Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 8**. This includes unused code, so don't submit extra files that don't compile. (Java is backwards compatabile so if it compiles under JDK 7 it *should* compile under JDK 8.)
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, or method signatures.
4. Do not add additional public methods when implementing an interface.
5. Do not use anything that would trivialize the assignment. (e.g. don't import/use `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
6. You must submit your source code, the `.java` files, not the compiled `.class` files.
7. After you submit your files redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

Sorting

For this assignment you will be coding 6 different sorts: selection sort, insertion sort, bubble sort, merge sort, quick sort and radix sort. We will be looking at the number of comparisons between elements while grading. **Failure to follow requirements will result in significant point deductions.** For example, if a sort is to be in-place, the result MUST be in-place.

Bubble Sort

Bubble sort should be inplace and stable. This means that duplicates should remain in the same relative positions after sorting as they were before sorting. It should have a worst case running time of $O(n^2)$ and a best case of $O(n)$.

Insertion Sort

Insertion sort should be inplace and stable. This means that duplicates should remain in the same relative positions after sorting as they were before sorting. It should have a worst case running time of $O(n^2)$ and a best case running time of $O(n)$.

Selection Sort

Selection sort should be inplace. It should have a worst case running time of $O(n^2)$ and a best case running time of $O(n^2)$.

Quick Sort

Quick sort should be inplace. It should have a worst case running time of $O(n^2)$ and a best case running time of $O(n \log n)$.

Merge Sort

Merge sort should be stable. This means that duplicates should remain in the same relative positions after sorting as they were before sorting. It should have a worst case running time of $O(n \log n)$ and a best case running time of $O(n \log n)$.

Radix Sort

Radix sort should be stable. This means that duplicates should remain in the same relative positions after sorting as they were before sorting. It should have a worst case running time of $O(kn)$ and a best case running time of $O(kn)$ where k is the number of digits in the longest number. You will be sorting ints. Note that you CANNOT change the ints into strings for this excerise.

Other Sorts

These are examples of sorts that you should not do. (Source: `http://xkcd.com/1185/`)

Style and Formatting

It is important that your code is not only functional but is also written clearly and with good style. We will be checking your code against a style checker that we are providing. It is located in resources along with instructions on how to use it. We will take off a point for every style error that occurs. If you feel like what you wrote is in accordance with good style but still sets off the style checker please email Jonathan Jemson (jonathanjemson@gatech.edu) with the subject header of "CheckStyle XML". **Please make sure that you are using the CheckStyle XML file named CS1332-checkstyle-v2.xml. All homework assignments will use this CheckStyle.**

Javadocs

Javadoc any helper methods you create in a style similar to the Javadocs for the methods in the interface.

Provided

The following files have been provided to you:

1. `SortingInterface.java` This is the interface you will implement. All instructions for what the methods should do and the requirements for each method are in the javadocs. **Do not alter this file.**
2. `Sorting.java` This is the class in which you will actually implement the interfaces. Feel free to add private helpers but **do not add any new public methods.**
3. `SortingTestsStudent.java` This is the test class that contains a set of tests covering the basic cases for sorting. It is not intended to be exhaustive nor guarantee any type of grade. **Write your own tests to ensure you cover all edge cases.**

Deliverables

You must submit all of the following files. Please make sure the filename matches the filenames below.

1. `Sorting.java`

Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interface, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc. You may attach each file individually, or submit them in a zip archive.