

Nº matrícula: _____ Nombre: _____

Apellidos: _____

- 1) **Problema 1 (5 puntos).** Dado un *array* de números enteros positivos y negativos, se desea encontrar la suma máxima de cualquiera de sus subarrays¹ 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) (4 puntos) Diseñar un algoritmo basado en **Divide y Vencerás**² con complejidad $O(N \log N)$ en el caso peor³ (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 $\exists k \in \{0, \dots, N-M\}$ tal que $\forall i \in \{0, \dots, M-1\} v[k+i] = w[i]$.

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

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

```

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) (1 puntos) Justifica que la complejidad del algoritmo desarrollado en el apartado anterior para el caso peor es $O(N \log N)$.

La ecuación de recurrencia sería:

$$T(N) = \begin{cases} O(1) & N = 1 \\ 2T\left(\frac{N}{2}\right) + O(N^1) & N > 1 \end{cases}$$

Aplicando el Teorema Maestro,

- $p = 2$ (número de llamadas recursivas),
- $q = 2$ (número de divisiones),
- $a = 1$ (la complejidad de la fase de división es $O(1)$ y la complejidad de la fase de combinación es de $O(N) = O(N^a) \rightarrow a=1$).

$\lg_q p = \log_2 2 = 1$.

$a = \log_q p \rightarrow$ estamos en el caso 2 del Teorema Maestro, por lo que:

$$T(N) \in O(N^{\log_q p} \log N) = O(N^1 \log N) = O(N \log N)$$