My QuickerThanMergeSort class was written in hopes to come close to the running times of the merge sort algorithm we covered in class.

My algorithm implemented a sortArray function, a findLastSubArrayIndexFunction, a sort function, and a merge function.

sortArray: This method returns an ArrayList of int arrays. This is important because ArrayLists have a dynamic size. I was struggling with the implementation of a method like this at first, but I realized I could utilize ArrayLists. The int arrays contain the first and last index of the sorted portions of the unsorted array. Ex: [0, 2] , [3, 3]

The sortArray method references the findLastSubArrayIndex to find the last sorted index of the sub array.

findLastSubArrayIndex: This method loops through the whole unsorted array and returns the last index of the sorted subarray. It does the by using a while loop that ensures the current position is in bounds and sorted(i.e. arr[index] <= arr[index+1]

sort: The sort method starts by creating an ArrayList of type int array. This int array stores all the sorted subarray and their start and end indexes. This is important because we can slowly shrink the array into just one sorted integer array. As long as the ArrayList of index arrays is larger than size 1, it will continue merging.

I assign 4 variables, two for the start and end index of the first subarray, and two more for the start and end index of the second subarray. I'm doing this because I want to merge them.

The method references the merge function to merge the two sorted subarrays, and uses the 4 variables and the copyOfRange method to copy the specified range of the specified subarrays into a new array (getLeft, getRight essentially).

Once it's merged the two subarrays the method adds a sortedRegion array to index 0 of the ArrayList. The sortedRegion array is the merged version of the first two subarrays, and it will continue to grow, since we will be comparing/merging everything else to it. Now we call the ArrayList remove method to get rid of the two subarrays we just merged, to slowly get down to just one sorted array. The ArrayList now contains a merged version of the first two subarrays at position zero, and adjacent to it would be the third subarray of the original array.

merge: This method is identical to the method we looked at in class for the original merge sort. It initially compares the two subarrays and fills the sorted array. Once one side has been sorted, it will fill the remainder of either the first or second array into the sorted portion.