1. Problema propuesto a resolver.

**Optimizar el ordenamiento de arreglos de números con QuickSort implementando la concurrencia con Fork/Join y Executor.**

2. Características importantes a considerar, como por ejemplo en qué condiciones mejora el tiempo del concurrente al secuencial o viceversa.

**Algo importante a considerar es que con una cantidad de elementos reducida de 10k o menos, no se nota la mejora con el método concurrente. Por lo tanto, si realmente se desea aprovechar al cien por ciento se deben hacer pruebas con un numero de elementos mayor a 100k.**

**también es importante decir que, de los dos métodos concurrentes, la versión antigua, es decir, el Executor, tuvo un mejor rendimiento en todos los casos presentados.**

3. Algoritmo propuesto e implementación del procesamiento secuencial.

**El algoritmo propuesto es el QuickSort, la implementación es bastante simple, al recibir el arreglo, tenemos que identificar el inicio y el fin de este, así como un pivote, que es un elemento en una posición de nuestra preferencia que servirá como punto de referencia. El resto del algoritmo se basa en dividir el arreglo en dos, al lado izquierdo del pivote van los elementos menores a el y del lado derecho van los elementos mayores a él. Y a partir de aquí se comienza a usar recursividad, llamándose a sí mismo la función dos veces para seguir dividiendo y ordenando el arreglo tanto de lado derecho como del izquierdo.**

4. Algoritmo propuesto e implementación del procesamiento concurrente.

**Lo primero que se tiene que hacer crear un objeto de la clase ForkJoinPool que contendrá el arreglo que queremos ordenar. Con el método invoke le enviamos el objeto de la clase donde se encuentra nuestro algoritmo concurrente para poder iniciarlo.**

**Una vez dentro hacemos lo mismo que en el método secuencial, primero tenemos que setear las variables de inicio y fin del arreglo. La clase RecursiveAction nos da un método llamado compute, que es donde debemos poner todo el código o las funciones que queremos ejecutar concurrentemente. En este caso en el algoritmo original QuickSort no se usa la recursividad hasta después de haber reordenado el algoritmo. Por lo tanto, primero ejecutamos el bloque de código que hace dicho trabajo y guardando su resultado en una variable. Una vez hecho esto es cuando las funciones recursivas del algoritmo original las ejecutamos de manera concurrente con el método invokeAll así haciendo que el acomodo del arreglo sea mucho más rápido.**