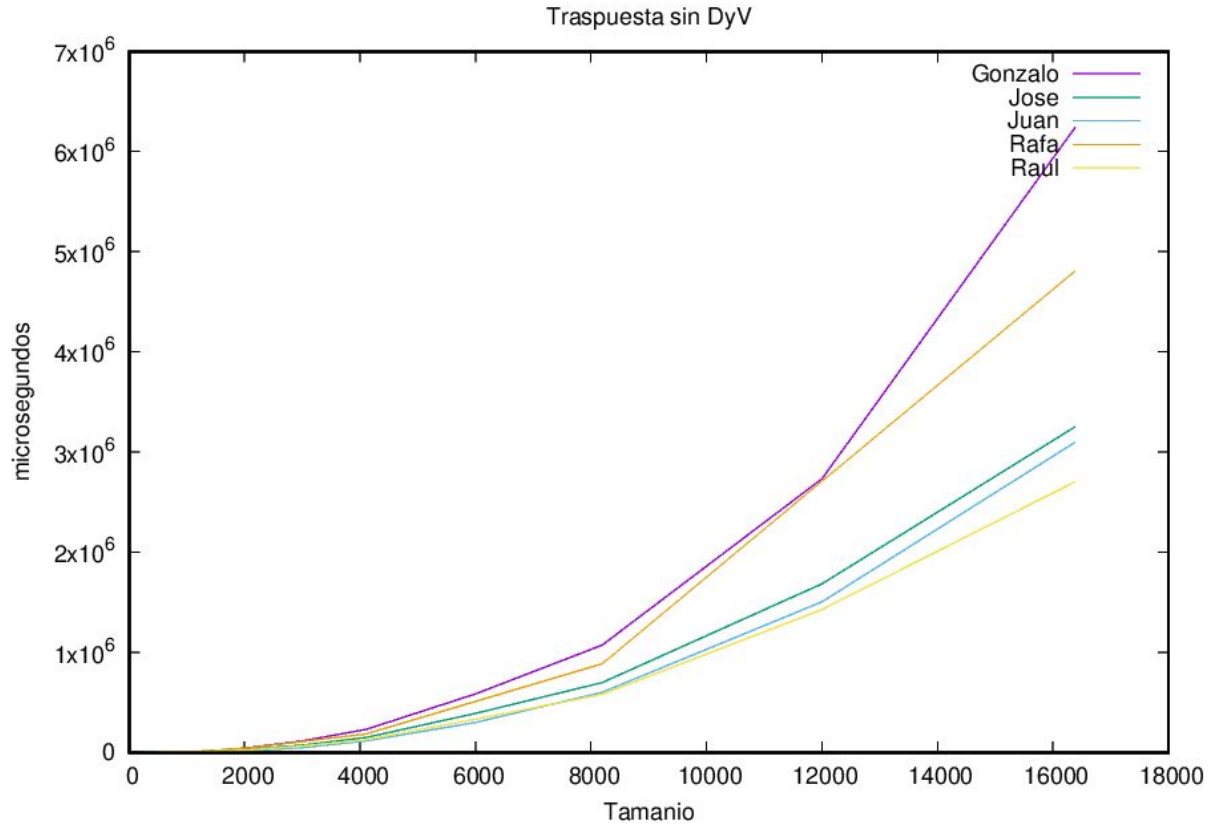
$t_k = c_1 n^{\log 44} + c_2 \log_4 n(n^{\log 44})$

$t_k = c_2 n(\log_4 n) + c_1 n$

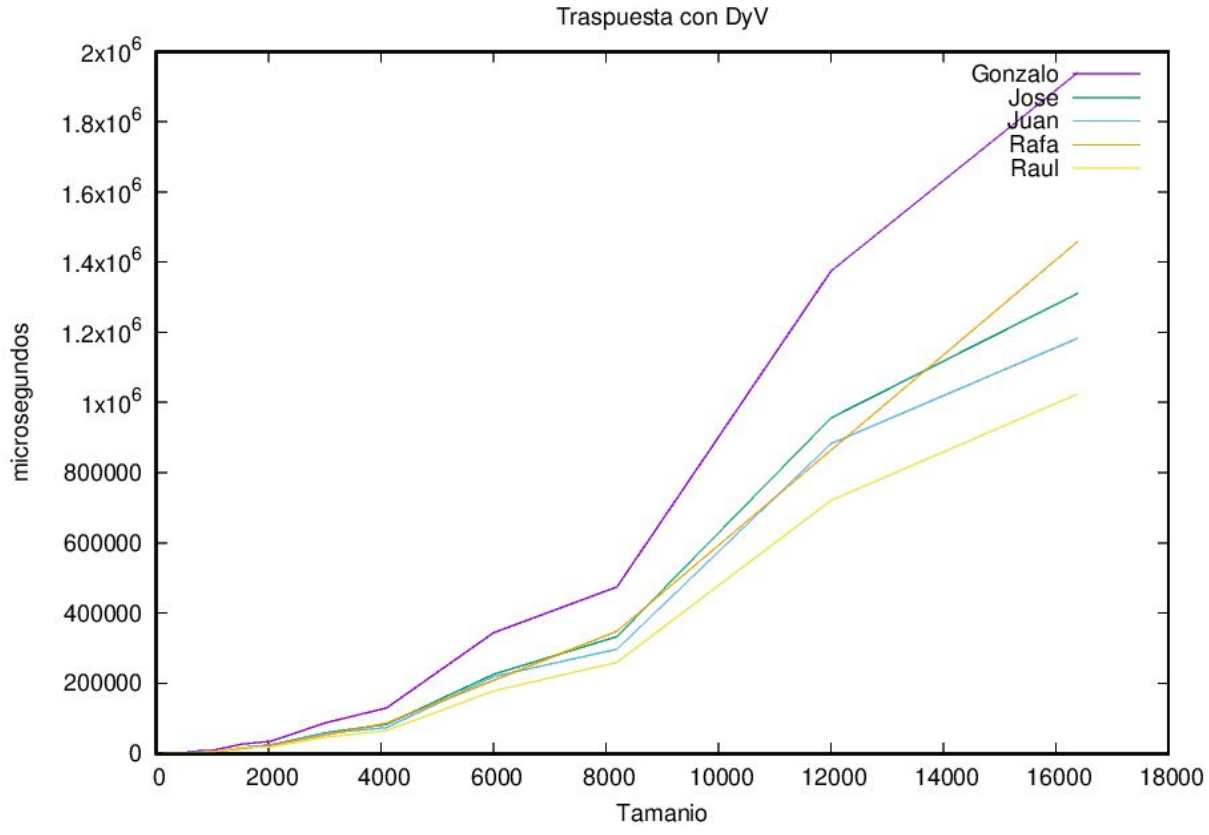
La eficiencia de este algoritmo es de $O(n*\log_4 n)$.

Eficiencia empírica

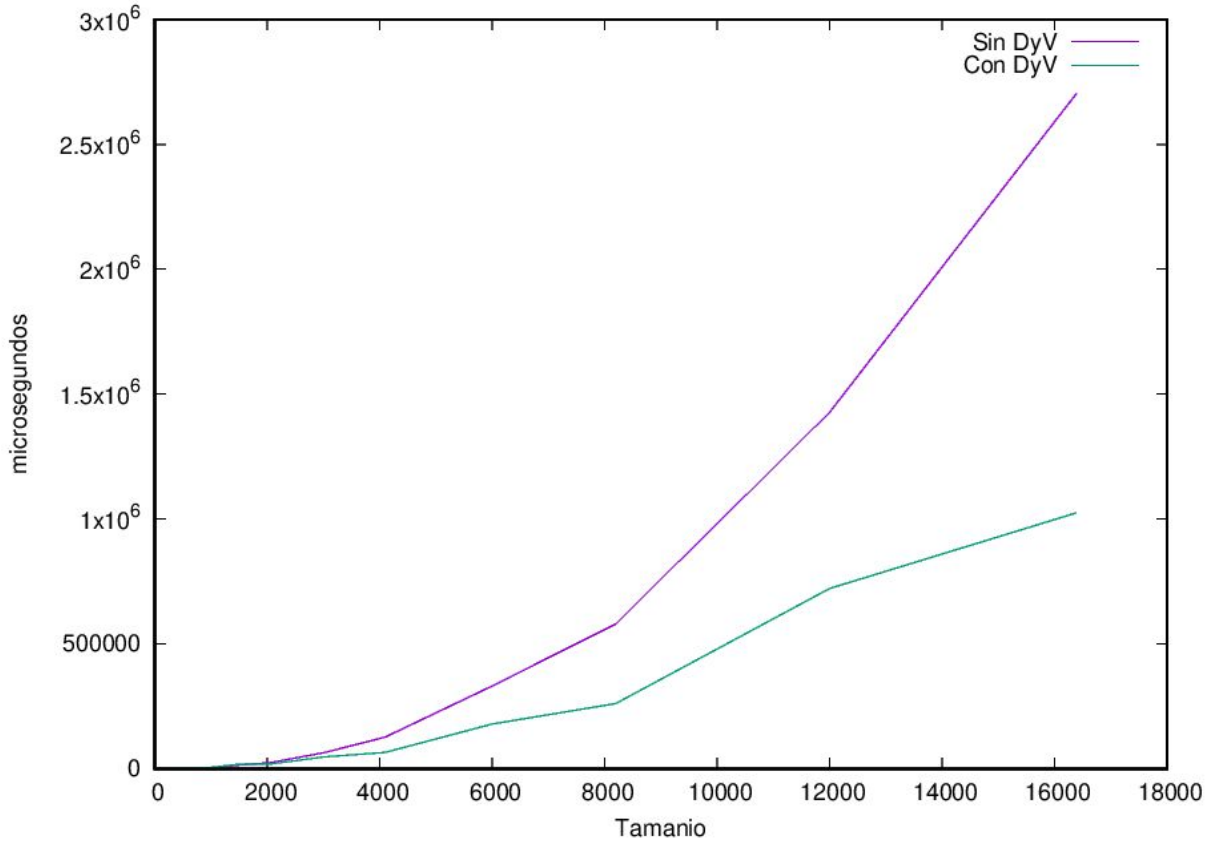
Gráfica con los tiempos sin usar método divide y vencerás.



Gráfica con los tiempos de usar el método de divide y vencerás.



En esta gráfica se muestra una comparación de los tiempos entre el cálculo de matriz usado el método de divide y vencerás y sin usarlo.

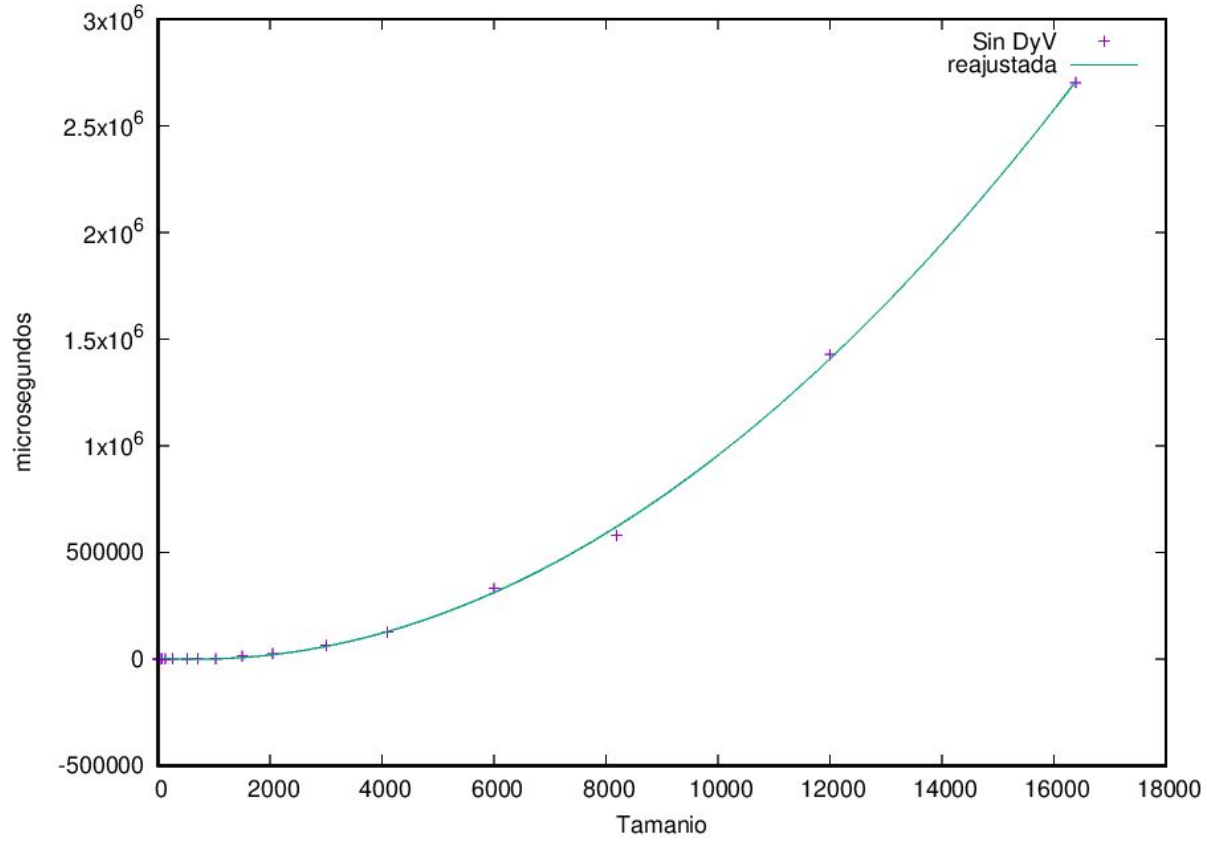


Tamaño	Sin DyV (µseg)	Con DyV (µseg)
8	0	2
16	1	4
32	2	14
64	9	72
128	38	224
256	49	295
512	321	1271
1024	1898	5285
2048	23876	18526
4096	126280	65017
8192	579049	260104
16384	2702220	1024050

Eficiencia híbrida

Para realizar la eficiencia empírica hemos usado gnuplot que tiene la herramienta para ajustar por mínimos cuadrados.

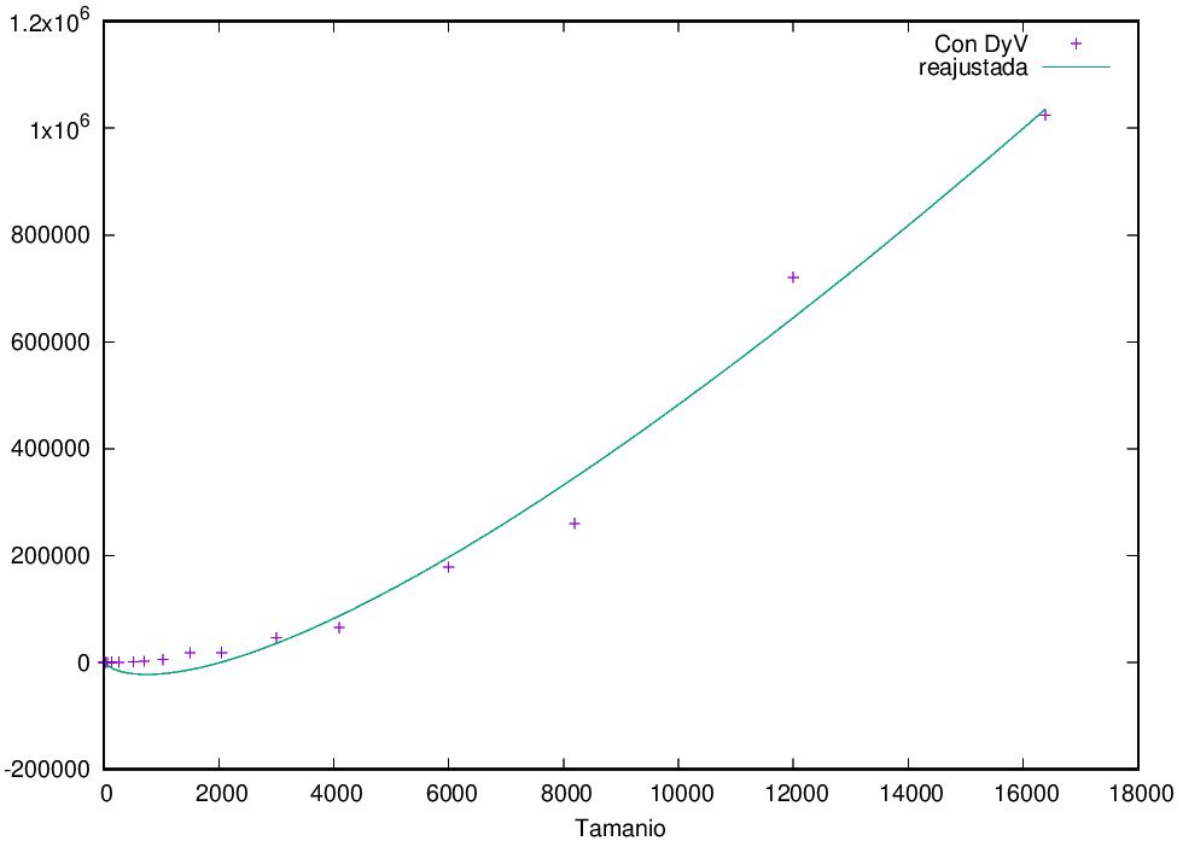
Gráfica sin divide y vencerás con el ajuste.



$f(x) = a_0*x^2 + a_1*x + a_2$

a0 = 0.0109336
a1 = -14.1681
a2 = 2691.4

Gráfica con divide y vencerás con el ajuste.



$f(x) = a_0 * x * \log_4 x + a_1 * x$

$a_0 = 41.9085$
$a_1 = -230.173$

Conclusión sobre divide y vencerás

Como se puede apreciar en la tabla comparando ambos algoritmos, se puede ver que el algoritmo sin divide y vencerás es más eficiente para valores inferiores a 2^{10}, pero al pasar ese tamaño el algoritmo que emplea divide y vencerás es más eficiente.

Algoritmo de comparación de preferencias

Explicación del problema

El algoritmo de comparación de preferencias se basa en calcular qué tan parecidos son las “preferencias” de 2 usuarios contando el número de inversiones que tienen entre sí.
Para realizar ésto, se realiza calculando si dos preferencias están o no invertidas entre sí. Por ej: El usuario A, “prefiere” antes el valor 1 al 4 y el usuario B, prefiere antes el valor 4 al 1, así pues tendrían preferencias invertidas y viceversa.

De ésta forma se puede determinar cuán parecido son dos usuarios, convirtiendo cada producto en un valor numérico y generando un vector formado con estos valores numéricos para, de esta forma poder compararlos.

Código del algoritmo de comparación de preferencias

```
int comparacionPreferencias(const vector<int> &a, vector<int> &b, int ini, int fin){
   int res;
   if (ini == fin)   {
      if (b[ini] != a[ini])      {
         int pos = BuscarBinario(a, 0, a.size()-1, b[ini]);
         if (pos > ini)         {
            int aux = b[ini];
            b[ini] = b[ini + 1];
            b[ini + 1] = aux;
         }
         else{
            int aux = b[ini];
            b[ini] = b[ini - 1];
            b[ini - 1] = aux;
         }
         res = 1;
      }
      else
         res = 0;
   }
   else{
      int centro = (ini + fin) / 2;
      res = comparacionPreferencias(a, b, ini, centro) + comparacionPreferencias(a, b, centro + 1, fin);
   }
   return res;
}
```

No hemos realizado el análisis de la eficiencia empírica ya que hemos tenido problemas para automatizar el proceso y además en algunos casos se genera un core que detiene la ejecución, no obstante hemos realizado la eficiencia teórica.

Eficiencia teórica

Este algoritmo divide el vector uno de los vectores en dos mitades sucesivamente y por tanto la ecuación de recurrencia es del tipo: $T(n) = 2T(n/2) + cn$

Cambio de variable, $n = 2^k$.
$T(2^k) = 2T(2^k/2) + c2^k$; $T(2^k) = 2T(2^{k-1}) + c2^k$

Cambio de variable, $T(2^k) = t_k$.
$t_k = t_{k-1} + c2k$; $t_k - t_{k-1} = c2^k$

Ecuación característica: $(x - 2)^2 = 0$.
$t_k = c_1 2^k + c_2 k 2^k$

Deshacer el cambio de variable, $k = \log_2 n$
$t_k = c_1 2^{\log 2n} + c_2 \log_2 n(2^{\log 2n})$
$t_k = c_1 n + c_2 n(\log_2 n)$

La eficiencia de este algoritmo es $O(n*\log_2 n)$