Quick Sort

1a: Revisit Lab 2b part 5 and then implement the quickSortDivide() function below for an array of integers. Your function **must** have the following signature.

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

Note: You return the final position of the pivot in the array. The array variable provides the address of the first element of the array to be divided, so first index position will always be 0.

1b: Write the recursive quicksort. You **must** use the following signature.

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

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

Merge Sort

2a: Write the code for the `mergeSort` function, assuming a merge function 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.

2b: 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)*