Divide y vencerás

Comparación de preferencias

S.Cruz, C.Enríquez, M.Ariza, J.Bueno

PROBLEMA 1

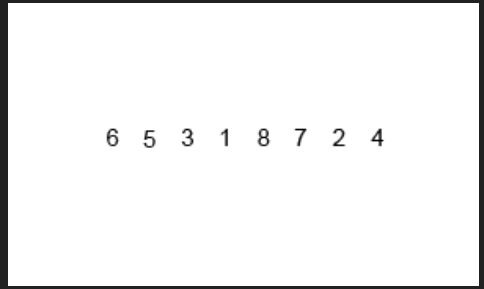
Dado un ranking de n productos (p.ej.películas) mediante el cual los usuarios indicamos nuestras preferencias, un algoritmo puede medir la similitud de nuestras preferencias contando el número de inversiones: dos productos i y j están “invertidos” en las preferencias de A y B si el usuario A prefiere el producto i antes que el j, mientras que el usuario B prefiere el producto j antes que el i.

i y j están invertidos si $i < j$ pero $a_i > a_j$

PROBLEMA 1

Proponemos 2 algoritmos para la resolución de este problema.

- Algoritmo trivial
- Algoritmo usando Divide y Vencerás (mergesort)



TRIVIAL:

Dado un vector v de n elementos con índices 0, ..., n − 1, la idea trivial es la siguiente:

- Para i desde 0 hasta n − 1
- Para j desde i + 1 hasta n − 1
- Si v[i] > v[j]
- inversiones++
- Devolver inversiones.

```
int fuerzabruta(vector<int> v){
int n_inversiones = 0;

    for (int i = 0; i < v.size(); i++){
        for(int j = i+1; j < v.size(); j++){
            if(v[i]>v[j])
            n_inversiones++;
        }
    }
return n_inversiones;
}
```

PROBLEMA 1

MERGESORT ADAPTADO: Tenemos dos partes claramente diferenciadas en el algoritmo: la parte que se encarga de dividir y la parte que se encarga de mezclar.

```
int sortAndCount(int *v, int begin, int end){
    int middle = (begin + end) / 2;
    int invs = 0;

    if(middle - begin > 1)
        invs += sortAndCount(v,begin,middle);

    if(end - middle > 1)
        invs += sortAndCount(v,middle,end);

    if(begin < middle && middle < end)
        invs += mergeAndCount(v,begin,middle,end);

    return invs;
}
```

```
int mergeAndCount(int *v, int begin, int middle, int end){
    int *sorted = new int[(unsigned)(end - begin)];

    int invs = 0, j = middle, i = begin, k = 0;

    while(i < middle){
        if(j < end){

            if(v[i] <= v[j]){
                sorted[k] = v[i];
                k++; i++;
            }
            else{
                sorted[k] = v[j];
                k++;j++;
                invs += (middle -i);
            }
        }
        else{
            sorted[k] = v[i];
            k++; i++;
        }
    }

    while(j < end){
        sorted[k] = v[j];
        k++; j++;
    }

    for(int r = 0; r < end - begin; r++){
        v[r+begin] = sorted[r];
    }

    delete [] sorted;
    return invs;
}
```

PROBLEMA 1

EFICIENCIA DE LOS ALGORITMOS:

Algoritmo trivial

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1}(n-i-1) = \frac{n(n-1)}{2}$$ ∈ O(n^2)

Algoritmo Divide y Vencerás

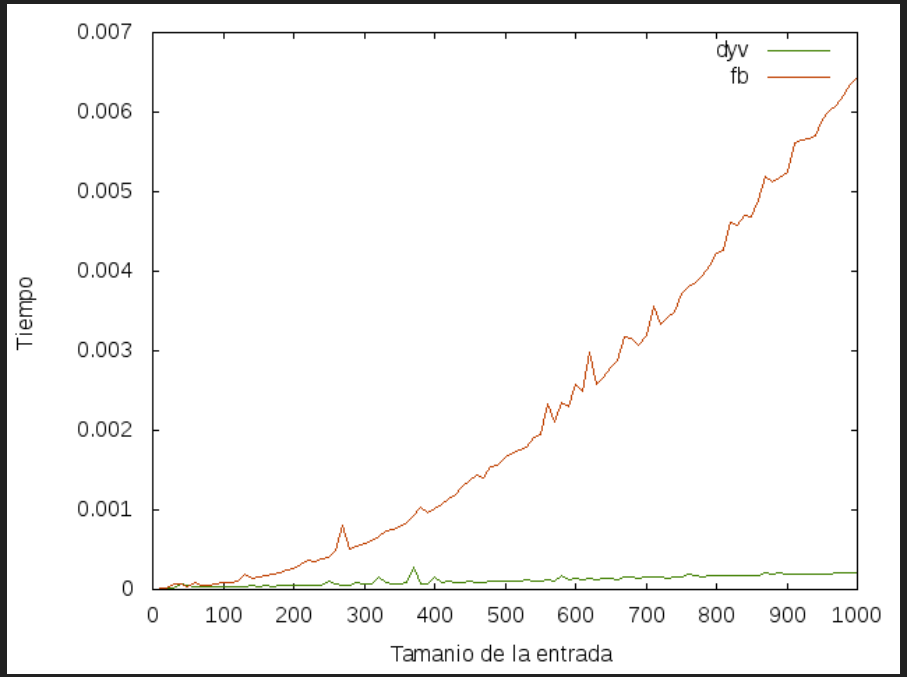
T (n) = 2T (n/2) + n

n = 2^k

T (2^k) = 2T (2^k−1) + 2^k ⇐⇒

T (2^k) = a2^k + bk2^k ⇐⇒

T (n) = an + bn log n ∈ O(n log n)



Divide y vencerás

Eliminar elementos repetidos

S.Cruz, C.Enríquez, M.Ariza, J.Bueno

PROBLEMA 2

Dado un vector de n elementos, de los cuales algunos pueden estar duplicados, el problema es obtener otro vector donde todos los elementos duplicados hayan sido eliminados.

OBJETIVO:
Diseñar, analizar la eficiencia e implementar un algoritmo sencillo para esta tarea, y luego hacer lo mismo con un algoritmo más eficiente, basado en Divide y Vencerás, con eficiencia O(n log n)

Para resolver este problema proponemos 2 soluciones:

- Solución trivial
- Solución “mezcla y elimina”

PROBLEMA 2

SOLUCIÓN TRIVIAL: Introducir el primer elemento del vector en el vector resultado. Recorrer el resto de posiciones del vector dado y para cada elemento que no este ya en el vector resultado añadirlo.

```
vector<int> FuerzaBruta(std::vector<int> v)
{
    std::vector<int> x;

    sort(v.begin(),v.end());

    x.push_back(v[0]);
    int k=0;
     for(int i=1;i<v.size();i++)
     {
        if(v[i] != x[k])
        {
            k++;
            x.push_back(v[i]);
        }
     }

     return x;
}
```

PROBLEMA 2

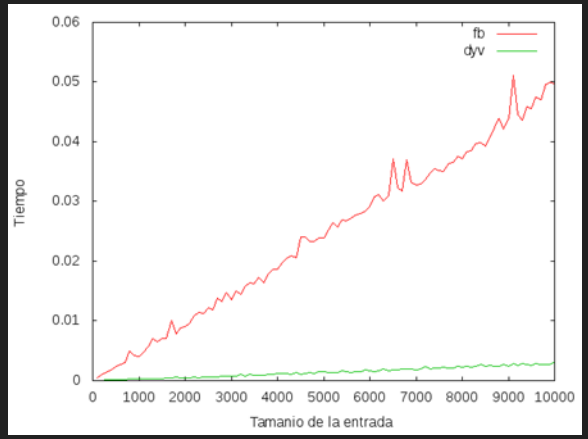
SOLUCIÓN “MEZCLA Y ELIMINA”:

Dividimos el problema en 2 subproblemas de tamaño mitad aproximadamente, eliminamos los duplicados en cada una de las mitades, y luego eliminamos los elementos duplicados en las dos mitades.

```
vector<int> EliminarDuplicados( vector<int> v )
{
    int mitad = v.size() / 2;
    std::vector<int> a;
    std::vector<int> b;
    std::vector<int> a_devolver;

    if(v.size()==1)
        return v;

    for(int i=0;i<mitad;i++)
    {
        a.push_back(v[i]);
    }


    for(int j=mitad;j<v.size();j++)
    {
        b.push_back(v[j]);
    }

    a=EliminarDuplicados(a);
    b=EliminarDuplicados(b);
    V=mezcla_y_elimina(a,b);

    return v;
}
```

PROBLEMA 2

Si suponemos que los subproblemas se resuelven eliminado duplicados y ordenándolos de menor a mayor, entonces es posible diseñar un método eficiente para componer las soluciones de los dos subproblemas (que también ordena el resultado de menor a mayor):




```
std::vector<int> mezcla_y_elimina(std::vector<int> a,std::vector<int> b){
    std::vector<int> a_devolver;
    int final_A=a.size();
    int final_B=b.size();
    int indice_a=0;
    int indice_b=0;

    while(indice_a<final_A && indice_b<final_B){
        if(a[indice_a] < b[indice_b]){
            a_devolver.push_back(a[indice_a]);
            indice_a++;
        }
        else if(a[indice_a] > b[indice_b]){
            a_devolver.push_back(b[indice_b]);
            indice_b++;
        }
        else if( a[indice_a] == b[indice_b] ){
            a_devolver.push_back(b[indice_b]);
            indice_a++;
            indice_b++;
        }
    }

    if(indice_a==final_A){
        while(indice_b<final_B){
            a_devolver.push_back(b[indice_b]);
            indice_b++;
        }

    }

    if(indice_b==final_B){
        while(indice_a<final_A){
            a_devolver.push_back(a[indice_a]);
            indice_a++;
        }
    }
    return a_devolver;
}
```

Divide y vencerás

Cada elemento en su posición

S.Cruz, C.Enríquez, M.Ariza, J.Bueno

PROBLEMA 3

Dado un vector V de números enteros, todos distintos, ordenado de forma no decreciente, se quiere determinar si existe un índice i tal que V[i] = i y encontrarlo en ese caso.
Para resolver este problema proponemos 2 soluciones:

-Solución trivial
-Solución Divide y Vencerás

SOLUCIÓN TRIVIAL: Recorrer el vector desde 0 hasta n-1. Si v[m] = m entonces devolvemos 'm' y acabamos la ejecución.
Ya que en el peor caso recorremos el vector al completo no cabe duda de que este algoritmo es O(n).

```
int fuerza_bruta(vector<int> &v,int TAM){
    for (int i = 1; i < TAM; ++i)
    {
        if(v[i]==i){
            return i;
        }
    }

    return 0;
}
```

PROBLEMA 3

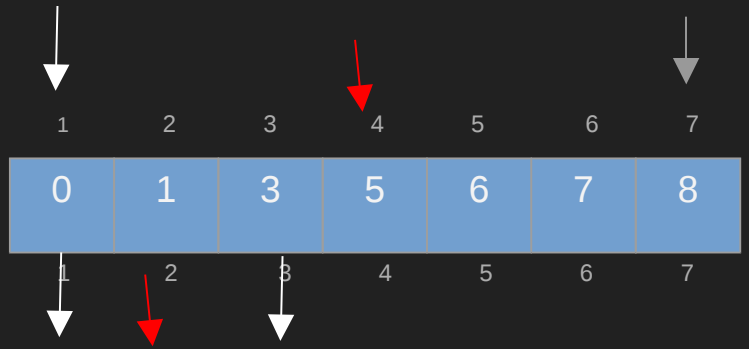
SOLUCIÓN “LOCALIZA BINARIA”:

Usamos la búsqueda binaria, examinando el elemento que se encuentra en la posición de la mitad, m = (primero+ultimo)/2.
Si coincide para ese elemento su valor con el índice (v[m] = m).
En caso contrario, si el valor de ese entero es mayor que el índice (v[m] > m), al estar los elementos ordenados y no repetirse, sabemos que para todos los índices mayores que m, los valores en esas posiciones serán siempre mayores que los propios índices, es decir v[j] > j, ∀j > m

```
int localiza(vector<int> &v, int primero, int ultimo){

  if (primero==ultimo)
  {
    if (v[primero]==primero)
    {
        return primero;
    }
    else
        return 0;
  }
  else{
    int i=(primero+ultimo)/2;
    if (v[i]==i)
    {
        return i;
    }
    else if (v[i]>i)
    {
        return localiza(v,primero,i-1);
    }
    else
    {
        return localiza(v,i+1,ultimo);
    }
  }
}
```

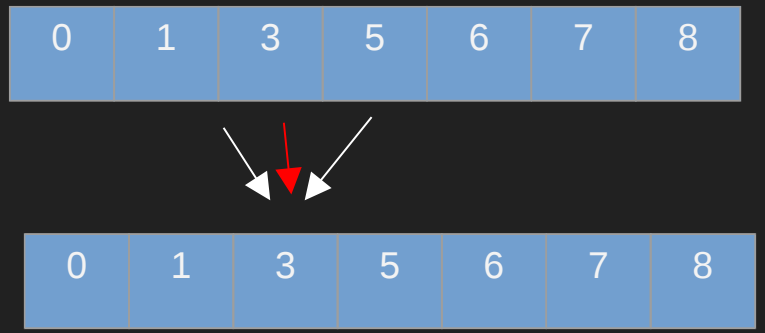
EJEMPLO

v[i] > i



La posición buscada está en la parte izquierda del vector.

v[i] < i



Nuestro elemento se encuentra en la subdivisión derecha.

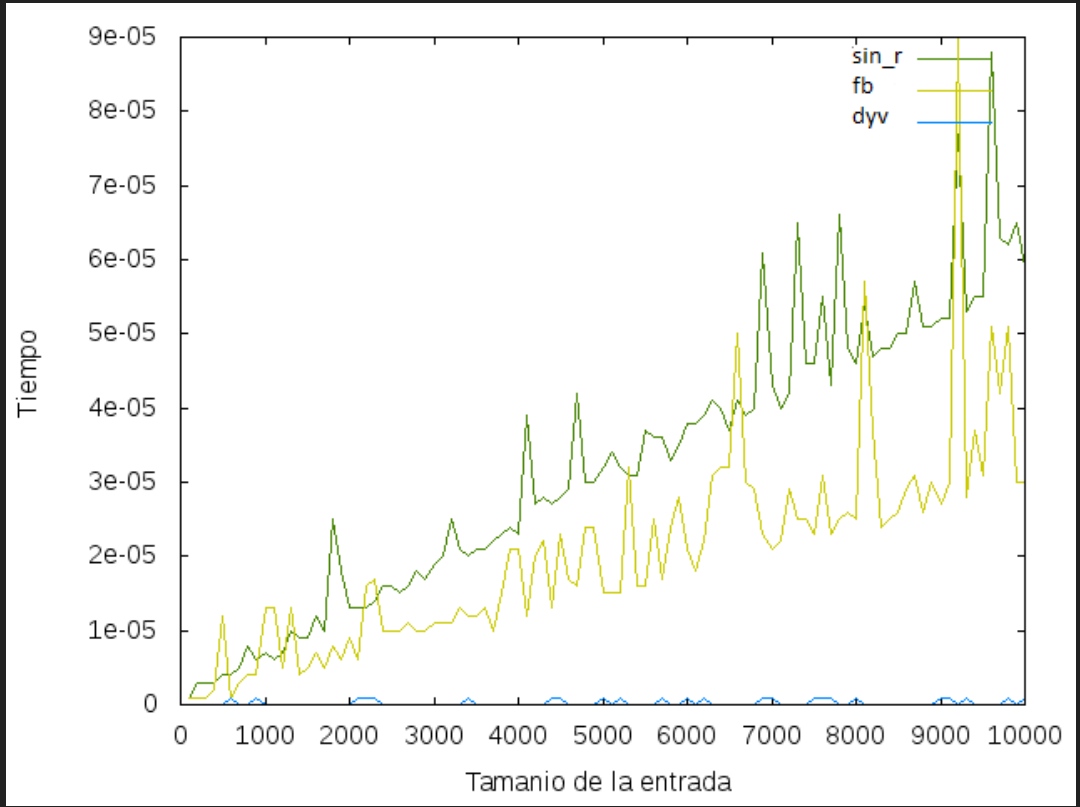
v[i] = i

Hemos encontrado la posición buscada.

PROBLEMA 3: EFICIENCIA

Peor de los casos:
$T(n) = 2T(n/2) + O(1)$

Solución:
Tenemos una ecuación que podemos resolver con cambio de variable $n = 2^k$:
$T(n) = T(2^k) = 2T(2^{k-1}) + O(1)$
$\Leftrightarrow\Rightarrow T(1) + (k-1)O(1) \in O(k) = O(\log n)$



Divide y vencerás

Mezcla de k vectores ordenados

S.Cruz, C.Enríquez, M.Ariza, J.Bueno

PROBLEMA 4

Tenemos un serie de k vectores ordenados de menor a mayor con el mismo número n de elementos. Queremos combinarlos en un único vector ordenado.

Matemáticamente:

Si tenemos *k* vectores de tamaño *n.*
El vector resultante tendrá un tamaño de *k x n*

PROBLEMA 4

TRIVIAL:
Lo único que hacemos es pasarle como argumento, un vector de vectores, que contiene todos los vectores que tenemos que mezclar entre ellos, cada vez que mezclamos dos vectores con la función MezclarVectoresOrdenados(), obtenemos un auxiliar que posteriormente será pasado como argumento a la función mencionada anteriormente y así hasta llegar al último vector de la lista de vectores.

```
std::vector<int> FuerzaBruta( vector < vector<int> > v )
{
    std::vector<int> aux=v[0];
    for(int i=1; i<v.size();i++)
    {
        aux=MezclaVectoresOrdenados(aux,v[i]);
    }

    return aux;
}

int main(int argc, char *argv[])
{
```

```
std::vector<int> MezclaVectoresOrdenados(std::vector<int> a,std::vector<int> b){
    std::vector<int> a_devolver;
    int final_A=a.size();
    int final_B=b.size();
    int indice_a=0;
    int indice_b=0;

    while(indice_a<final_A && indice_b<final_B){
        if(a[indice_a] < b[indice_b]){
            a_devolver.push_back(a[indice_a]);
            indice_a++;
        }
        else if(a[indice_a] > b[indice_b]){
            a_devolver.push_back(b[indice_b]);
            indice_b++;
        }
        else if( a[indice_a] == b[indice_b] ){
            a_devolver.push_back(b[indice_b]);
            a_devolver.push_back(a[indice_a]);
            indice_a++;
            indice_b++;
        }
    }

    if(indice_a==final_A){
        while(indice_b<final_B){
            a_devolver.push_back(b[indice_b]);
            indice_b++;
        }
    }

    if(indice_b==final_B){
        while(indice_a<final_A){
            a_devolver.push_back(a[indice_a]);
            indice_a++;
        }
    }
    return a_devolver;
}
```

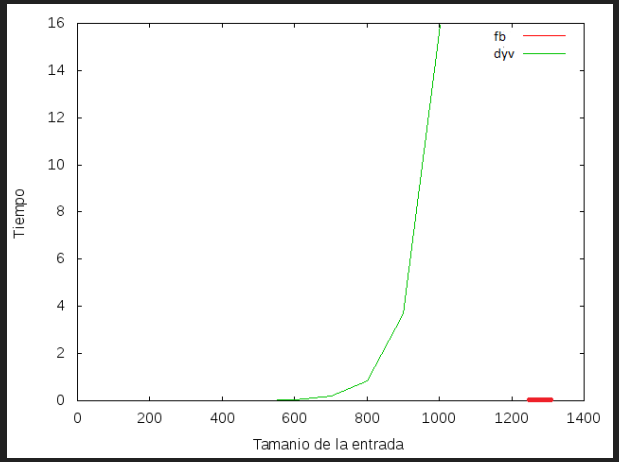
PROBLEMA 4

ALGORITMO RECURSIVO:

Recorremos un vector que contiene los vectores de enteros ordenados.
Emparejarlos hasta llegar al final del vector.
Llamar sucesivamente a MezclaVectoresOrdenados, que da como resultado el vector mezclado al que ha dado lugar la mezcla de los dos anteriores.
La llamada a la función MezclaVector() se llama de forma recursiva, tantas veces como sea divisible el tamaño del vector de vectores original entre 2.

```
vector< vector<int> > MezclaVectores( vector < vector<int> > v )
{
    vector< vector<int> > a_devolver;
    vector < vector<int> > aux;
    int j=0;

    if(v.size()==1)
        return v;

    for( int i=v.size();i>1;i/=2)
    {
        a_devolver.push_back(MezclaVectoresOrdenados(v[j],v[j+1]));
        j=j+2;
    }

    v=MezclaVectores(a_devolver);

    return v;
}
```

EFICIENCIA



Eficiencia del algoritmo trivial:

$$\sum_{i=1}^{k-1} (in+n)=n$$

$$\sum_{i=1}^{k-1} i + n \sum_{i=1}^{k-1} 1 = n(\ k\ (k-1)\ )/2 + (k-1)n = n(\ (k-1)(k+2)\ )/2$$

Eficiencia del algoritmo Divide y Vencerás:

En el proceso, en cada iteración mezcla k/2i = 2m−i parejas de vectores de tamaño 2i−1n, tardando pues un tiempo proporcional a 2m−i 2 2i−1n = 2mn = kn.
Como se realizan m = log k iteraciones, el tiempo total es proporcional a kn log k.

Divide y vencerás

Serie unimodal de números

S.Cruz, C.Enríquez, M.Ariza, J.Bueno

PROBLEMA 5

Sea un vector v de números con n componentes, todas distintas, de forma que existe un índice p (que no es ni el primero ni el último) tal que a la izquierda de p los números están ordenados de forma creciente y a la derecha de ‘p’ están ordenados de forma decreciente, es decir:

∀i , j ≤ p con **i < j ⇒ v[i] < v[j]** y **∀i, j ≥ p** con **i < j ⇒ v[i] > v[j]**

PROBLEMA 5

Soluciones:

- Trivial
- Dividir por la mitad
- TRIVIAL: La idea es sencilla: recorrer el vector hasta encontrar el punto p. Por tanto, en nuestro algoritmo trivial basta recorrer todas las componentes 2, 3, ..., n − 1 y encontrar la primera componente que verifica, que será p por unicidad.

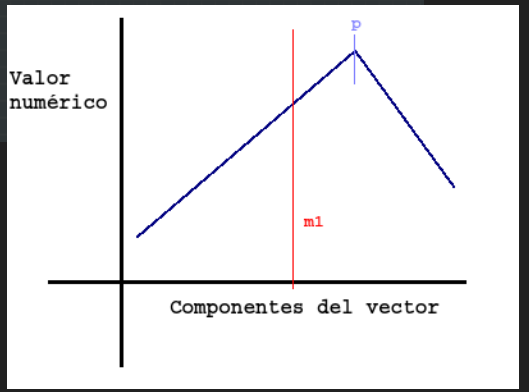
```
int FuerzaBruta(int *v, int size){
    bool es = false;
    int pos;
    for(int i=0; i< size-1 && !es; i++){
        if(v[i]>v[i+1]){
            pos = i;
            es = true;
        }
    }
    if(es == false)
        pos = size-1;
    return pos;
}
```

PROBLEMA 5

DIVIDE POR LA MITAD:

En primer lugar, comprobamos si el elemento en la posición mitad del vector verifica la imagen. En tal caso, este elemento debe ser ‘p’, devolviendo su índice como resultado del algoritmo. En el caso de no ser p recurrimos a divide y vencerás. Para ello, tal y como sucede en la búsqueda binaria, obviaremos uno de los dos vectores mitades (izquierda y derecha) llamando a nuestro algoritmo recursivo sobre el otro subvector.

```
int Divide_mitad(int *v, int principio, int fin){
    bool es = false;
    int medio = (fin+principio)/2;
    if (medio > principio){
        if(v[medio-1] < v[medio] && v[medio] < v[medio+1])
           return Divideyvenceras(v, medio, fin);
        else if(v[medio-1] > v[medio] && v[medio] > v[medio+1])
           return Divideyvenceras(v, principio, medio);
        else if(v[medio-1]<v[medio] && v[medio] > v[medio-1])
            return medio;
    }
    else{
        if(v[principio] <= v[fin]){
            return fin;
        }
        else
            return principio;
    }
}
```



PROBLEMA 5: Eficiencia

Teórica:

Peor caso: $T(n) = T(n/2) + \theta(1)$

Solución: Tenemos una ecuación de fácil solución. Tomando $n = 2^k$

$T(n) = T(2^k) = T(2^{k-1}) + \theta(1) = \ldots = T(1) + (k-1)\theta(1) \in \theta(k) = \theta(\log n)$