

Divide y Vencerás

Algorítmica. Práctica 2

Jose Alberto Hoces Castro Javier Gómez López Moya Martín Castaño

Contenidos

1. Introducción

2. Ejercicio 1

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++) //0(n)
    {
        if (v[i] == i){ //0(1)
            return i; //0(1)
        }
    }
    return -1;//0(1)
}</pre>
```

Búsqueda secuencial. Eficiencia teórica

Observamos claramente que

$$T(n) \in O(n)$$

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

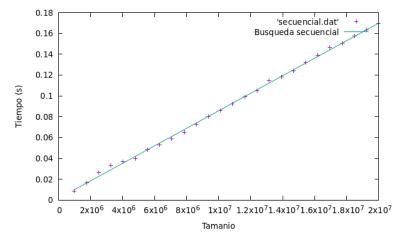


Figura 1: Gráfica con los tiempos de ejecución de la búsqueda a fuerza bruta

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

```
1 <<<<<< HFAD
int buscarBinaria(int *v, int inicio, int fin){
      if(fin >= inicio) \{ // 0(1) \}
3
          int medio = inicio + (fin - inicio) / 2; // 0(1)
4
5
          if(v[medio] == medio) { // 0(1)}
6
               return medio; // 0(1)
8
9
          if(v[medio] > medio) { // 0(1)}
10
               return buscarBinaria(v, inicio, medio - 1); // 0(n
11
      /2)
12
13
          //else
14
          return buscarBinaria(v, medio + 1, fin); // O(n/2)
      }
16
      return -1; // 0(1)
18
19
```

Búsqueda binaria. Eficiencia teórica

Observamos claramente que

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

$$\downarrow \qquad \qquad \downarrow \qquad \qquad \downarrow$$

$$(x-1)^2 \qquad \qquad \downarrow$$

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

$$\downarrow \qquad \qquad \downarrow$$

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

$$\downarrow \qquad \qquad \downarrow$$

$$T(n) \in O(\log(n))$$

Búsqueda binaria. Eficiencia empírica

Elementos (n) Tiempo (s) 1760000 0.000000402733 2520000 0.0000004723 3280000 0.000000472 4040000 0.000000538667 4800000 0.0000006558 5560000 0.0000006632 6320000 0.000000618467 708000 0.000000617267 8600000 0.000000617267 8600000 0.000000617267 10120000 0.000000638133 10880000 0.000000672 1164,0000 0.000000672 1360000 0.00000059667 1360000 0.000000698133 1544,0000 0.000000698133 1544,0000 0.000000697867 16200000 0.000000697867 17720000 0.000000623667 18,480000 0.000000623667 18,480000 0.000000623667 200000000 0.00000067553	Búsqueda binaria	
252000 0.000005118 3280000 0.0000005118 3280000 0.000000538667 4800000 0.000000538667 4800000 0.00000058667 5560000 0.0000006558 5560000 0.000000618467 7080000 0.000000617267 8600000 0.000000618667 9360000 0.000000618667 9360000 0.000000618667 11640000 0.000000672 11240000 0.000006672 13920000 0.000006692 13920000 0.000006692 13920000 0.000006692 13960000 0.00000659667 16200000 0.000000659667 16200000 0.000000659833 154440000 0.000000659833 15440000 0.000000659833 15440000 0.000000659833 15440000 0.000000659833 15440000 0.000000659833	Elementos (n)	Tiempo (s)
328000 0.00000472 4040000 0.000000558 480000 0.000000658 5560000 0.000000658 632000 0.000000638 67840000 0.00000067267 860000 0.0000067267 1012000 0.00000638133 1088000 0.00000672 1164000 0.00000672 1164000 0.00000572 1316000 0.000005978 1359000 0.00000572 1468000 0.00000569667 1316000 0.00000569667 136000 0.00000569667 136000 0.00000569667 1468000 0.0000056978 1544000 0.00000569333 1544000 0.00000569333 1544000 0.000005693867 1620000 0.00000697867 1620000 0.00000697867 1772000 0.00000669667	1760000	0.000000402733
404000 0.00000538667 480000 0.00000538667 480000 0.000006558 556000 0.0000006632 632000 0.000000618467 708000 0.00000617267 860000 0.00000617267 860000 0.00000618667 936000 0.0000007254 10120000 0.000000638133 10880000 0.000000677 11640000 0.000000599667 13160000 0.0000005822 13920000 0.000000631667 14680000 0.0000065783 15440000 0.0000065786 16200000 0.00000697867 16200000 0.00000697867 16200000 0.00000697867 16200000 0.00000697867 188480000 0.000000697867	2520000	0.0000005118
4,800000 0.000006558 5560000 0.0000006632 5520000 0.00000068467 7080000 0.0000005378 7840000 0.000000617267 8600000 0.000000617267 8600000 0.0000007254 10120000 0.00000068133 10880000 0.0000006772 11640000 0.000000569667 13160000 0.000000569667 13160000 0.000000631667 14680000 0.00000055933 15440000 0.000000578 16960000 0.000000578 16960000 0.000000697867 17720000 0.00000063667 184,80000 0.000000623667 184,80000 0.000000623667	3280000	0.000000472
556000 0.000006632 632000 0.00000618467 708000 0.00000618467 784000 0.00000617267 860000 0.00000617267 936000 0.00000618667 936000 0.00000638133 1088000 0.000006672 11640000 0.000006672 13160000 0.0000066822 13920000 0.000006681667 14680000 0.00000659667 15440000 0.00000659667 17720000 0.00000659667 16200000 0.00000659667 16200000 0.00000659667 16200000 0.00000659667 16200000 0.000000659667 16200000 0.00000659667	4040000	0.000000538667
632000 0.00000618467 7080000 0.00000617267 860000 0.00000617267 860000 0.00000617267 9360000 0.00000618667 9360000 0.00000638133 10880000 0.000006572 11640000 0.00000672 112400000 0.000006572 13160000 0.000006822 13920000 0.000006823 15440000 0.00000697867 16200000 0.00000697867 16200000 0.00000697867 16200000 0.00000697867 16200000 0.00000697867 16200000 0.00000697867 16200000 0.00000697867 18480000 0.00000663673	4800000	0.0000006558
7080000 0.000005378 7840000 0.000000517267 8600000 0.000000618667 9360000 0.000000754 10120000 0.000000754 11640000 0.000000671 12400000 0.00000059667 13160000 0.00000059667 14680000 0.000000597867 14620000 0.000005758 16960000 0.00000697 17720000 0.00000632433 19240000 0.00000634433	5560000	0.0000006632
784000 0.00000617267 860000 0.00000617267 860000 0.00000618667 936000 0.0000007254 10120000 0.000000638133 10880000 0.000000677 11640000 0.000000569667 13160000 0.000006822 13920000 0.000006832 14680000 0.00000659333 15440000 0.00000659333 15440000 0.000000659637 16200000 0.000000659667 16200000 0.000000659667 18480000 0.000000697867 18480000 0.000000634613 19240000 0.000000623667	6320000	0.000000618467
860000 0.00000618667 936000 0.000007254 10120000 0.00000638133 10880000 0.000006072 1164,0000 0.00000569667 13160000 0.000006822 13920000 0.00000631667 14680000 0.00000697867 16200000 0.0000065933 1544,0000 0.00000659637 16200000 0.00000659637 16200000 0.00000659637 16300000 0.00000659637 1640000 0.00000659637	7080000	0.0000005378
9360000 0.000007254 10120000 0.000000638133 10880000 0.0000006072 11640000 0.00000059667 13160000 0.000000569367 13920000 0.000000569333 15440000 0.000000569333 15440000 0.00000057867 16200000 0.0000005786 16960000 0.000000569 17720000 0.000000638433 19240000 0.000000624363	7840000	0.000000617267
10120000 0.000000638133 10880000 0.0000006072 11640000 0.00000071 12400000 0.000000569667 13160000 0.000000632 13920000 0.000000631667 14680000 0.000000599867 16200000 0.0000005788 16960000 0.0000006978 17720000 0.000000623667 18480000 0.000000644133 19240000 0.0000007254	8600000	0.000000618667
10880000 0.000006072 11640000 0.00000071 12400000 0.000000596967 13160000 0.000006822 13920000 0.00000631667 14680000 0.00000599867 16200000 0.0000005758 16960000 0.0000069 17720000 0.00000623667 18480000 0.00000644133 19240000 0.0000007254	9360000	0.0000007254
1164,000 0.0000071 124,0000 0.00000569667 1316000 0.000006822 1392000 0.000006832 1468000 0.0000056933 1544,000 0.000005786 1620000 0.000005758 1696000 0.00000623667 1772000 0.00000623667 184,8000 0.00000624313 1924,000 0.000007254	10120000	0.000000638133
12400000 0.000000569667 13160000 0.000006822 13920000 0.000000681667 14680000 0.00000057867 16200000 0.0000005758 16960000 0.00000069 17720000 0.000000623667 18480000 0.000000644133 19240000 0.0000007254	10880000	0.0000006072
13160000 0.000006822 13920000 0.000000631667 14680000 0.000000569333 15440000 0.0000005758 16200000 0.0000005758 16960000 0.000000623667 17720000 0.00000623667 18480000 0.00000644133 19240000 0.0000007254	11640000	0.00000071
13920000 0.000000631667 14680000 0.000000569333 15440000 0.000000697867 16200000 0.0000005758 16960000 0.00000069 17720000 0.00000623667 18480000 0.00000644133 19240000 0.0000007254	12400000	0.000000569667
14680000 0.00000569333 15440000 0.00000697867 16200000 0.000005758 16960000 0.0000069 17720000 0.00000623667 18480000 0.00000644133 19240000 0.0000007254	13160000	0.0000006822
15440000 0.000000697867 16200000 0.0000005758 16960000 0.0000069 17720000 0.000000623667 18480000 0.000000644133 19240000 0.0000007254	13920000	0.000000631667
16200000 0.0000005758 16960000 0.00000069 17720000 0.00000623667 18480000 0.00000644133 19240000 0.0000007254	14680000	0.000000569333
16960000 0.00000069 17720000 0.000000623667 18480000 0.000000644133 19240000 0.0000007254	15440000	0.000000697867
17720000 0.000000623667 18480000 0.000000644133 19240000 0.0000007254	16200000	0.0000005758
18480000 0.000000644133 19240000 0.0000007254	16960000	0.00000069
19240000 0.0000007254	17720000	0.000000623667
	18480000	0.000000644133
20000000 0.000000673533	19240000	0.0000007254
	20000000	0.000000673533

Búsqueda binaria. Eficiencia híbrida

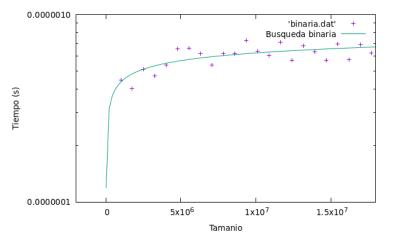


Figura 2: Gráfica con los tiempos de ejecución de la búsqueda binaria

Búsqueda binaria. Fuerza bruta vs Divide y Vencerás

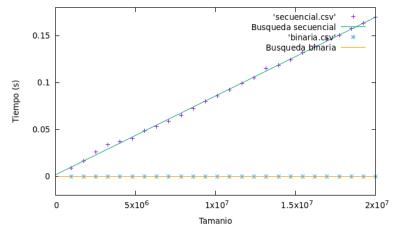


Figura 3: Gráfica comparativa: Fuerza Bruta vs DyV sin repeticiones

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

```
int buscarBinaria(int v[], int inicio,int fin){
      int medio = (inicio + fin)/2; // 0(1)
3
      int resultado = -1; // 0(1)
4
5
      if(v[medio] == medio) { // 0(1)}
6
          return medio; // 0(1)
8
      else{
9
          if(inicio <= fin){ // 0(1)
10
              resultado = buscarBinaria(v, inicio, medio - 1); //
11
      0(n/2)
12
              if(resultado == -1){
                  resultado = buscarBinaria(v, medio + 1, fin); //
14
       0(n/2)
15
16
17
18
```

¿Elementos repetidos?. Eficiencia teórica

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

$$T(n) = co + c1 \cdot n$$

$$\downarrow$$

$$T(n) \in O(n)$$

¿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

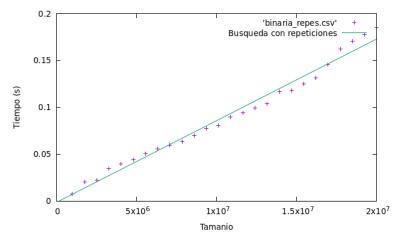


Figura 4: Gráfica con los tiempos de ejecución de la búsqueda con repeticiones

¿Elementos repetidos?. Fuerza bruta vs DyV con repeticiones

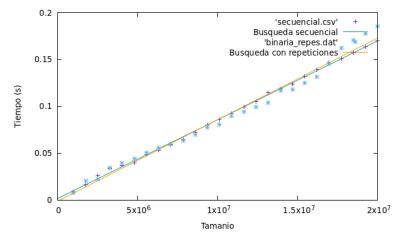


Figura 5: Gráfica comparativa: Fuerza Bruta vs DyV con repeticiones

¿Elementos repetidos?. Fuerza bruta vs DyV con repeticiones

Fuerza bruta
$$\longrightarrow T(n) = 8.41755 \cdot 10^{-9} n + 0.00153755$$
.
DyV con repeticiones $\longrightarrow T(n) = 8.71886 \cdot 10^{-9} \cdot n - 0.00140853$.

¡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)
2 {
3
      int k = 0:
4
      bool encontrado;
5
      for(int i = 0; i < nElementos; i++){ //0(n)
6
        v_resultante[i] = arr[0][i];
8
9
      //iteramos por cada vector y por cada elemento del nuevo
10
      vector a insertar
      for(int i = 1; i < nVectores; i++){ //0(k)
11
        for(int j = 0; j < nElementos; j++){
                                                   //0(n)
12
          encontrado = false;
          k = 0:
14
          while(k < nElementos * i + j && !encontrado){ //0(kn +</pre>
15
       n)
            if(v_resultante[k] > arr[i][j])
16
              encontrado = true;
```

Mezcla secuencial. Eficiencia teórica

Observamos como se indica en los comentarios que

$$T(n) \in O(k^2n^2)$$

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

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

que finalmente resultará

$$T(n) = n^2 + c_2 \cdot \log_2(n) \cdot k \cdot n$$

Mezcla Divide y Vencerás. Código

```
void mergeKArrays(int n, int **arr, int n1,int n2, int * &
      array_resultante)
2 {
      //si solo hay un array
3
      if(n1==n2){
                                          //0(n)
4
           for(int i=0; i < n; i++)</pre>
5
           array_resultante[i]=arr[n1][i];
6
      else{
8
9
         int nVect = n2-n1+1:
10
         int mitad = (n2+n1)/2;
11
12
         //Dimensiones arrays auxiliares
13
         int tam2 = nVect/2;
14
         int tam1 = nVect - tam2;
15
16
           //Arrays resultantes
17
18
           int *array1 = nullptr;
19
```

Mezcla Divide y Vencerás. Eficiencia teórica

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

$$T(n) \in O(M)$$

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.