Sub-Array de suma máxima

Daniel Daher Pérez
Diego Luis Afonso

Índice

- Introducción y Descripción del problema
- Pseudocódigo
- Análisis del algoritmo
- Experimentos
- Conclusiones

Introducción y Descripción del problema

Problema del SubArray de suma máxima

- Este problema pretende encontrar el subvector dentro de un vector en el que la suma de sus números sea la máxima
- 2 resoluciones al problema
  - Fuerza Bruta
  - Divide y Vencerás

Pseudocódigo

Fuerza Bruta

```
Array maxSubArrayFB(Array array){
    max, inic, fin = 0;

    para (i = 0 in array.size(){
        suma = 0;
        para (j = i in array.size()){
            suma += array.get(j);
            si (suma > max){
                max = suma;
                inic = i;
                fin = j;
            }
        }
    }

    return array[inic-to-fin];
}
```

Divide y Vencerás

```
SubArray maxSubArrayD&C(Array array, inic, fin){
     medio = 0;

     si (inic == fin)
          return Array(inic, fin, array[inic]);

     si no {
          medio = floor((inic + fin) / 2);

          SubArray izq = maxSubArrayD&C(array, inic, medio);
          SubArray der = maxSubArrayD&C (array, medio+1, fin);
          SubArray cruce = maxSubArrayCruce(array, inic, medio, fin);

          si(izq.getSuma() >= der.getSuma() && izq.getSuma() >= cruce.getSuma())
               return new SubArray(izq.getInic(), izq.getFin(), izq.getSuma());
          si(der.getSuma() >= izq.getSuma() && der.getSuma() >= cruce.getSuma())
               return new SubArray(der.getInic(), der.getFin(), der.getSuma());
          si no
               return new SubArray(cruce.getInic(), cruce.getFin(), cruce.getSuma());

     }
}
```

Cruce

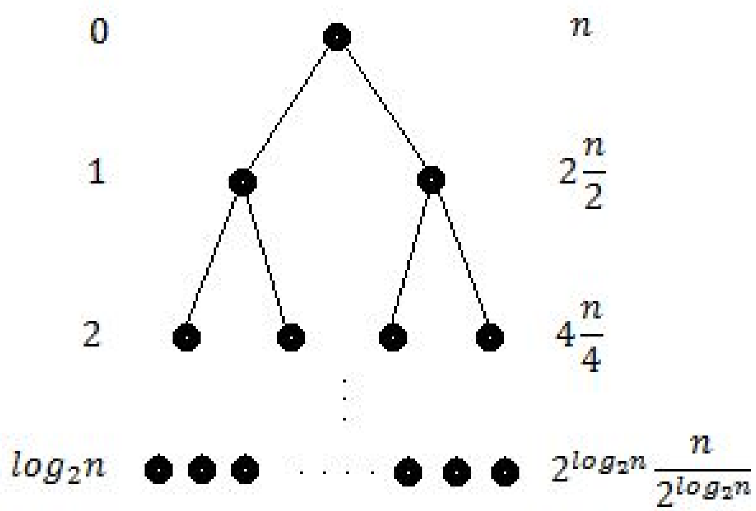
```
SubArray maxSubArrayCruce(Array array, inic, medio, int fin){
     izqSuma = -infinito;
     suma = 0;
     izqIndice = 0;
     para (i = medio; i >= inic; i--){
          suma += array[i];
          si (suma > izqSuma){
               izqSuma = suma;
               izqIndice = i;
          }
     }
     derSuma = -infinito;
     suma = 0;
     derIndice = 0;
     para (i = medio + 1; i <= fin; i++){
          suma += array[i];
          si (suma > derSuma){
               derSuma = suma;
               derIndice = i;
          }
     }
     return new SubArray(izqIndice, derIndice, derSuma + izqSuma);
}
```

Análisis del algoritmo

$$T(n) = 2T(n/2) + \Theta(n)$$

- $a = 2$
- $b = 2$
- $d = 1$

$$T(n) = \Theta(n\ logn)$$

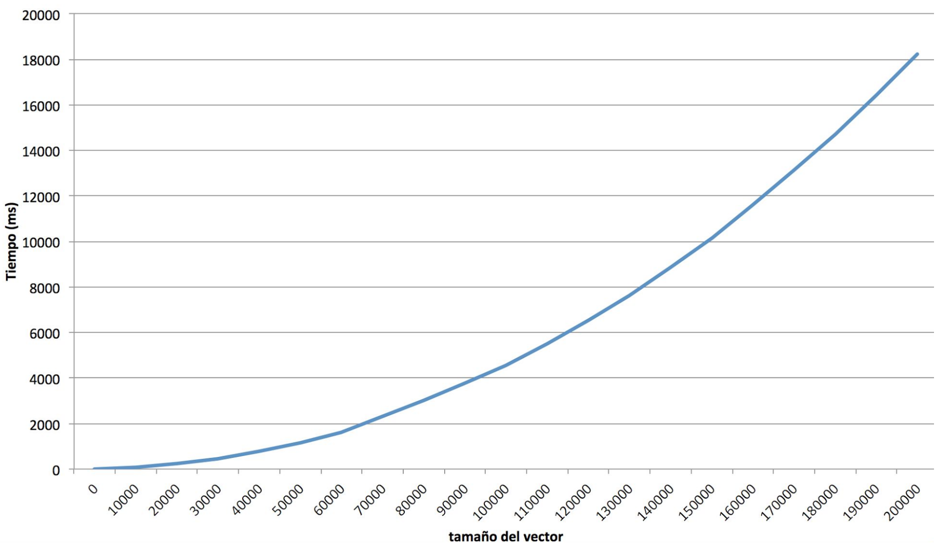


$$T(n) = \sum_{i=0}^{log_2 n} n = n\ log_2 n$$

$$T(n) = \Theta(n\ logn)$$

Experimentos

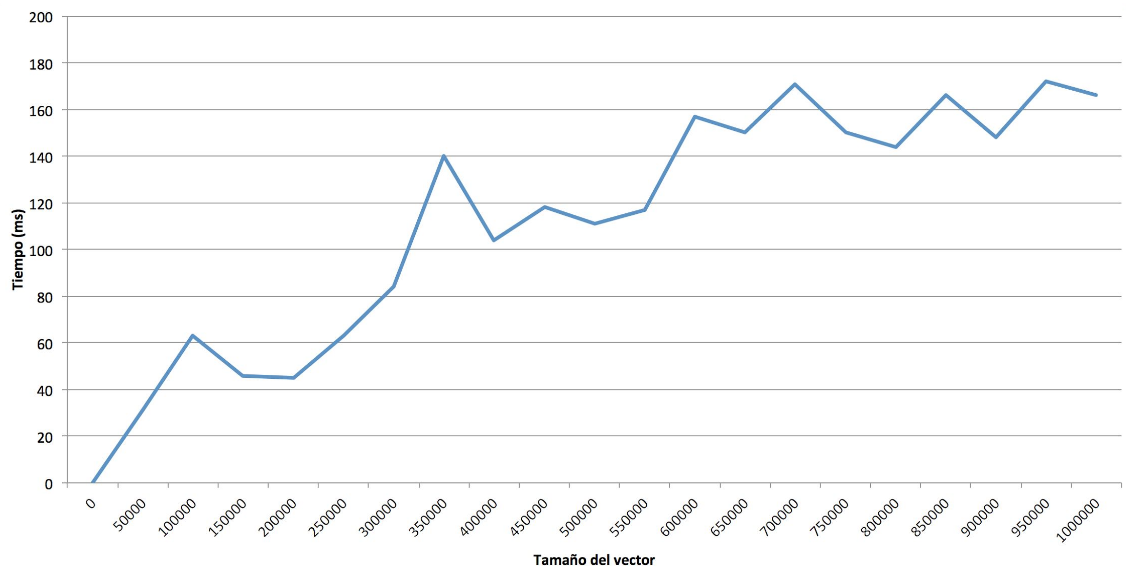
Fuerza Bruta



Tamaño vector	Tiempo (ms)
0	0
10000	61
20000	258
30000	464
40000	770
50000	1160
60000	1596
70000	2315
80000	3031
90000	3746
100000	**4542**
120000	6526
140000	8856
160000	11606
180000	14711
200000	18248

Experimentos

Divide y Vencerás



Tamaño de entrada	Tiempo (ms)
0	0
50000	31
100000	63
150000	46
200000	45
250000	63
300000	84
350000	140
400000	104
450000	118
500000	111
600000	157
700000	171
800000	144
900000	148
1000000	166

Conclusiones

“

- Para un resultado no trivial siempre será necesaria la existencia de enteros **negativos** en el array

- Fuerza Bruta -> alto coste.

- El algoritmo de *Divide & Conquer* es la mejor solución a este problema, encuentra la solución en un orden $T(n) = O(n\ logn)$.

- *D&C* mejora en mucho el desempeño del *Brute Force*.

- Los algoritmos no siempre dan la misma solución en el caso de que haya varios subarrays del mismo tamaño.