Ejercicio 2

Dada una lista no ordenado de números enteros, diseñe utilizando la técnica de ***divide y vencerás***, un algoritmo que determine cuantos números aparecen sólo una vez en la lista.

Por ejemplo, dada la siguiente lista de números enteros como entrada, el algoritmo devolvería "2" ya que el número "3" y "4" sólo aparecen una vez en la lista.

9	*1*	*9*	*1*	*8*	*8*	**3**	*5*	*5*	**4**

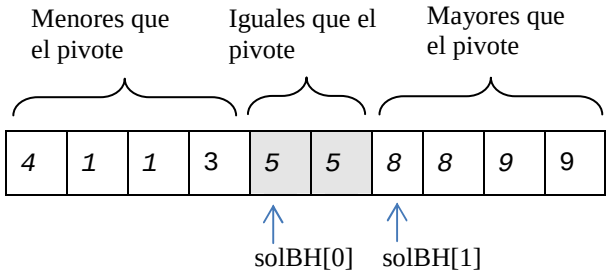
Puede utilizar los siguientes métodos:

`Integer elegirPivote(List<Integer> numeros, Integer i, Integer j)`: método que elige un valor como pivote para una lista de números enteros entre los índices "i" y "j".

`List<Integer> bh(List<Integer> numeros, Integer p, Integer i, Integer j)`: método que realiza el algoritmo de la bandera holandesa para la lista de números de enteros pasada como parámetro para un pivote "p" entre los índices "i" y "j". Dicho método devuelve una lista de enteros de tamaño dos que contiene los índices que indican las fronteras de los elementos menores, iguales y mayores que el pivote. Por ejemplo, para el caso anterior, si hicieramos:

```
List<Integer> solBH = new ArrayList<Integer>();
p = elegirPivote(numeros, 0 , numeros.size()) // si por ejemplo p fuera 5
solBH = bh(números,p,0,números.size());
```

Podríamos obtener usando los métodos anteriores una lista que divide el problema en 3 partes, una parte con los menores que el pivote, otra para los que son iguales que el pivote y otra última con los mayores.



SE PIDE:

1. Rellenar la siguiente ficha
2. Diseñar utilizando la técnica de ***divide y vencerás*** los siguientes métodos:
    a. `private Integer dYV(ProblemaDyV<Integer> p) (`de la clase *AlgoritmoDivideYVencerasSinMemoria<Integer>*). Dicho método permite obtener la solución a la técnica Divide y Vencerás sin memoria.
    b. `public Boolean esCasoBase()(`de la clase *ProblemaDyV<Integer>*). Indica si el problema que se está resolviendo es tan simple que no necesita recursión para su resolución.

c. `public Integer getSolucionCasoBase()(`de la clase *ProblemaDyV<Integer>*). . Devuelve la solución al problema, resuelto de forma directa.
d. `public int getNumeroSubProblemas()(`de la clase *ProblemaDyV<Integer>*). . Indica el número de subproblemas.
e. `public ProblemaDyV getSubproblema(int i) (`de la clase *ProblemaDyV<Integer>*). Devuelve el subproblema iésimo en el que se descompone el problema que se está resolviendo.
f. `public Integer combinaSoluciones (Integer[] soluciones) (`de la clase *ProblemaDyV<Integer>*). Combina cada solución de la lista de soluciones de manera que juntas resuelvan el problema completo.

FICHA	
Ejercicio de funciones con comportamiento similar	
Técnica: Divide y Vencerás sin memoria	
Representación: Cada problema un conjunto de parámetros	
Tamaño: J - I	
Propiedades Compartidas	numeros, List<Ingeter> umbral, Integer
Propiedades Individuales	i, Integer [0, m.getSize()) j, Integer [0, m.getSize())
Solucion: Integer	

`numerosUnaVez` () = `numerosUnaVezAux` (números, 0, numeros.size())

`numerosUnaVez` (numeros, i, j) =

`numerosUnaVezBase(números,i,j)`	`j-i <= umbral`

`p = elegirPivote(numeros,i,j);`
`(r1,r2) = bh(numeros,p,i,j);`
r = 0;
if (r2 – r1 <= 1){
r = 1;
}
`numerosUnaVez(numeros, i, r1) +`
`r +`
`numerosUnaVez(numeros, r2, j)`
	`j-i > umbral`

Puntuación: 10 puntos

Sin esquemas:

```
#include <stdio.h>

int numeros[] = {9,9,1,1,8,8,3,5,5,4};
const int TAM=10;

int main()
{

    int n = numerosUnaVez();

    printf("Existen %d numeros que aparecen una sola vez en el array", n);
    scanf("%d",&n);
       return 0;
}



int numerosUnaVez() {
       int n = numerosUnaVezAux(0,TAM);
       return n;
}

int numerosUnaVezAux(int i, int j){
       int r1,r2,p;
       int r = 0;

       if (j-i <= umbral){

              r = numerosUnaVezBase(números,i,j);

       }else {

              if (j-i == 0){
                     r = 0;
              }else {
                     p = elegirPivote(i,j);
                     banderaHolandesa(p,i,j,&r1,&r2);
                     if (r2 - r1 == 1){
                            r = r + 1;
                     }
                     r = r + numerosUnaVezAux(i,r1) + numerosUnaVezAux(r2,j);
              }
       }
       return r;
}

int elegirPivote(int i, int j){
       int k = (i + j) / 2;
       return numeros[k];
}

void intercambia(int i, int j){
       int aux = numeros[i];
       numeros[i] = numeros[j];
       numeros[j] = aux;
}

int banderaHolandesa(int p, int i, int j, int * r1, int * r2){
       int a, b, c;
```

```
	a = i; b = i; c = j;
	while(c-b>0){
		if(numeros[b]<p){
			intercambia(a,b);
			a++;
			b++;
		}else if(numeros[b]==p){
			b++;
		}else{
			intercambia(b,c-1);
			c--;
		}
	}
	*r1 = a;
	*r2 = b;
}



public Integer numerosUnaVezBase(int i int j){

	quicksort(numeros,i,j);
	if (j - i == 1){ //1 elemento
			r = 1;
		}else if ((j - i == 2) && (a[i] != a[j-1]))    { //2 elementos

			r = 2;
		} else { //al menos 3 elementos

			if (a[i] != a[i+1]){
				r++;
			}

			for (int k = i+1; k <= j-2; k++){
				if (a[k] != a[k-1] && a[k]!=a[k+1]){
					r++;
				}
			}

			if (a[j-2] != a[j-1]){
				r++;
			}
		}
	}
}
```

Con esquema:

```
public Boolean esCasoBase(){
		Boolean b = false;

		if (this.j - this.i <= umbral) {
			b = true;
		}
		return b;
	}

	public Integer getSolucionCasoBase(){
		Integer r = numerosUnaVezBase(this.i, this.j);
		return r;
	}
```

```
	public Integer getNumeroSubProblemas(){
		return 2;
	}

	public ProblemaDyV<Integer> getSubproblema(Integer k){

		Integer p = elegirPivote(this.i, this.j);
		List<Integer> solBH = bh(p, this.i,this.j);

		switch(k){

		case 0:
			return new ProblemaDyV(this.i, solBH.get(0));

		case 1:
			return new ProblemaDyV(solBH.get(1), this.j);

		default:
			return new ProblemaDyV(0,0);
		}

	}

	public Integer combinaSoluciones (Integer[] soluciones){

		Integer p = elegirPivote(this.i, this.j);
		List<Integer> solBH = bh(p,this.i,this.j);

		Integer r = 0;
		if (solBH.get(1) - solBH.get(0) == 1){
			r = r + 1;
		}

		return soluciones[0] + r + soluciones[1];
	}



private E dYV(ProblemaDyV<S,E> p){
   E s;
   if( p.esCasoBase()){
       s = p.getSolucionCasoBase();
   } else {
       int numeroDeSubProblemas = p.getNumeroDeSubProblemas();
       E[] soluciones =
            (E[])Array.newInstance(tipoSolucion,numeroDeSubProblemas);
       for(int i = 0; i < numeroDeSubProblemas; i++){
       s = dYV(p.getSubProblema(i));
       soluciones[i]=s;
       }
       s = p.combinaSoluciones(soluciones);
   }
   return s;
}
```