Merge Sort

• 1: Write the code for a `mergeSort` function, assuming a merge function (see below) already exists and can be used in the implementation.
– `mergeSort` sorts an array of integers from a given start index for a given length.
– **mergeSort(int arrayToSort[], int startIndex, int lengthToSort)**
– It will be required that the given startIndex and lengthToSort will not overrun the array bounds.

• 2: write the code for the `merge` function
– this merges the contents of an array which has been sorted in 2 halves (from startIndex to length/2, and from startIndex + length/2 until the full sorted length)
– **merge(int arraySortedInTwoHalves[], int startIndex, int length)**
– *Tip: create(**using new**) a temporary array to hold the merged result, merge to this array, then copy back to the array to be merged, and delete the temp array)*

Quick Sort

• 1: Implement a quickSortDivide() function for an array of integers.
– Your function may have one of the following signatures

```
int quickSortDivide(int *theArray, int size);
```

or

```
int quickSortDivide(int theArray[], int first, int last);
```

Note: in either case, you return the final position of the pivot in the array. In the first case, the array variable provides the address of the first element of the array to be divided, so first index position will always be 0. This will make the implementation of the method easier, but calling it might be harder!
In the second case, we pass in the index positions of the section of the array to be sorted, so some thought might be needed in the implementation, but it is still straightforward.

• 2: Write the recursive quicksort.

```
quickSort(int *theArray, int size);
```

or

```
quickSort(int theArray[], int first, int last);
```

This function will use the `quickSortDivide` you have already written, and should be very short.