UNIVERSIDAD DE GRANADA

Divide y Vencerás

Algorítmica. Práctica 2

Jose Alberto Hoces Castro
Javier Gómez López
Moya Martín Castaño

Contenidos

Introducción

Problemas planteados

- **Ejercicio 1**: Buscar en un vector ordenado un elemento tal que $v[i] = i$.
- **Ejercicio 2**: Dados k vectores ordenados, de n elementos cada uno, combinarlos en un vector ordenado.

Objetivo de la prática

Apreciar la utilidad de la técnica divide y vencerás (DyV) para resolver problemas de forma más eficiente que otras alternativas más sencillas o directas.

Ejercicio 1

Búsqueda secuencial

Es la manera más obvia de buscar en un vector. Empezamos en el primer elemento y lo vamos recorriendo hasta encontrar el elemento deseado. En caso de no encontrarlo, devolvemos un valor que indique error (en nuestro caso -1).

Búsqueda secuencial. Código

```
int buscarSecuencial(int v[], int n){
    for (size_t i = 0; i < n; i++) //O(n)
    {
        if (v[i] == i){ //O(1)
            return i; //O(1)
        }
    }

    return -1;//O(1)
}
```

Búsqueda secuencial. Eficiencia teórica

Observamos claramente que

$$T(n) \in \mathcal{O}(n)$$

Búsqueda secuencial. Eficiencia empírica

Búsqueda secuencial	
Elementos (n)	**Tiempo (s)**
1760000	0.0165694
2520000	0.0262689
3280000	0.0336055
4040000	0.0368924
4800000	0.0399273
5560000	0.0485439
6320000	0.0529679
7080000	0.0585823
7840000	0.0649594
8600000	0.0723527
9360000	0.0801981
10120000	0.0856522
10880000	0.0922361
11640000	0.0992702
12400000	0.105115
13160000	0.114969
13920000	0.118283
14680000	0.123955
15440000	0.132098
16200000	0.139156
16960000	0.146774
17720000	0.150614
18480000	0.157312
19240000	0.163214
20000000	0.169743

Tabla 1: Experiencia empírica de la búsqueda a fuerza bruta

Búsqueda secuencial. Eficiencia híbrida

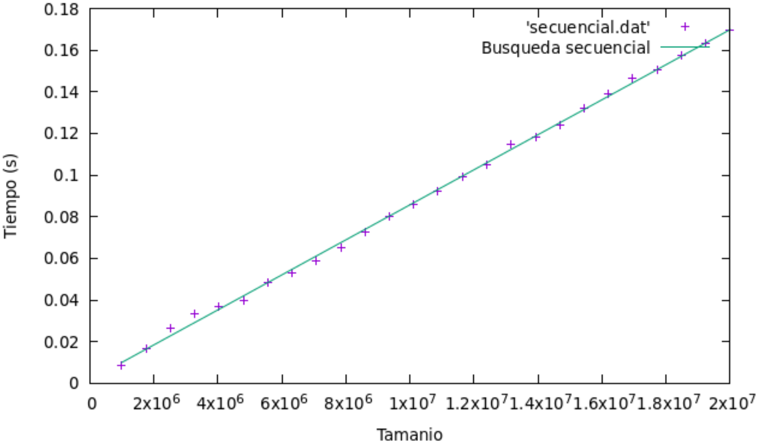

Figura 1: Gráfica con los tiempos de ejecución de la búsqueda a fuerza bruta

Búsqueda binaria

La tećnica Divide y Vencerás usada es la búsqueda binaria. Al estar ante un vector ordenado, podemos recurrir hasta algoritmo cuya eficiencia es logarítmica, mucho más preferible que una lineal.

Búsqueda binaria. Código

```
<<<<<<< HEAD
int buscarBinaria(int *v, int inicio, int fin){
    if(fin >= inicio){ // O(1)
        int medio = inicio + (fin - inicio) / 2; // O(1)

        if(v[medio] == medio){ // O(1)
            return medio; // O(1)
        }

        if(v[medio] > medio){ // O(1)
            return buscarBinaria(v, inicio, medio - 1); // O(n/2)
        }

        //else
        return buscarBinaria(v, medio + 1, fin); // O(n/2)
    }

    return -1; // O(1)
}
=======
```

Búsqueda binaria. Eficiencia teórica

Observamos claramente que

$$T(n) = T\left(\frac{n}{2}\right) + a$$

$$\downarrow$$

$$(x-1)^2$$

$$\downarrow$$

$$T(2^k) = (c0 + c1 \cdot k) \cdot 1^k$$

$$\downarrow$$

$$T(n) = c0 + c1 \cdot log(n)$$

$$\downarrow$$

$$T(n) \in \mathcal{O}(\log(n))$$

Búsqueda binaria. Eficiencia empírica

Búsqueda binaria	
Elementos (n)	**Tiempo (s)**
1760000	0.000000402733
2520000	0.0000005118
3280000	0.000000472
4040000	0.000000538667
4800000	0.0000006558
5560000	0.0000006632
6320000	0.000000618467
7080000	0.0000005378
7840000	0.000000617267
8600000	0.000000618667
9360000	0.0000007254
10120000	0.000000638133
10880000	0.0000006072
11640000	0.00000071
12400000	0.000000569667
13160000	0.0000006822
13920000	0.000000631667
14680000	0.000000569333
15440000	0.000000697867
16200000	0.0000005758
16960000	0.00000069
17720000	0.000000623667
18480000	0.000000644133
19240000	0.0000007254
20000000	0.000000673533

Búsqueda binaria. Eficiencia híbrida

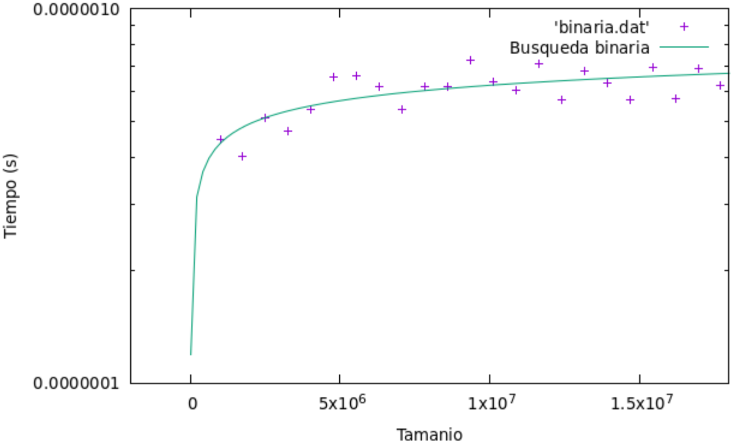

Figura 2: Gráfica con los tiempos de ejecución de la búsqueda binaria

Búsqueda binaria. Fuerza bruta vs Divide y Vencerás

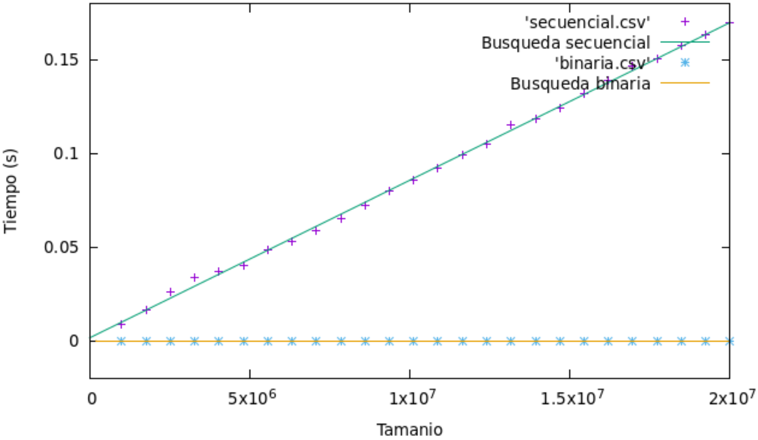

Figura 3: Gráfica comparativa: Fuerza Bruta vs DyV sin repeticiones

Búsqueda binaria. Fuerza bruta vs Divide y Vencerás

Las expresiones del tiempo de cada algoritmo son:

Fuerza bruta $\longrightarrow T(n) = 8.41755 \cdot 10^{-9} n + 0.00153755$

DyV sin repeticiones $\longrightarrow T(n) = 5.63832 \cdot 10^{-8} \cdot \log_2(n) - 6.87177 \cdot 10^{-7}$.

E igualando las expresiones obtenemos que: **Umbral:** $n = 1$

¿Elementos repetidos?

¿Qué pasaría si tuviésemos elementos repetidos? Por ejemplo:

1 2 3 4 4 5 6 7

¿Elementos repetidos?. Solución

```
<<<<<<< HEAD
int buscarBinaria(int v[], int inicio,int fin){
    int medio = (inicio + fin)/2; // O(1)
    int resultado = -1; // O(1)

    if(v[medio] == medio){ // O(1)
        return medio; // O(1)
    }
    else{
        if(inicio <= fin){ // O(1)
            resultado = buscarBinaria(v, inicio, medio - 1); // O(n/2)

            if(resultado == -1){
                resultado = buscarBinaria(v, medio + 1, fin); // O(n/2)
            }
        }
    }

    return resultado; // O(1)
```

¿Elementos repetidos?. Eficiencia teórica

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + a$$

$$\downarrow$$

$$(x-1)(x-2)$$

$$\downarrow$$

$$T(2^k) = c0 + c1 * 2^k$$

$$\downarrow$$

$$T(n) = c0 + c1 * n$$

$$\downarrow$$

$$T(n) \in \mathcal{O}(n)$$

¿Elementos repetidos?. Eficiencia empírica

Divide y Vencerás con repeticiones	
Elementos (n)	**Tiempo (s)**
1760000	0.020179
2520000	0.0220417
3280000	0.0344659
4040000	0.0398963
4800000	0.0443348
5560000	0.0502432
6320000	0.0558109
7080000	0.0592223
7840000	0.0630519
8600000	0.0698851
9360000	0.0772074
10120000	0.0808893
10880000	0.0893751
11640000	0.0940162
12400000	0.0992901
13160000	0.103868
13920000	0.116623
14680000	0.118174
15440000	0.12476
16200000	0.131296
16960000	0.145325
17720000	0.162416
18480000	0.170681
19240000	0.177497
20000000	0.185571

¿Elementos repetidos?. Eficiencia híbrida

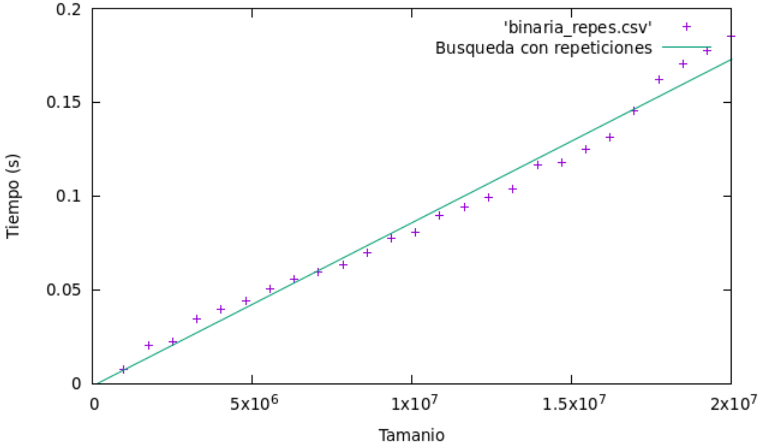

Figura 4: Gráfica con los tiempos de ejecución de la búsqueda con repeticiones

¿Elementos repetidos?. Fuerza bruta vs DyV con repeticiones

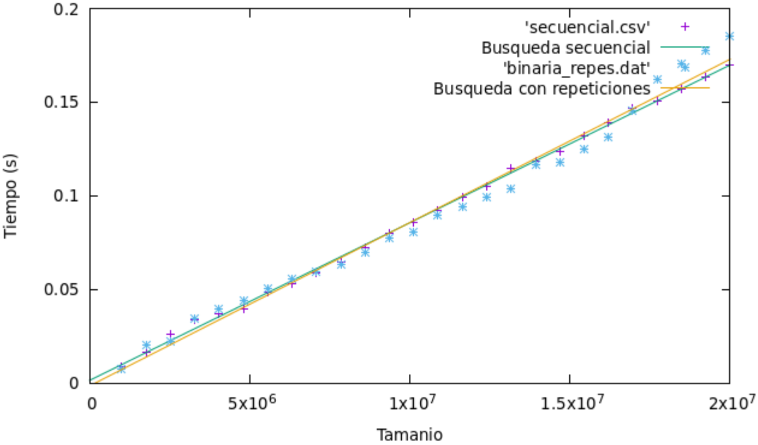

Figura 5: Gráfica comparativa: Fuerza Bruta vs DyV con repeticiones

¿Elementos repetidos?. Fuerza bruta vs DyV con repeticiones

$$\text{Fuerza bruta} \longrightarrow T(n) = 8.41755 \cdot 10^{-9} n + 0.00153755.$$

$$\text{DyV con repeticiones} \longrightarrow T(n) = 8.71886 \cdot 10^{-9} \cdot n - 0.00140853.$$

$$\downarrow$$

¡La pendiente del algoritmo de fuerza bruta es menor que la del “Divide y Vencerás”!

Ejercicio 2

Mezcla secuencial

Es la forma más simple de mezclar k vectores. Copias el primer vector y después vas añadiendo cada elemento de los sucesivos vectores en su posición correspondiente.

Mezcla secuencial. Código

```
void mergeKArrays(int nElementos, int **arr, int nVectores, int * &v_resultante)
{

    int k = 0;
    bool encontrado;
    for(int i = 0; i < nElementos; i++){    //O(n)
      v_resultante[i] = arr[0][i];
    }

    //iteramos por cada vector y por cada elemento del nuevo vector a insertar
    for(int i = 1; i < nVectores; i++){          //O(k)
      for(int j = 0; j < nElementos; j++){      //O(n)
        encontrado = false;
        k = 0;
        while(k < nElementos * i + j && !encontrado){    //O(kn + n)
          if(v_resultante[k] > arr[i][j])
            encontrado = true;
          else
```

Mezcla secuencial. Eficiencia teórica

Observamos como se indica en los comentarios que

$$T(n) \in \mathcal{O}(k^2 n^2)$$

Mezcla Divide y Vencerás

Aplicamos el método Divide y Vencerás, donde aplicamos una función recursiva en la que dividimos el array de vectores de entrada en 2 arrays de tamaño la mitad, la función se llama a sí misma y, finalmente, se mezclan ambos arrays. El caso base es que el array contenga un único vector y se copia en el resultado final. Tras resolver la ecuación recurrente asociada

$$T(n) = \begin{cases} n & si \quad n = 1 \\ \\ 2T(\frac{n}{2}) + c2k \cdot n & si \quad n > 1, n = 2^z \end{cases}$$

que finalmente resultará

$$T(n) = n^2 + c2 \cdot \log_2(n) \cdot k \cdot n$$

Mezcla Divide y Vencerás. Código

```
void mergeKArrays(int n, int **arr, int n1,int n2, int * &
     array_resultante)
{
     //si solo hay un array
     if(n1==n2){                              //O(n)
          for(int i=0; i < n; i++)
          array_resultante[i]=arr[n1][i];
     }
     else{

       int nVect = n2-n1+1;
       int mitad = (n2+n1)/2;

       //Dimensiones arrays auxiliares
       int tam2 = nVect/2;
       int tam1 = nVect - tam2;

          //Arrays resultantes

          int *array1 = nullptr;
          reservarArray(n*(tam1), array1);
```

Mezcla Divide y Vencerás. Eficiencia teórica

Observamos por lo indicado en la diapositiva anterior que si $M = max\left\{n^2, \log_2(n) \cdot k \cdot n\right\}$ nuestra función es

$$T(n) \in \mathcal{O}(M)$$

Conclusiones

- El uso de la técnica “Divide y Vencerás” no siempre es garantía de mejora respecto al uso del algoritmo de fuerza bruta.
- En aquellos casos en los que el uso de “Divide y Vencerás” sí nos ayuda a mejorar los tiempos, es importante saber que el algoritmo de fuerza bruta es preferible si se usan tamaños por debajo del umbral.
- EL uso de la recursividad requiere un uso excesivo de la pila y en algunos casos, esto da lugar a algoritmos ineficientes.