**Nº matrícula:**_____ **Nombre:** _____

**Apellidos:** _____

**Problema**. Sea un *array* ordenado. Por ejemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| **1** | **2** | **3** | **4** | **7** | **10** | **15** | **23** | **32** |

Rotar un *array* consiste en mover los elementos del array en $k$ unidades a la derecha (imaginando el *array* como si fuera circular). Por ejemplo, si rotamos el array anterior en 4 unidades, quedaría de la siguiente forma:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| **10** | **15** | **23** | **32** | **1** | **2** | **3** | **4** | **7** |

Deseamos encontrar el valor mínimo de un array sabiendo que estaba ordenado y que se ha rotado $k$ unidades (**desconociendo** el valor de $k$). En el ejemplo anterior, se devolvería 1.

**a)** Diseñar e implementar un algoritmo en Java basado en *Divide y Vencerás* con complejidad O(log $N$) en el caso peor[1] (donde $N$ es el tamaño del vector), que devuelva el valor mínimo de un array ordenado y rotado $k$ unidades (desconociendo el valor de $k$), teniendo en cuenta que no hay números repetidos.

```
int minArrayRotado(int[] vector)
```

```java
int minArrayRotado(int[] vector){
  return minArrayRotadoAux(vector ,0, vector.length-1);
}


int minArrayRotadoAux(int[] vector ,int i0, int iN){
    if (i0==iN)
        return vector[i0];
    else{
        int k=(i0+iN)/2;
        if ((vector[i0]<=vector[k]) && (vector[k]<vector[iN]))
            return vector[i0];
        else if (vector[i0]>vector[k])
            return minArrayRotadoAux(vector,i0,k);
        else // vector[iN]<vector[k] por estar ordenado y rotado
            return minArrayRotadoAux(vector,k+1,iN);
    }
}
```

[1] Desarrollar un algoritmo que tenga una complejidad diferente a O(log $N$) conllevará una puntuación de 0 en la pregunta.

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

El algoritmo implementado obedece a la siguiente ecuación de recurrencia en el tiempo para N>1:

$$T(N) = T(N/2) + O(1)$$

Esta ecuación es del tipo $T(N) = p \cdot T(N/q) + f(N)$, donde $f(N) \in O(N^a)$, con p=1, q=2 y a=0, por lo que podemos aplicar el Teorema Maestro. Dado que $\log_q(p) = \log_2(1)=0$ y a=0 nos encontramos en el caso 2º del Teorema maestro ($a=\log_q(p)$), por lo que la complejidad del algoritmo es: $T(N) \in O\left(N^{\log_q(p)} \cdot \log N\right) = O(N^0 \log N) = O(\log N)$