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 7**. This includes unused code, so don't submit extra files that don't compile. (Java is backwards compatabile so if it compiles under JDK 6 it *should* compile under JDK 7.)
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. Always be very conscious of efficiency. Even if your method is to be O(n), traversing the structure multiple times is considered non-efficient unless that is absolutely required (and that case is extremely rare). Traversing beyond your data and into unoccupied areas of the backing store is not-efficient and is not actually O(n). See more discussion of Big O and efficiency below.
7. You must submit your source code, the `.java` files, not the compiled `.class` files.
8. 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.

Bubble Sort

Bubble sort should be inplace and stable. 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. 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(nlogn).

Merge Sort

Merge sort should be stable. It should have a worst case running time of O(nlogn) and a best case running time of O(nlogn).

Radix Sort

Radix sort should be stable. 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.

Driver

You will also be required to create a Driver class that will take in a number from the user and then create a random array of Integers or ints of that size. If the user enters 0, your program should exit. You should print the unsorted array and once you have sorted the array, you should print it as well. It will sort the array with each sorting algorithm and tell the user how long it took to sort the array in seconds. You must sort the array with every algorithm and report the times. Note that if you created an array of Integers you will need to create an array of ints for radix sort and if you created a random array of ints you will nee to create an array of Integers for all the other sorts. These two array should hold the same data, just in different forms. Here is an example of what your drive could print:

Please enter the size of the array you would like to sort or enter 0 to exit:
7
The unsorted array is: 8972, -1689, 5957, 270, -6277, 5484, -19
The sorted array is: -6277, -1689, -19, 270, 5484, 5957, 8972

It took bubble sort 2.62751E-4 seconds.
It took insertion sort 2.07164E-4 seconds.
It took selection sort 2.5561E-4 seconds.
It took quick sort 2.06575E-4 seconds.
It took merge sort 2.13568E-4 seconds.
It took radix sort 1.94572E-4 seconds.

Please enter the size of the array you would like to sort or enter 0 to exit:

Style and Formatting

It is important that your code is functional and is also written clearly and with good style. Efficiency also always matters. Be sure you are using the updated style checker located in T-Square-Resources. (It was updated on February 5, 2014.) Be sure to run your code against this latest style checker. While it is backward compatible with the original style checker, some style rules have been made easier to pass, so it is in your interest to update to this Feb 5 style checker file.) If you feel like what you wrote is in accordance with good style but still sets off the style checker please email Kush Mansingh kmansingh3@gatech.edu with the subject header of "Style XML".

Javadocs

All public methods must have javadocs (or @Override's where appropriate). All private methods must be commented, but it is not necessarily that they be in javadoc style.

Provided

The following files have been provided to you:

1. `SortingInterface.java`
2. `CheckStyle.zip` - see T-Square Resources for the latest file from February 5, 2014

Deliverables

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

1. `Sorting.java`
2. `Driver.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.