Ejercicio 2 (ITIS):

Utilice la técnica de Divide y Vencerás para resolver el problema de ordenación de listas mediante el método QuickSort. Dicha técnica se basa en la elección de un pivote de manera que las dos sub-listas resultantes de dividir el conjunto original en base a dicho pivote, se ordenarán siguiendo el criterio dado por el comparador, el cual se supone una propiedad compartida del problema. La lista de datos a ordenar será declarada como otra propiedad compartida al igual que el umbral.

Para plantear el algoritmo se suponen implementadas las siguientes funciones **estáticas** (**No tiene que implementar ninguna de ellas**), las cuales se encuentran en la clase **Utiles**:

- ***static E ep(List<E> datos ,int i, int j)****:* Elige aleatoriamente un elemento de la sub-lista *datos* definida por las posiciones [*i, j*). A dicho elemento se le denomina pivote. Este método se ejecuta en tiempo $\Theta(1)$.
- ***static int rs(List<E> datos, E e, int i, int j, Comparator<T> comp)****:* Reestructura la sub-lista *datos* comprendida entre las posiciones [i, j), alrededor del pivote *e*. Esto quiere decir que deja todos los elementos menores que el pivote a su izquierda y los mayores o iguales a su derecha. El método devuelve la posición del pivote. Este algoritmo está implementado iterativamente con complejidad $\Theta(n)$.
- ***static void sb(List<E> datos, int i, int j, Comparator<T> comp)***: Ordena por algún método iterativo conocido la sub-lista *datos* definida por los índices [*i, j)*. Estos algoritmos (como el de ordenación por inserción) tienen una complejidad de $\Theta(n^2)$.

La clase **ProblemaQuickSortImpl** define un constructor con la siguiente cabecera:

```
public ProblemaQuickSortImpl(List<E> datos, Comparator<E> comparator,
                             Integer i, Integer j, Integer umbral)
```

Se pide:

1. Complete la ficha que se adjunta en la que se ilustra el comportamiento del algoritmo de búsqueda mediante la técnica QuickSort
2. Implementar los siguientes métodos de la clase **ProblemaQuickSortImpl:**
   - **a.** private List<E> dYV(ProblemaQuickSort p);
   - **b.** boolean esCasoBase();
   - **c.** List<E> getSolucionCasoBase();
   - **d.** int getNumSubProblemas();
   - **e.** ProblemaDyV<List<E>> getSubProblema(int ind);
   - **f.** List<E> combinaSoluciones(List<E>... soluciones);

Nota:

No está permitido crear listas auxiliares. Se deberá trabajar con la propiedad **Datos.**
Como regla general, los intervalos serán cerrados al inicio y abiertos al final [i, j).

Puntuación: 3 puntos **Tiempo estimado: 60 minutos**

<table>
<tr><th colspan="2">Ficha</th></tr>
<tr><td colspan="2">Ordenación de una Lista: Quicksort</td></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>Tamaño: N=J-I</td><td></td></tr>
<tr><td>Propiedades Compartidas</td><td>_____
_____
_____</td></tr>
<tr><td>Propiedades Individuales</td><td>_____
_____</td></tr>
<tr><td colspan="2">Solución: _____</td></tr>
<tr><td colspan="2">$qs(d, o) =$ s(0,Datos.Size)
$s(i,j) = \begin{cases} \_\_\_\_\_, & n \leq umbral \\ p = ep(i,j); k = rs(i,j,p); s(i,k); s(k+1,j), & n > umbral \end{cases}$</td></tr>
<tr><td>Recurrencia</td><td>$T(n) = n - 1 + \frac{1}{n}\sum_{k=0}^{n-1}(T(k-1) + T(n-k))$</td></tr>
<tr><td>Complejidad</td><td>$\Theta(n \log n)$</td></tr>
</table>

1. Solucion:

Ficha	
Ordenación de una Lista: Quicksort	
Técnica: Divide y Vencerás con Memoria	
Representación: Cada problema un conjunto de parámetros	
Tamaño: N =J-I	
Propiedades Compartidas	*Datos, List<E>* *Orden, Comparator<E>* *Umbral, Integer*
Propiedades Individuales	*I, entero en [0,Datos.size())* *J, entero en [I,Datos.size())*
Solución: List<E>	
$qs(d,o) = s(0,Datos.Size)$ $s(i,j) = \begin{cases} sb(i,j), & n \leq umbral \\ p = ep(i,j); k = rs(i,j,p); s(i,k); s(k+1,j), & n > umbral \end{cases}$	
Recurrencia	$T(n) = n - 1 + \frac{1}{n}\sum_{k=0}^{n-1}(T(k-1) + T(n-k))$
Complejidad	$\Theta(n \log n)$

2. Funciones propias del algoritmo DyV

a.

```
private List<E> dYV(ProblemaDyV<List<E>> p) {
        List<E> s;
        if (p.esCasoBase()) {
                s = p.getSolucionCasoBase();
        } else {
                int numeroDeSubProblemas = p.getNumeroSubProblemas();


                S[] soluciones = Utiles.creaArray(numeroDeSubProblemas);


                for (int i = 0; i < numeroDeSubProblemas; i++){
                        ProblemaDyV<List<E>> pr = p.getSubProblema(i);

                        s = dYV(pr);
                        soluciones[i] = s;
                }
                s = p.combinaSoluciones(soluciones);
        }
        return s;
}
```

b.

```
public boolean esCasoBase() {
	return (getJ()-getI()) <= getUmbral();
}
```

c.

```
public List<E> getSolucionCasoBase() {
	Utiles.sb(datos, i, j, comparator);
	return datos;
}
```

d.

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

e.

```
public ProblemaDyV<List<E>> getSubProblema(int ind) {
	ProblemaQuickSort<E> pqs;
	E pivote;
	Integer k;

	pivote= Utiles.pe(datos, i, j);

	k= Utiles.rs(datos, pivote, i, j, comparator);

	switch (ind) {
		case 0:
			pqs=new ProblemaQuickSortImpl<E>(datos,comparator,i,k);
			break;
		case 1:
			pqs=new ProblemaQuickSortImpl<E>(datos,comparator,k,j);
			break;
		default:
			throw new SituacionIlegal("Solo hay dos casos!");
	}
	return pqs;
}



public List<E> combinaSoluciones(List<E>... soluciones) {
	return datos;
}
```