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

```
1  int buscarSecuencial(int v[], int n){
2      for (size_t i = 0; i < n; i++) //O(n)
3      {
4          if (v[i] == i){ //O(1)
5              return i; //O(1)
6          }
7      }
8
9      return -1;//O(1)
10 }
```

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

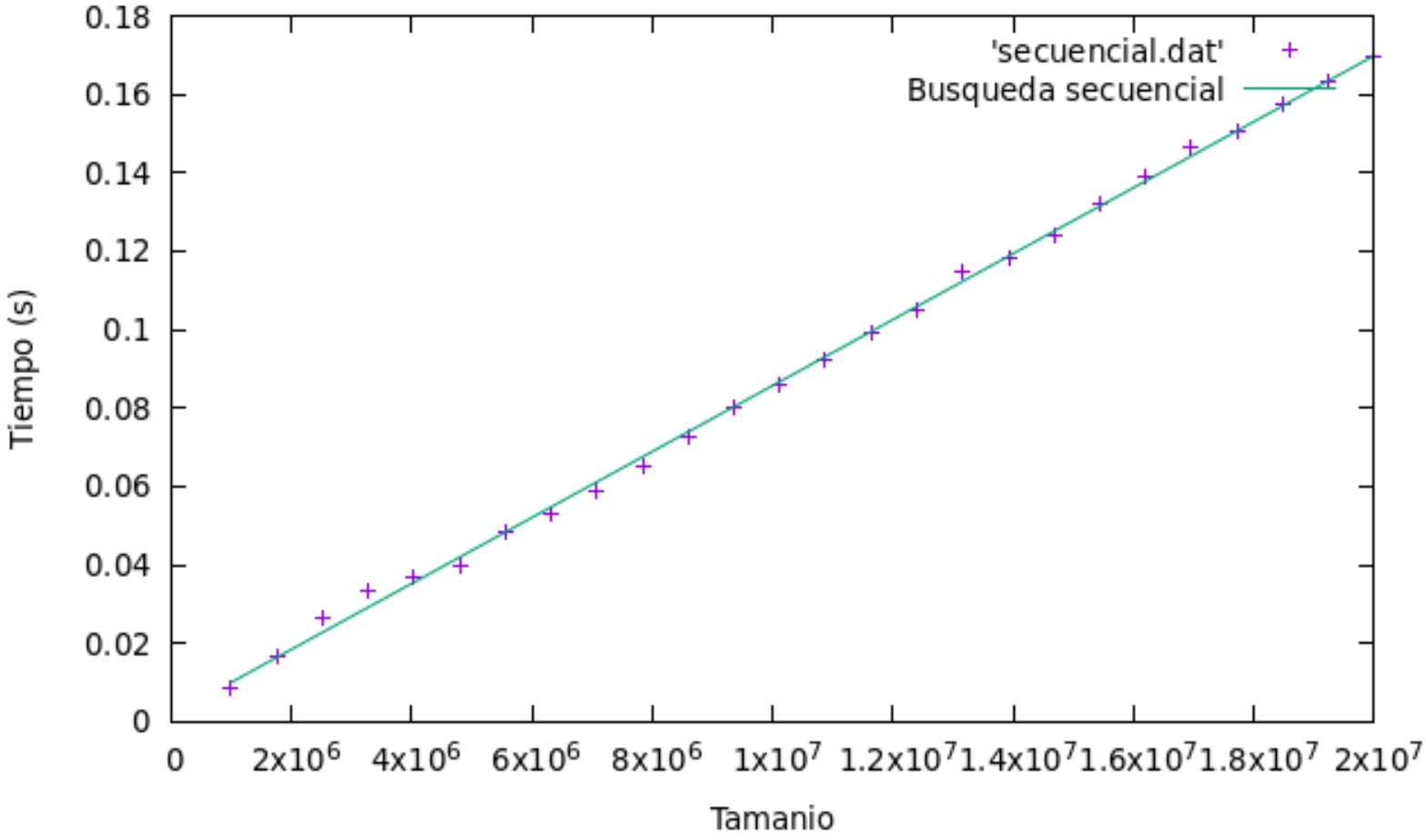


Figura 1: Gráfica con los tiempos de ejecución de la búsqueda a fuerza bruta

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  int buscarBinaria(int *v, int inicio, int fin){
2      if(fin >= inicio){ // O(1)
3          int medio = inicio + (fin - inicio) / 2; // O(1)
4
5          if(v[medio] == medio){ // O(1)
6              return medio; // O(1)
7          }
8
9          if(v[medio] > medio){ // O(1)
10             return buscarBinaria(v, inicio, medio - 1); // O(n
       /2)
11         }
12
13         //else
14         return buscarBinaria(v, medio + 1, fin); // O(n/2)
15     }
16
17     return -1; // O(1)
18 }
```

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

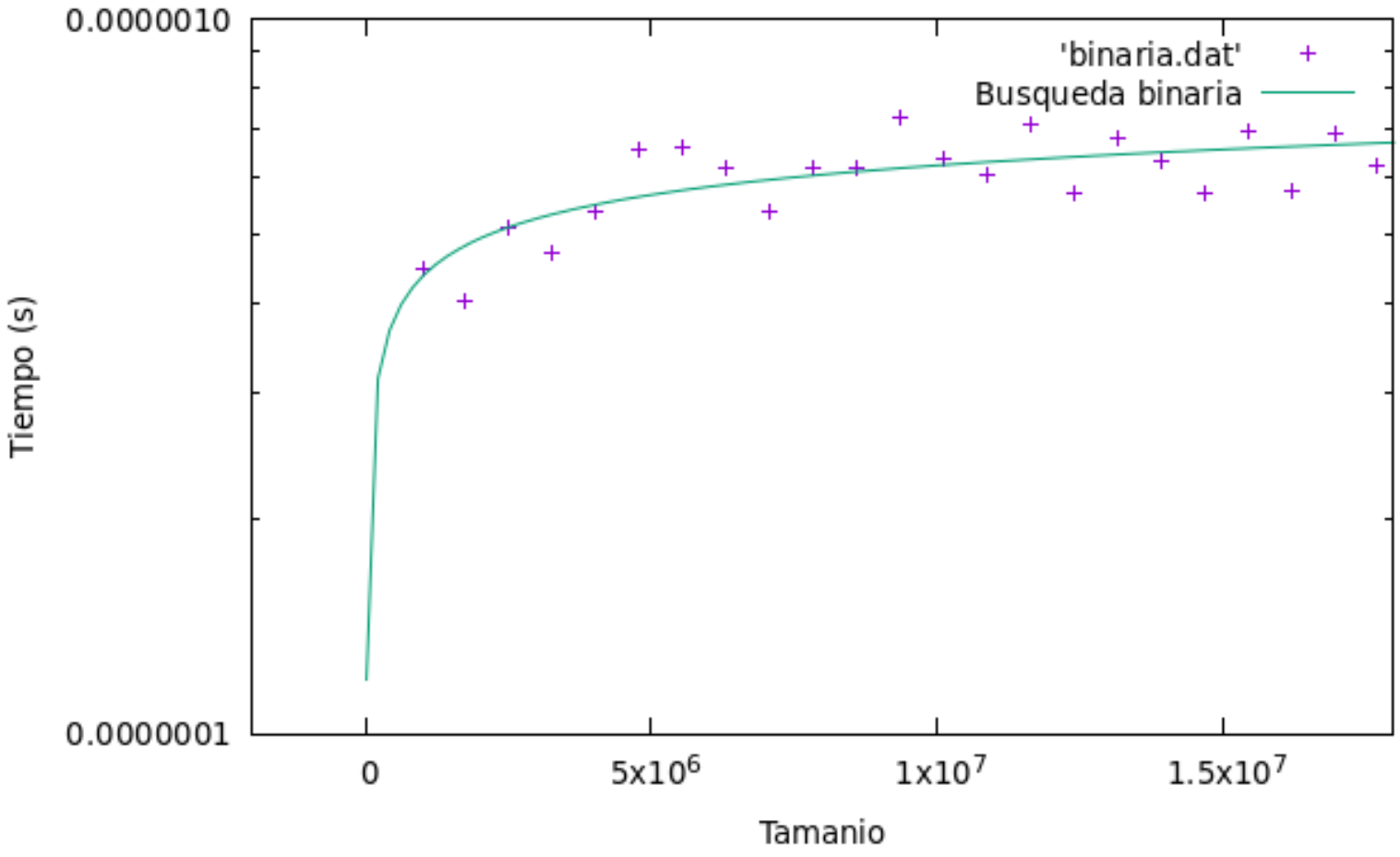


Figura 2: Gráfica con los tiempos de ejecución de la búsqueda binaria

Búsqueda binaria. Fuerza bruta vs Divide y Vencerás

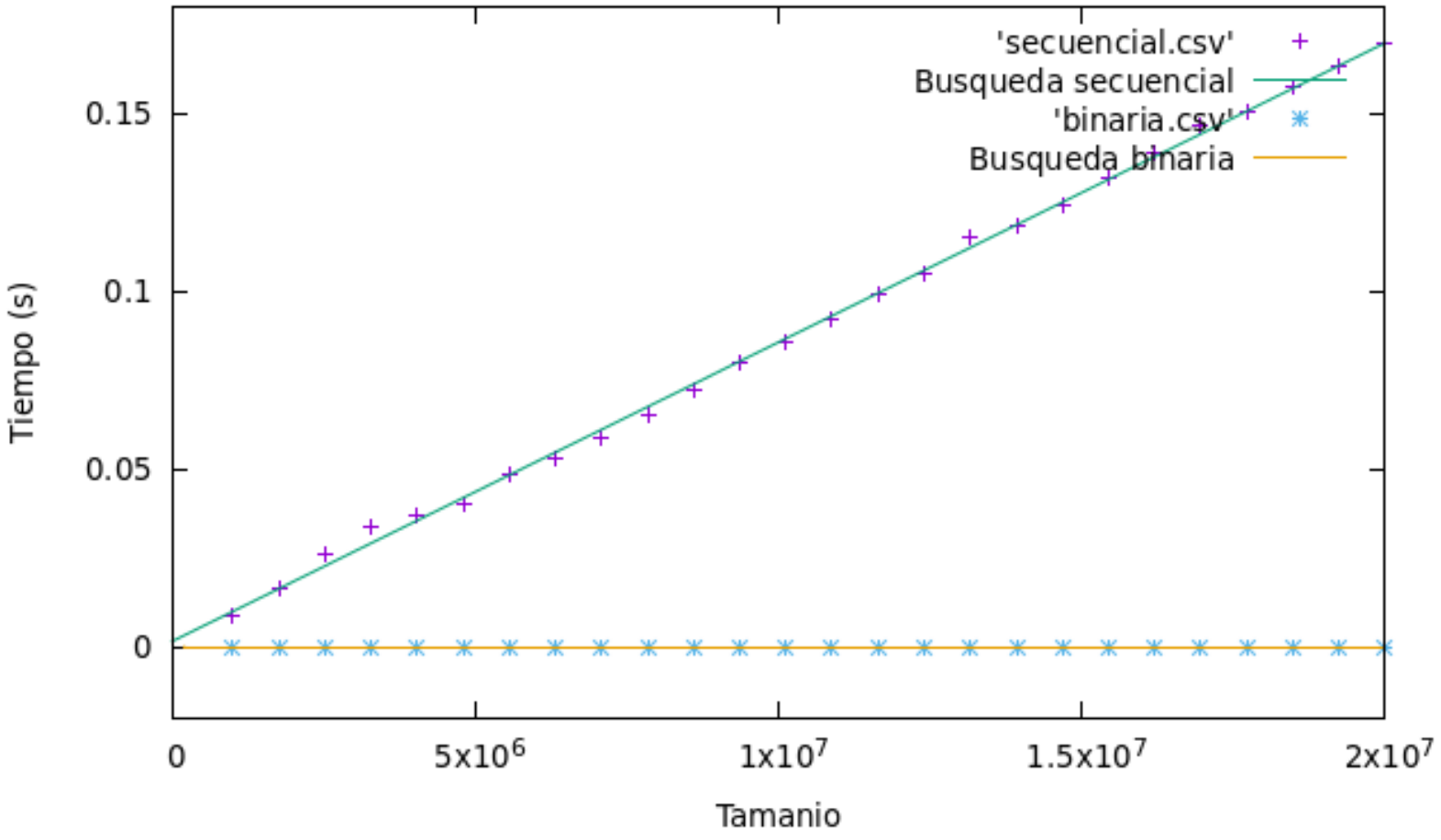


Figura 3: Gráfica comparativa: Fuerza Bruta vs DyV sin repeticiones

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

```
1  int buscarBinaria(int v[], int inicio,int fin){
2      int medio = (inicio + fin)/2; // O(1)
3      int resultado = -1; // O(1)
4
5      if(v[medio] == medio){ // O(1)
6          return medio; // O(1)
7      }
8      else{
9          if(inicio <= fin){ // O(1)
10             resultado = buscarBinaria(v, inicio, medio - 1); //
       O(n/2)
11
12             if(resultado == -1){
13                 resultado = buscarBinaria(v, medio + 1, fin); //
        O(n/2)
14             }
15         }
16     }
17
18     return resultado; // O(1)
```

¿Elementos repetidos?. Eficiencia teórica

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

¿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

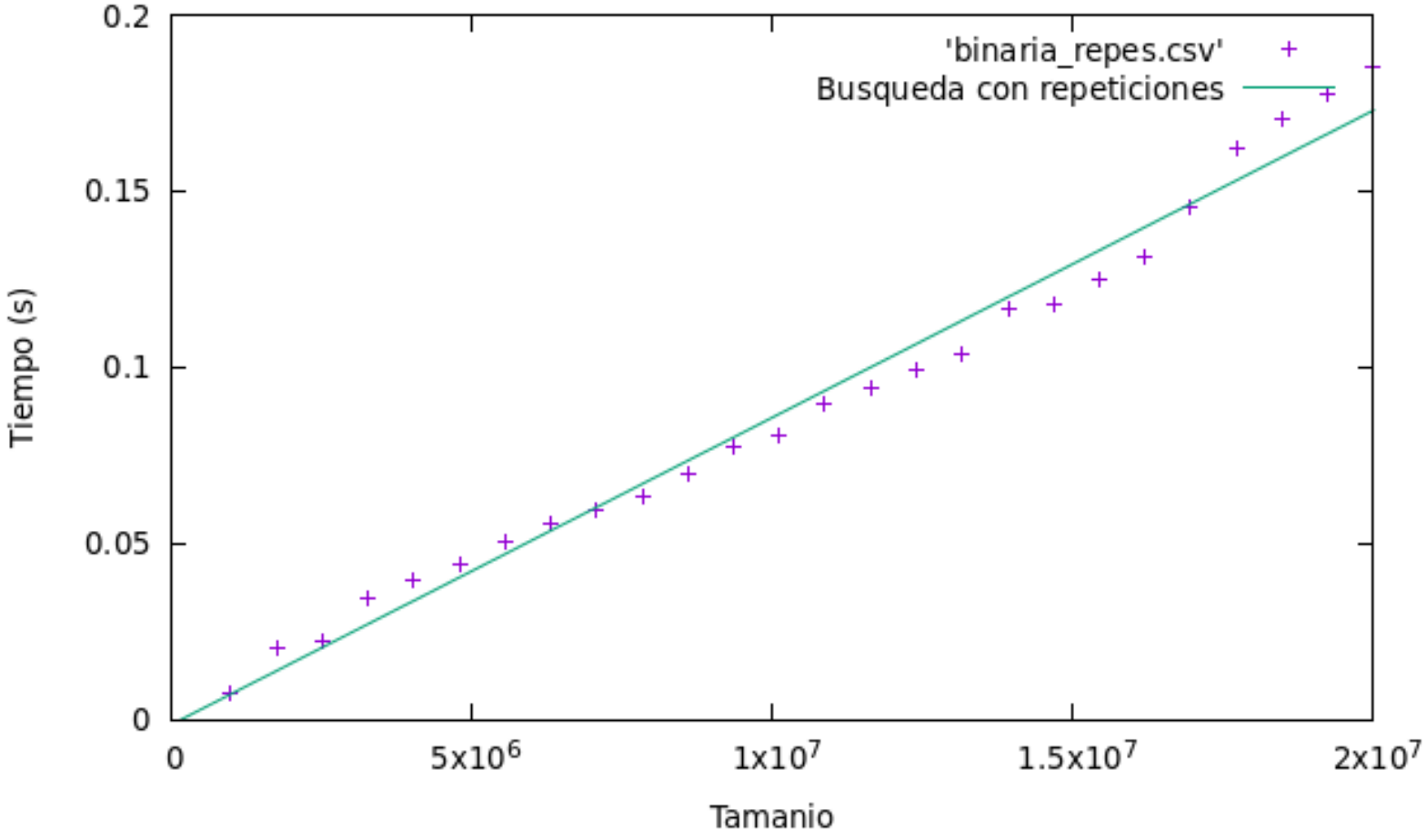


Figura 4: Gráfica con los tiempos de ejecución de la búsqueda con repeticiones

¿Elementos repetidos?. Fuerza bruta vs DyV con repeticiones

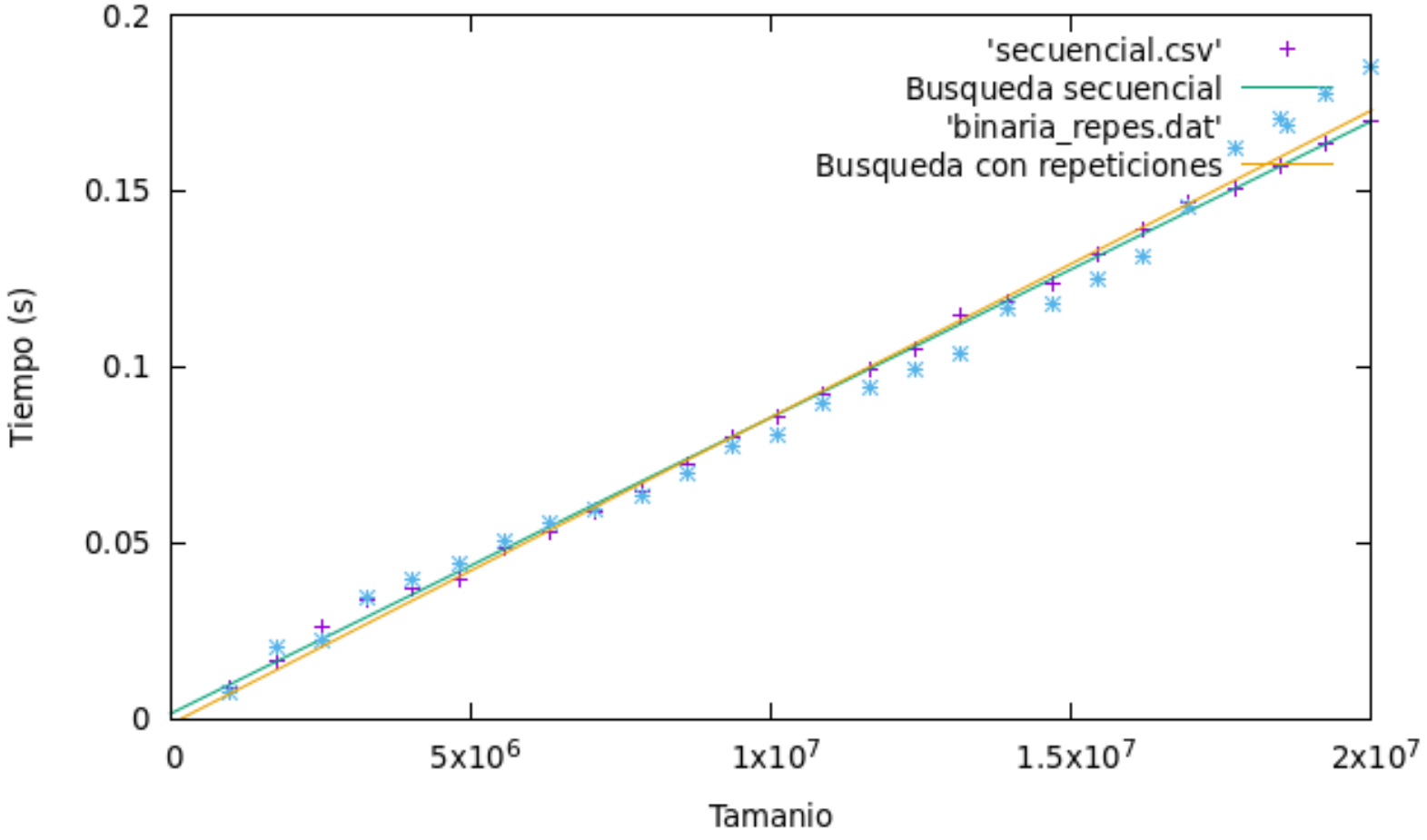


Figura 5: Gráfica comparativa: Fuerza Bruta vs DyV con repeticiones

¿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

Fuerza Bruta

Este algoritmo consiste en ir mezclando los dos primeros vectores, después mezclar el resultante con el tercero y así sucesivamente.

Fuerza Bruta. 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
            k++;
        }
        //Realizamos traslacion a la derecha para insertar
    elemento si es menor
        if(encontrado){
          for(int l = nElementos*i+j-1; l >= k; l--){   //O(kn)
            v_resultante[l+1] = v_resultante[l];
          }
          v_resultante[k] = arr[i][j];
        }
        else{
          v_resultante[nElementos * i + j ] = arr[i][j];
        }
      }
    }

}
```

Fuerza Bruta. Análisis Teórico

Vemos tras lo explicado que

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

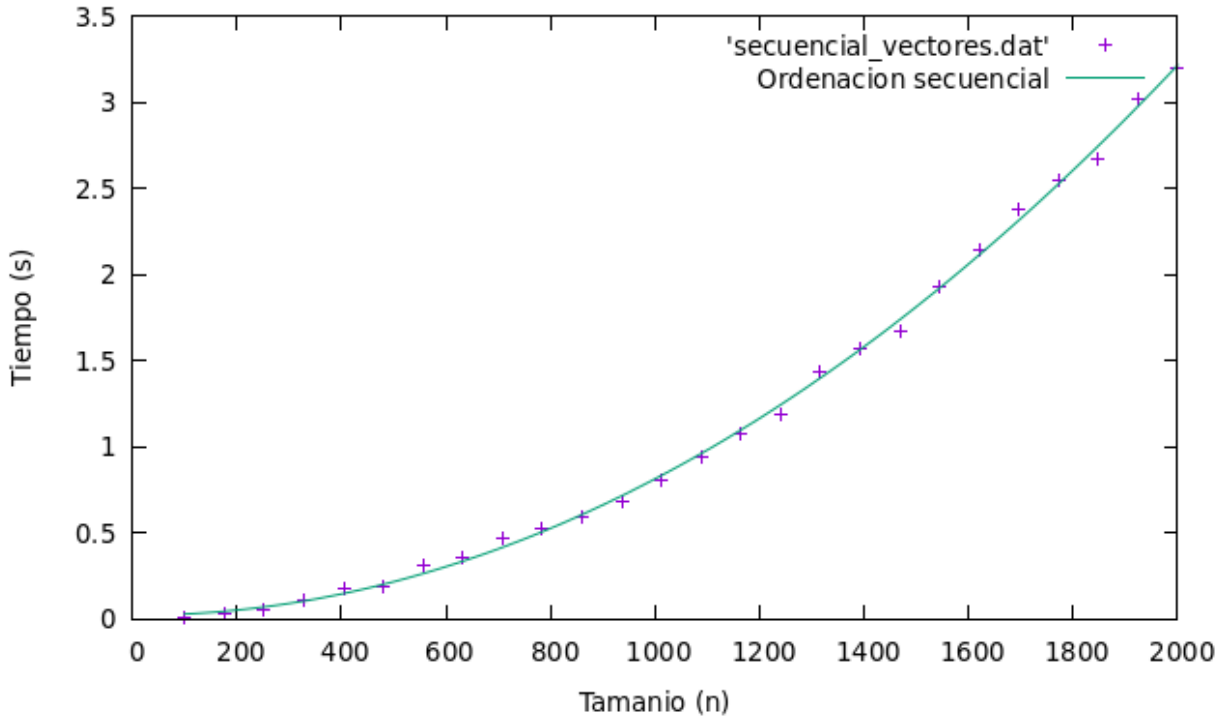
Fuerza Bruta. Análisis Empírico

Algoritmo de fuerza bruta fijado n	
Elementos (n)	**Tiempo (s)**
176	0.025546
252	0.0542358
328	0.110631
404	0.179092
480	0.192662
556	0.307523
632	0.355095
708	0.472475
784	0.528376
860	0.588945
936	0.679135
1012	0.810773
1088	0.935887
1164	1.07567
1240	1.19148
1316	1.43658
1392	1.57392
1468	1.67358
1544	1.92939
1620	2.13879
1696	2.38177
1772	2.54604
1848	2.67226
1924	3.0154
2000	3.19709

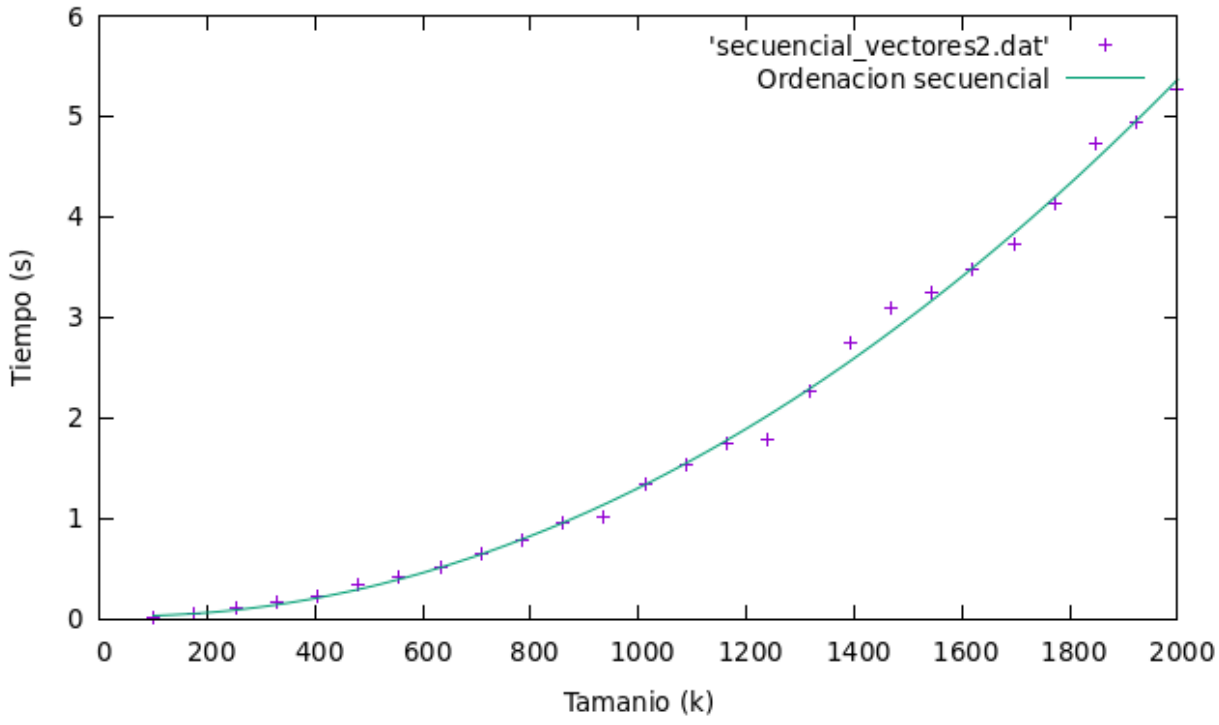
Algoritmo de fuerza bruta fijado k	
Elementos (n)	**Tiempo (s)**
176	0.0459797
252	0.10393
328	0.159313
404	0.231346
480	0.340936
556	0.417845
632	0.517876
708	0.649491
784	0.790378
860	0.961221
936	1.01366
1012	1.33613
1088	1.53368
1164	1.7476
1240	1.78693
1316	2.27582
1392	2.73955
1468	3.09484
1544	3.2492
1620	3.48283
1696	3.72723
1772	4.12856
1848	4.72144
1924	4.95091
2000	5.2639

Tabla 4: Experiencia empírica de algoritmo de Inserción sin optimizar

Fuerza Bruta. Análisis Híbrido



Fuerza Bruta. Análisis Híbrido



Divide y vencerás

Vamos creando en cada iteración 2 arrays auxiliares de la matriz de tamaño $\frac{k \cdot n}{2}$ y llamar a la función de nuevo para cada uno de los arrays.

Divide y vencerás. Análisis Teórico

La recursividad a plantear es

$$T(n) = 3 * T\left(\frac{n}{2}\right) + a$$

Resolviendo esta ecuación, obtenemos que

$$T(n) \in O(n^{\log_2(3)}) \qquad T(k) \in O(k^{\log_2(3)})$$

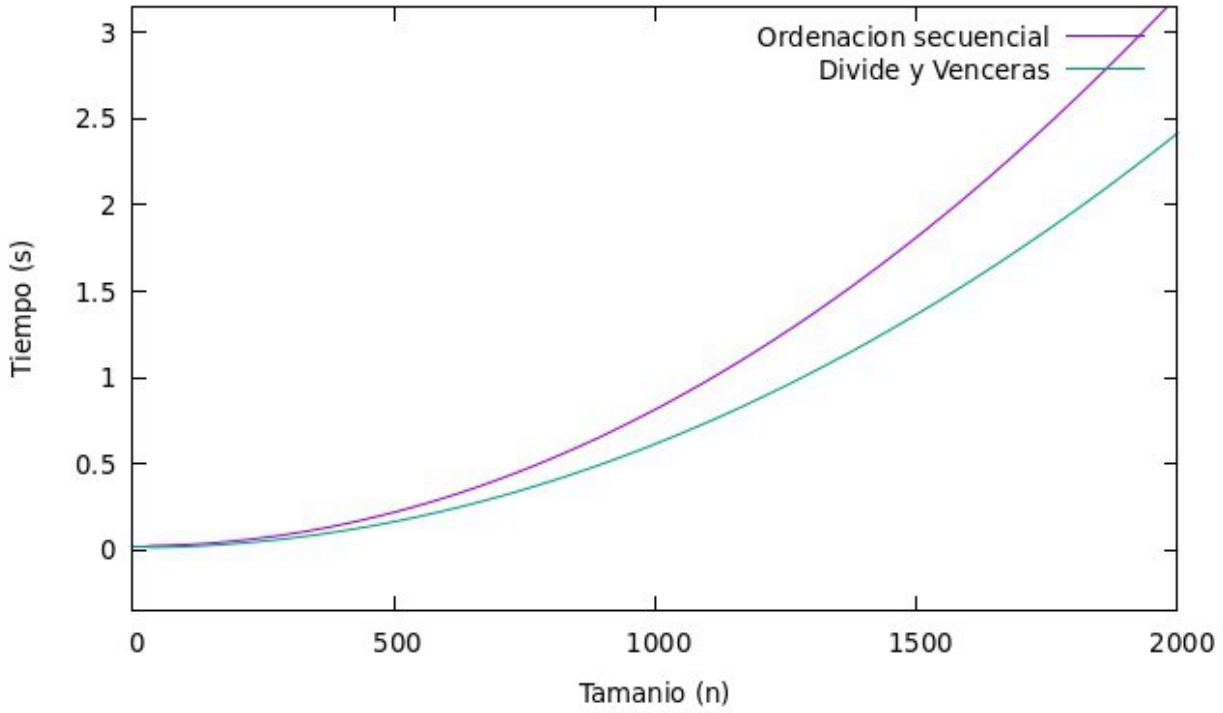
Divide y vencerás. Análisis Empírico

Divide y Vencerás fijado k	
Elementos (n)	**Tiempo (s)**
176	0.0355
252	0.0539
328	0.0823
404	0.1298
480	0.1566
556	0.1893
632	0.2341
708	0.2911
784	0.3463
860	0.3992
936	0.4528
1012	0.5013
1088	0.576
1164	0.7865
1240	0.8748
1316	0.9568
1392	1.1786
1468	1.2567
1544	1.4251
1620	1.6832
1696	1.854
1772	2.092
1848	2.1477
1924	2.3216
2000	2.4351

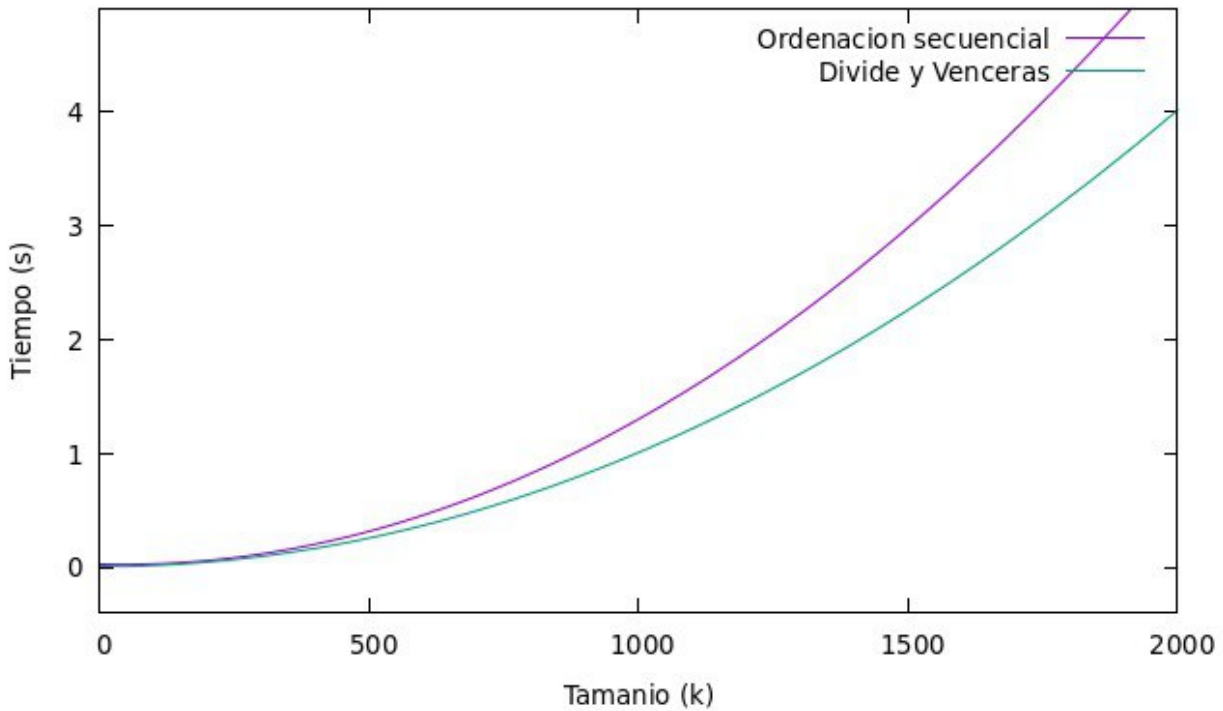
Divide y vencerás. Análisis Empírico

Divide y Vencerás fijado n	
Elementos (n)	**Tiempo (s)**
176	0.0596
252	0.0865
328	0.1133
404	0.2911
480	0.4478
556	0.5541
632	0.6012
708	0.7355
784	0.8129
860	0.9115
936	1.0218
1012	1.1244
1088	1.2152
1164	1.4891
1240	1.5771
1316	1.7632
1392	1.8914
1468	1.9788
1544	2.1257
1620	2.3554
1696	2.783
1772	2.961
1848	3.3428
1924	3.7213
2000	3.9121

Divide y Vencerás. Análisis Híbrido



Divide y Vencerás. Análisis Híbrido



Conclusiones

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.