Homework #5

Due: **11/29/18 at 10:00 PM**
Late Submissions: **No Late Submissions**
Program Files (1): **MergeSortComparison**

In class, we learned various O(n log n) sorting algorithms. In this assignment, you will implement a more general version of the merge sort algorithm from the book, and then implement an alternate version that is designed to be more memory efficient.

You should write one class for this assignment: **MergeSortComparison.** All methods in this class should be static. In this assignment, I describe the signature and behavior of three methods. You should create additional methods to support these methods and avoid writing redundant code.

mergeSort(list:E[]):void – This generic method adapts the merge sort algorithm from the book to work with arrays of any object that implements the Comparable interface. Your first task is to correctly define the formal generic type parameter. Then you will have to modify the body of this method and the **merge** method so that both use the formal generic type where appropriate. Also, you will need to modify the code to rely on the Comparable interface for determining order.

newMergeSort(list:E[]):void – Like the method above, this is a generic method that accepts any array whose type is a class that implements the Comparable interface. However, unlike the previous method, it will not create copies of the first and second half of the array. Instead, it will use a recursive helper method that takes as parameters the original array and the start and end indices of the portion of the array to sort. Each recursive case will call this method again with indices that specify the half of the list to sort. This method will also require its own merge method. This new merge method should have the following parameters: the original list, the start index of the first list to merge, the start index of the second list to merge, and the end index of the second list. It is assumed that the first list and second list are consecutive, so the last element of the first list is the element prior to the first element of the second list. To make this work, you will need a temporary place to store the merged results of the two lists. To do this, create a static field in the class that is an array of type Comparable, and make sure that when you call **newMergeSort** on a new array, that this field is initialized to be large enough to handle the largest merge that will be done to solve the problem. To save time and memory, do not resize the array for the subproblems; instead just use as much of it as you need on each merge call, and ignore the rest. Whenever you have successfully merged two lists, you will need to copy the results from the temporary static array back into the appropriate place in the original list.

main(args:String[]):void – Your main method should run an experiment that tests both versions of merge sort on different sizes of random Integer arrays and again on different sizes of random Double arrays. You should test on arrays of sizes 100,000, 500,000, 1,000,000 and 5,000,000. You should use the Random class to populate one set of these arrays with random integers from the full set of possible int values. Run each algorithm on a copy of an array of the desired size, and output the time it takes in milliseconds. On the same line, print the percentage speedup of the new algorithm vs. the original one. For example, if the original algorithm took 200ms and the new one took 150ms, the speed up is 25% (because it take 75% of the time to finish). If the new algorithm is slower than the original, then this should be reported as a negative improvement. After printing the table for integers, repeat the process for arrays of Doubles of the same sizes.

Again, you can use the Random class to populate these arrays with random doubles in the range from 0.0 to 1.0.

See below for an example of the output format. Here, the actual times have been omitted, but each *d* stands for a possible digit. Times should be formatted to be right-aligned and with up to five digits. Percentages should have up to three characters before the decimal point, and one digit after.

```
> run MergeSortComparison
Integers:
Test Size     Orig       New    Improve(%)
   100000    ddddd     ddddd       ddd.d %
   500000    ddddd     ddddd       ddd.d %
  1000000    ddddd     ddddd       ddd.d %
  5000000    ddddd     ddddd       ddd.d %
Doubles:
Test Size     Orig       New    Improve(%)
   100000    ddddd     ddddd       ddd.d %
   500000    ddddd     ddddd       ddd.d %
  1000000    ddddd     ddddd       ddd.d %
  5000000    ddddd     ddddd       ddd.d %
```

At a minimum provide a Javadoc comment explaining what each method does, and another one that summarizes the class. Remember, Javadoc comments start with the characters "/**" and end with "*/". Include additional comments for lines that are especially complicated. At the top of the program include a comment of the following form:

```
/*
CSE 17
Your name
Your Lehigh e-mail
[if you used a tutor, provide his/her contact information here]
Homework #5      DEADLINE: November 29, 2018
Program: Merge Sort Comparison
*/
```

Once the program compiles, runs, and has been tested to your satisfaction, upload the .java file to Course Site. To do so, click on the name of the assignment in the Course Site page, and then press the "Add submission" button at the bottom of the next page. Drag and drop the file into the area under "File submissions". If necessary, you can update your submission at any time before the deadline passes. Be very careful to ensure that you named the class, file, and methods correctly, including using the correct case for all letters in the names. In addition to running your main method, we will use a test program that will directly call **mergeSort** and **newMergeSort** on small arrays of different types of our choosing, and will be inspecting the results to make sure the algorithms sort correctly.