Ejercicio 1

Sea una imagen representada en una matriz cuadrada de tamaño $2^k \times 2^k$, siendo k>0. Cada elemento de la matriz, representa un píxel, y puede tomar valores 0 para blanco y 1 para negro, que representan el color de dicho cuadrado. Diseñe utilizando la técnica de ***divide y vencerás***, un algoritmo que compruebe que todos los valores de la matriz son 0 y 1.

Por ejemplo, dada la siguiente matriz (a) de entrada, el algoritmo devolvería "false", ya que algunos de los valores de los pixeles son incorrectos. Sin embargo para la matriz (b), devolvería "true", ya que todos los pixeles contienen valores 0 y 1.

a)

0	1	**-1**	1	1	0	1	1
0	0	0	0	0	0	1	1
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	**3**	0	0
0	**4**	1	1	1	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

b)

0	1	0	1	1	0	1	1
0	0	0	0	0	0	1	1
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

SE PIDE:

1. Rellenar la siguiente ficha
2. Diseñar utilizando la técnica de ***divide y vencerás*** los siguientes métodos:

   a. `private Boolean dYV(ProblemaDyV<Boolean> p)` (de la clase *AlgoritmoDivideYVencerasSinMemoria<Boolean>*). Dicho método permite obtener la solución a la técnica Divide y Vencerás sin memoria.
   b. `public Boolean esCasoBase()`(de la clase *ProblemaDyV<Boolean>*). Indica si el problema que se está resolviendo es tan simple que no necesita recursión para su resolución.
   c. `public Boolean getSolucionCasoBase()`(de la clase *ProblemaDyV<Boolean>*). . Devuelve la solución al problema, resuelto de forma directa.
   d. `public int getNumeroSubProblemas()`(de la clase *ProblemaDyV<Boolean>*). . Indica el número de subproblemas.
   e. `public ProblemaDyV getSubproblema(int i)` (de la clase *ProblemaDyV<Boolean>*). Devuelve el subproblema iésimo en el que se descompone el problema que se está resolviendo.
   f. `public Boolean combinaSoluciones (Boolean[] soluciones)` (de la clase *ProblemaDyV<Boolean>*). Combina cada solución de la lista de soluciones de manera que juntas resuelvan el problema completo.

Puntuación: 10 puntos **Tiempo estimado: 50 minutos**

<table>
<tr><th colspan="2">FICHA</th></tr>
<tr><th colspan="2">Ejercicio de funciones con comportamiento similar</th></tr>
<tr><td colspan="2">Técnica: Divide y Vencerás sin memoria</td></tr>
<tr><td colspan="2">Representación: Cada problema un conjunto de parámetros</td></tr>
<tr><td colspan="2">Tamaño: [1]m.getSize() ó [2]m.getSize()*m.getSize()</td></tr>
<tr><td>Propiedades Compartidas</td><td>m, Array[][]
umbral, Integer</td></tr>
<tr><td>Propiedades Individuales</td><td>i, Integer [0, m.getSize())
j, Integer [0, m.getSize())
n, Integer [0, m.getSize()]</td></tr>
<tr><td colspan="2">Solucion: Boolean</td></tr>
<tr><td colspan="2">checkPixelValue () = checkPixelValueDyV(0, 0, m.getSize())

checkPixelValueDyV (i, j, n) =
{ checkValues(i,j,n) n <= umbral
{ checkPixelValueDyV(i, j, k) &&
checkPixelValueDyV(i, j+k, k) &&
checkPixelValueDyV(i+k, j, k) &&
checkPixelValueDyV(i+k, j+k, k) &&

k= n / 2</td></tr>
<tr><td>Ecuación de Recurrencia</td><td>[1]T(n) = 4T(n/2) + k
[2]T(p) = 4T(p/4) + k
relación: a=4 b=2 d=0 / a=4 b=4 d=0 / $n^{\log(4\,2)} = p^{\log(4\,4)}$ -> $n^2 = p$</td></tr>
<tr><td>Complejidad</td><td>[1]O(n^2)
[2]O(p)</td></tr>
</table>

Con esquema:

```
private Boolean dYV(ProblemaDyV<Boolean> p){
        Boolean s;
        If (p.esCasoBase()){
                s = p.getSolucionCasoBase();
        }
        else {
                int numeroDeSubProblemas = p.getNumeroSubProblemas();
                Boolean soluciones  = Utiles.creaArray(tipoSolucion, numeroDeSubProblemas);
                for (int i = 0; i < numeroDeSubProblemas; i++){
                s = dyV(p.getSubProblema(i));
                soluciones[i] = s;
        }
                s = p.combinaSoluciones(soluciones);
        }
        return s;
}


public Boolean esCasoBase(){
        return (n <= umbral)
}

public Boolean getSolucionCasoBase(){
        Boolean enc = false;
        Int i2, j2;
        i2 = i;
        while  ((i2 < n) && (!enc)){
                while ( ((j2 < n) && (!enc)){
                        If ((matrix[i2][j2] != 0) || (matrix[i2][j2] != 1))
                                enc = true;
                j2++;
                }
        i2++;
        }
        return enc;
}

public int getNumeroSubProblemas()(
        return 4;
}

public ProblemaDyV getSubproblema(int i){
        int k = n / 2;
        switch(i){
        case 0:
                return new ProblemaDyV(i, j, k);
                break;
        case 1:
                return new ProblemaDyV(i, j+k, k);
                break;
```

```
        case 2:
                return new ProblemaDyV(i+k, j, k);
                break;

        case 3:
                return new ProblemaDyV(i+k, j+k, k) ;
                break;

        default:
                break;
        }
}

public Boolean combinaSoluciones (Boolean[] soluciones){
        return (soluciones[0] && soluciones[1] && soluciones[2] && soluciones[3]);
}
```

Sin esquemas:

```
Boolean  checkPixelValueDyV (Integer i,Integer j, Integer n){
        Boolean res;
        If (n <= umbral){
                res = checkValues(i,j,n);
        }else {
                k = n / 2
                Boolean r1 = checkPixelValueDyV(i, j, k);
                Boolean r2 = checkPixelValueDyV(i, j+k, k);
                Boolean r3 = checkPixelValueDyV(i+k, j, k) ;
                Boolean r4 = checkPixelValueDyV(i+k, j+k, k) ;

                res = r1 && r2 && r3 && r4      ;
        }
        return res;
}

Boolean checkValues (Integer i, Integer j, Integer n){

        Boolean enc = false;

        Int i2, j2;
        i2 = i;
        while  ((i2 < n) && (!enc)){
                while ( ((j2 < n) && (!enc)){
                        If ((matrix[i2][j2] != 0) || (matrix[i2][j2] != 1))
                                enc = true;
                j2++;
                }
        i2++;
        }

        return enc;
}
```

Otra opción más eficiente sería:

```
Boolean  compSimDyV (Integer i, Integer j, Integer n){
        Boolean res;
        If (n <= umbral){
                res = checkValues(i,j,n);
        }else {
                k = n / 2
                res =   checkPixelValueDyV(i, j, k) &&
                        checkPixelValueDyV(i, j+k, k) &&
                        checkPixelValueDyV(i+k, j, k)  &&
                        checkPixelValueDyV(i+k, j+k, k) ;
        }
        return res;
}
```