**Quick Sort**

Para el algoritmo de Quick Sort se tiene un máximo orden de complejidad para el movimiento de elementos innecesario repetidamente para cada pivote, en este caso la complejidad esta dada por el orden de O(N^2). En el mejor de los casos se tiene una organización óptima de O(n\*log(n)), en la cual cada pivote implica el menor número de movimientos posible.

Este algoritmo es de tipo *in-Place* ya que no recurre a estructuras adicionales para su organización, es estable debido a que implementa elementos de tipo *Compare*, incluyendo la condición de organización en *compareTo*( ) . Este es adaptativo en el sentido que si los objetos están en su posición respectiva según el pivote no son movidos, permitiendo variar el orden de complejidad del algoritmo.

**Merge Sort**

En este caso el algoritmo no necesariamente tiene un peor o mejor caso, ya que el número de accesos y escrituras son siempre los mismos, la división del arreglo y su posterior unión requiere una complejidad de O(n\*log(n)).

Este algoritmo no es de tipo *in-Place*, este crea arreglos de tamaños menores para posteriormente unirlos en un arreglo ordenado. Es adaptativo en el sentido que no separa a todos los componentes sin importar su orden inicial. Por último, el algoritmo es estable por la implementación de *compareTo( ).*

**Shell Sort**

Dependiendo de la organización del arreglo y la separación de la selección el algoritmo se puede tener una complejidad óptima O(n\*log(n)), o en el peor de los casos de O(n).

Shell sort es *in-Place* ya que no utiliza nuevas estructuras para la creación de objetos, es adaptativo ya que puede recorrer el arreglo en intervalos o gaps variables hasta ordenar el arreglo. Por las razones mencionadas en los demás métodos de ordenamiento este método también es estable.