**Part 4: Sorting**

For this part, you will implement two different sorting methods and compare the number of swaps each one takes. It is strongly recommended that you start by implementing the functions and test them on arrays, then, once you are certain the sorting is done correctly, implement a count to keep track of the number of swaps. That is the value that will be returned. See the test cases for more details.

1) Implement insertion sort in insertionSort.java

function specifications:

public int insertionSort(int[] array)

array: an array of integers to be sorted(see test cases for sample arrays)

return value: number of swaps of elements

For example, from {1, 2, 3, 5, 4} to {1, 2, 3, 4, 5} is considered as 1 swap because the indices for 4 and 5 are exchanged.

note: do not swap if two neighboring elements are equal(ie. for {1, 2, 3, 4, 5, 5}, do not swap 5 and 5 as it will throw off the count and result in test case failure.)

2) Implement merge sort in Merge.java

function specifications:

public int sort(int[] array)

array: an array of integers to be sorted(see test cases for sample arrays)

return value: number of array accesses(read/write)

aux[k] = array[k] is considered as 2 array accessed, reading array[k] is one access and writing to aux[k] is another access.

You may write your own main method for your own testing but it is NOT required to pass the test cases. To run all test cases, click the green button on the left of public class insertionSortTest and select Run 'insertionSortTest'. (MergeTest is run the same way). You may also run each individual test by clicking the green button on the left of each test and select run.

Each test case checks:

1) whether your array is correctly sorted;

2) whether the correct number of swaps/read write accesses is returned.

Make sure that your array is sorted correctly before implementing counts.

Check test case output message at the console to determine which part of the test case you failed.