



Nº matrícula: \_\_\_\_\_ Nombre: \_\_\_\_\_

Apellidos: \_\_\_\_\_

- 1) **Problema 1 (5 puntos).** Dado un *array* de números enteros, encontrar la longitud del subarray<sup>1</sup> ordenado más largo. Ejemplo:

0	1	2	3	4	5	6	7	8
2	1	1	3	2	4	7	15	1

En este ejemplo, la longitud del *subarray* ordenado más largo es 4 (*subarray* 2, 4, 7 y 15).

- a) **(3.5 puntos)** Diseñar un algoritmo basado en *Divide y Vencerás*<sup>2</sup> con complejidad  $O(N \log N)$  en el caso peor<sup>3</sup> (donde  $N$  es el tamaño del *array*) que devuelva la longitud del *subarray* pedido.

`int longMaxSubArrayOrdenado(int[] vector)`

```
int longMaxSubArrayOrdenado(int[] vector){
    return longMaxSubArrayOrdenadoAux(vector, 0, vector.length-1);
}

int longMaxSubArrayOrdenadoCruzada(int[] vector, int i0, int k, int iN){
    int i=k;
    while (i>i0 && vector[i-1]<=vector[i]) {
        i--;
    }
    int j=k;
    while (j<iN && vector[j+1]>=vector[j]) {
        j++;
    }
    return j-i+1;
}

int longMaxSubArrayOrdenadoAux(int[] vector, int i0, int iN){
    if (i0==iN)
        return 1;
    else {
        int k = (i0 + iN) / 2;
        int m1 = longMaxSubArrayOrdenadoAux(vector, i0, k);
        int m2 = longMaxSubArrayOrdenadoAux(vector, k + 1, iN);
        int m3 = longMaxSubArrayOrdenadoCruzada(vector, i0, k, iN);
        return Math.max(m1, Math.max(m2, m3));
    }
}
```

- b) **(1.5 puntos)** Identifica el peor caso y justifica que la complejidad del algoritmo desarrollado en el apartado anterior para el caso peor es  $O(N \log N)$ .

<sup>1</sup> 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  $\forall i \in \{0, \dots, M-1\} v[k+i] = w[i]$ .

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

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

El peor caso sucede cuando **el vector está ordenado**. Veamos la complejidad del algoritmo en el peor caso:

- La complejidad del algoritmo `longMaxSubArrayOrdenadoCruzada` es  $\Theta(N)$  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 tiempo que tarda el algoritmo `longMaxSubArrayOrdenadoAux` es:

$$T(N) = 2 \cdot T(N/2) + \Theta(N)$$

Por lo tanto, nos encontramos en el caso 2 del teorema maestro y el algoritmo tiene complejidad  $\Theta(N \log N)$

- El algoritmo `longMaxSubArrayOrdenado` tiene la misma complejidad que `longMaxSubArrayOrdenadoAux`. Por tanto, tiene complejidad  $\Theta(N \log N)$ .