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. Any compile errors will result in a 0.
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.
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 $\mathcal{O}(n)$, traversing the structure multiple times is considered non-efficient unless that is absolutely required (and that case is extremely rare).
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 Algorithms

In this assignment you will implement 6 different sorting algorithms: bubble sort, insertion sort, selection sort, quick sort, merge sort, and LSD radix sorts. In addition to the requirements for each sort, we will be looking at the number of comparisons made between elements. For this homework, you will be sorting the arrays of data in **ascending order**. You may also assume that there will be no null data within the arrays of data.

Comparator

Each generic sorting method will take in a comparator. You are required to use this comparator to sort elements of the passed array using algorithms described below.

In-place Sorts

Some of the sorts below are in-place sorts. This means that the items in the array passed in are not copied over to another array or list. Note that you can still create variables that hold only one item; you cannot create another array or list inside the method.

Stable Sorts

Some of the sorts below are stable sorts. This means that duplicates should remain in the same relative positions after sorting as they were before sorting.

Bubble Sort

Bubble sort should be in-place and stable. It should have a worst case running time of $\mathcal{O}(n^2)$ and a best case running time of $\mathcal{O}(n)$.

Insertion Sort

Insertion sort should be in-place and stable. It should have a worst case running time of $\mathcal{O}(n^2)$ and a best case running time of $\mathcal{O}(n)$.

Note that, for this implementation, you should sort from the beginning of the array. This means that after the first pass, segment 0-1 should be considered as sorted. After the second pass, segment 0-2 should be considered as sorted. After the third pass, segment 0-3 should be considered as sorted, and so on.

Selection Sort

Selection sort should be in-place. It should have a worst case running time of $\mathcal{O}(n^2)$ and a best case running time of $\mathcal{O}(n^2)$.

Quick Sort

Quick sort should be in-place. It should have a worst case running time of $\mathcal{O}(n^2)$ and a best case running time of $\mathcal{O}(n \log n)$.

Your implementation of quick sort must match the method shown in lecture; otherwise, you may not receive full credit for the sort.

Merge Sort

Merge sort should be stable. It should have a worst case running time of $\mathcal{O}(n \log n)$ and a best case running time of $\mathcal{O}(n \log n)$.

Radix Sort

LSD radix should be stable. You will do LSD (Least Significant digit) radix sort which starts comparing with the least significant digit (LSD).

LSD radix sort has a worst case running time of $\mathcal{O}(kn)$ and a best case running time of $\mathcal{O}(kn)$ 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 at any point in the sort for this exercise. In addition, the sorts **must** be done in base 10.

If you need to calculate the result of a number raised to a power, you should use the `pow()` method we have provided. You may only use `Math.abs()` from the `Math` class.

A note on JUnits

We have provided a **very basic** set of tests for your code, in `SortingStudentTests.java`. These tests do not guarantee the correctness of your code (by any measure), nor does it guarantee you any grade. You may additionally post your own set of tests for others to use on the Georgia Tech GitHub as a gist. Do **NOT** post your tests on the public GitHub. There will be a link to the Georgia Tech GitHub as well as a list of JUnits other students have posted on the class Piazza (when it comes up).

If you need help on running JUnits, there is a guide, available on T-Square under Resources, to help you run JUnits on the command line or in IntelliJ.

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 T-Square, under 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 Joonho Kim (jkim844@gatech.edu) with the subject header of "CheckStyle XML".

Javadocs

Javadoc any helper methods you create in a style similar to the existing Javadocs (remember to keep helper methods private). If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing Javadocs.

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. **The message must be useful and tell the user what went wrong**. "Error", "BAD THING HAPPENED", and "fail" are not good messages. The name of the exception itself is not a good message.

For example:

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

```
throw new IllegalArgumentException("Cannot insert null data into data structure.");
```

Generics

If available, use the generic type of the class; do **not** use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`. Using the raw type of the class will result in a penalty.

Forbidden Statements

You may not use these in your code at any time in CS 1332. If you use these, we will take off points.

- `break` may only be used in switch-case statements
- `continue`
- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class
- `Array` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs

- Inner or nested classes

Debug print statements are fine, but nothing should be printed when we run them. We expect clean runs - printing to the console when we're grading will result in a penalty.

Provided

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

1. `Sorting.java`

   This is the class in which you will implement various sorting algorithms. Feel free to add private static helper methods but **do not add any new public methods, new classes, instance variables, or static variables**.

2. `SortingStudentTests.java`

   This is the test class that contains a set of tests covering the basic operations on the `Sorting` class. It is not intended to be exhaustive and does not guarantee any type of grade. **Write your own tests to ensure you cover all edge cases.**

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`