



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

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

- 1) **Problema 1 (5 puntos).** Decimos que un *array*,  $v$ , de  $N$  enteros está ordenado circularmente si, o bien el vector está ordenado, o bien  $v[N-1] \leq v[0]$  y  $\exists k$  con  $0 < k < N$  tal que  $\forall i \neq k \ v[i] \leq v[i+1]$  (esto es, está ordenado imaginando que fuera un *array* circular).

Sea un *array* ordenado circularmente en el que todos los números se encuentran repetidos 2 veces salvo uno que aparece solo una vez. Se desea encontrar el elemento que aparece sólo una vez.

Ejemplo:

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

- a) (3.5 puntos) Diseñar un algoritmo basado en Divide y Vencerás con complejidad  $O(\log N)$  en el caso peor<sup>1</sup> (donde  $N$  es el tamaño del vector) que devuelva el elemento que aparece una sola vez.

```
int elementoEspecial(int[] vector)
```

1ª Opción:

```
int elementoEspecialAux(int[] vector, int i0, int iN){
    if (i0==iN) return vector[i0];
    else {
        int k=(i0+iN)/2;
        if (vector[k]==vector[k+1]) {
            if ((k - 1 - i0+1) % 2 == 1) return elementoEspecialAux(vector, i0, k-1);
            else return elementoEspecialAux(vector, k+2, iN);
        }
        else if (vector[k-1]==vector[k]) {
            if ((k - 2 - i0 + 1) % 2 == 1) return elementoEspecialAux(vector, i0, k-2);
            else return elementoEspecialAux(vector, k+1, iN);
        }
        else return vector[k];
    }
}

int elementoEspecial(int[] vector){
    if (vector.length==1) return vector[0];
    else {
        int i0, iN;
        if (vector[0] == vector[vector.length-1]) {
            i0 = 1; iN = vector.length-2;
        }else {
            i0 = 0; iN = vector.length - 1;
        }
        return elementoEspecialAux(vector, i0, iN);
    }
}
```

<sup>1</sup> Desarrollar un algoritmo que tenga una complejidad diferente a  $O(\log N)$  en el caso peor conllevará una puntuación de 0 en la pregunta.

2ª Opción:

```
int elementoEspecialAux(int[] vector, int i0, int iN){
    if (i0==iN)
        return vector[i0];
    else {
        int i0Aux, iNAux;
        if (vector[i0] == vector[iN]) {
            i0Aux = i0 + 1; iNAux = iN - 1;
        }else {
            i0Aux=i0; iNAux=iN;
        }
        int k=(i0Aux+iNAux)/2;
        if (vector[k]==vector[k+1]) {
            if ((k - 1 - i0Aux+1) % 2 == 1)
                return elementoEspecialAux(vector, i0Aux, k - 1);
            else
                return elementoEspecialAux(vector, k + 2, iNAux);
        }
        else if (vector[k-1]==vector[k]) {
            if ((k - 2 - i0Aux + 1) % 2 == 1)
                return elementoEspecialAux(vector, i0Aux, k - 2);
            else
                return elementoEspecialAux(vector, k + 1, iNAux);
        }
        else
            return vector[k];
    }
}

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

**b) (1.5 puntos)** Justifica que la complejidad del algoritmo desarrollado en el apartado anterior para el caso peor es  $O(\log N)$ , indicando qué condición tiene que cumplir el vector para estar en ese caso peor.

En el caso peor tenemos que:

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

Aplicando el Teorema Maestro para el caso 2 (ya que  $\log_2 1=0$ ), la complejidad es  $O(\log N)$ .