1) Problema 1. 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) (**3 puntos**) 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*).

```
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);
    }
}

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

b) (**2 puntos**) Calcular la complejidad del algoritmo desarrollado en el apartado anterior.

Las ecuaciones de recurrencia son:
$T(N)= \Theta(1)$ si $N=1$
$T(N)= T(N/2) + \Theta(1)$ si $N>1$
Podemos aplicar el caso 2 del teorema Maestro ya que $\log_2(1)=0$ y $\Theta(N^0)= \Theta(1)$.
Por lo tanto, tenemos que la complejidad del algoritmo es $\Theta(N^0 \cdot \log N)= \Theta(\log N)$

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