Important

There are general submission 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.
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. Do not add any new instance variables.
8. All methods must be efficient, even if a runtime is not specified.
9. 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.

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. For example:

```
throw new PDFReadException("Did not read PDF, will lose points.");
```

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".

Javadocs

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

Forbidden Statements

You may not use these in your code at any time in CS 1332.

- `break` may only be used in switch-case statements
- `continue`
- `package`
- `System.arrayCopy()`

- `clone()`
- `assert()`
- `Arrays` class
- `Array` class
- `Collections` class
- Reflection APIs

Debug print statements are fine, but should not print anything when we run them. We expect clean runs - printing to the console when we're grading will result in a penalty. If you use these, we will take off points.

Sorting

For this assignment you will be coding 6 different sorts: bubble sort, insertion sort, shell sort, quick sort, merge sort, and radix sort. In addition to the requirements for each sort, we will be looking at the number of comparisons made between elements while grading.

Comparator

Each sorting method (except radix sort) will take in a comparator and use it to sort the elements of the array using various sorting algorithms described below and in the sorting file.

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 $O(n)$.

Shell Sort

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

For shell sort, there are various formulas for determining the gap distances. For this homework, use the sequence $\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{8} \rfloor, \ldots, 1$. For example, if the length of the array is 38, then the first gap distance is 19, the second gap distance is 9, and so on until you reach a gap distance of 1.

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 exercise.

Provided

The following file(s) have been provided to you. There are two files, but you will edit only one of them.

1. `Sorting.java` This class is where you will be implementing your sorting algorithms. **Do not add any public methods to this file.**
2. `SortingStudentTest.java` These are some sample JUnits, similar to those that will be used to grade your assignment. **Passing this does not guarantee any sort of grade.**

Deliverables

You must submit all of the following file(s). Please make sure the filename matches the filename(s) below. Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interfaces, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `Sorting.java`

You may attach each file individually or submit them in a zip archive.