Examen de Algorítmica y Complejidad (Plan 2014)

25 de enero de 2023

Nº matrícula:_____ **Nombre:** _____

Apellidos: _____

Problema. Dado un *array* de números enteros positivos y negativos, se desea encontrar la suma máxima de cualquiera de sus subarrays[1] formado sólo por números positivos. Ejemplo:

0	1	2	3	4	5	6	7	8
-2	**5**	**1**	**-3**	**-2**	**4**	**7**	**-1**	**9**

En este ejemplo, la suma máxima del subarray de números positivos es 11 (*subarray* 4 y 7).

a) Diseñar un algoritmo basado en ***Divide y Vencerás***[2] con complejidad O(N·log N) en el caso peor[3] (donde N es el tamaño del *array*) que devuelva la suma máxima del *subarray* pedido. Si todos los elementos del vector son negativos se deberá devolver 0.

```
int maxSubArrayPositivos(int[] vector)
```

```
int maxSubArrayPositivos(int[] v){
    return maxSubArrayPositivosAux(v, 0, v.length-1);
}



int maxSubArrayPositivosAux(int[] v, int i0, int iN){
    if (i0==iN){
        if (v[i0]>0) return v[i0];
        else return 0;
    }
    else{
        int k=(i0+iN)/2;                                    // fase de división
        int maxIzq = maxSubArrayPositivosAux(v, i0, k);     // fase de…
        int maxDer= maxSubArrayPositivosAux(v, k+1, iN);    //  …conquista
        int maxCentral = maxSubArrayCentral(v, i0, k, iN); // fase de combinación
        return Math.max(maxIzq, Math.max (maxDer, maxCentral));
    }
}
```

[1] Dado v un array de longitud N y w un array de longitud $M \leq N$. Decimos que w es un subarray de v si y solo si $\exists k \in \{0,\ldots N\text{-}M\}$ tal que $\forall i \in \{0,\ldots M\text{-}1\}$ $v[k+i]=w[i]$.

[2] Desarrollar un algoritmo que no esté basado en la estrategia divide y vencerás conllevará una puntuación de 0 en todo el problema 1.

[3] Desarrollar un algoritmo con una complejidad diferente a O(Nlog N) en el caso peor conllevará una puntuación de 0 en la pregunta.

```java
int maxSubArrayCentral(int[] v, int i0, int k, int iN){
// Este método tiene complejidad O(N)
    int m = 0;
    if (v[k]<0) return 0;
    else{
        int i=k;
        while ((i>=i0) && (v[i]>0)){
            m = m + v[i];
            i--;
        }
        i=k+1;
        while ((i<=iN) && v[i]>0){
            m=m+v[i];
            i++;
        }
        return m;
    }
}
```

b) Justifica que la complejidad del algoritmo desarrollado en el apartado anterior para el caso peor es O(Nlog *N*).

El peor caso sucede cuando el vector está formado sólo por números positivos. Veamos la complejidad del algoritmo en el peor caso:

* La complejidad del algoritmo maxSubArrayCentral es O(N) en el peor caso ya que el número de iteraciones de los dos bucles while coincide con N, la longitud del vector.

* En este caso el algoritmo maxSubArrayPositivosAux obedece a la siguiente ecuación de recurrencia en el tiempo:

$$T(N)= 2\cdot T(N/2) + O(N) \text{ para } N>1$$

Esta ecuación es del tipo $T(N) = p \cdot T(N/q) + f(N)$, donde $f(N) \in O(N^a)$, con p=2, q=2 y a=1, por lo que podemos aplicar el Teorema Maestro. Dado que $\log_q(p) = \log_2(2)=1$ y a=1 nos encontramos en el caso 2º del Teorema maestro ($a=\log_q(p)$), por lo que la complejidad del algoritmo es: $T(N) \in O\left(N^{\log_q(p)} \cdot \log N\right) = O(N^1 \log N) = O(N\log N)$

* El algoritmo lmaxSubArrayPositivos tiene la misma complejidad que maxSubArrayPositivosAux. Por tanto, tiene complejidad O(N·log N).