EECS 2030: Lab 4

(about 2 % of the final grade, may be done in groups of up to three students)

Due: as set in *eClass*

Motivation

The purpose of this lab is to complete two implementations of *insertion sort* and compare their performance to that of a modern algorithm.

Part 1: Getting Started

Download a zip file containing the Lab 4 Eclipse project.

Import the project into Eclipse by doing the following:

1. Under the **File** menu choose **Import...**
2. Under **General** choose **Existing Projects into Workspace** and press **Next**
3. Click the **Select archive file** radio button, and click the **Browse...** button.
4. In the file browser that appears, navigate to your download directory (exactly where this is depends on what computer you working on; on the lab computers the file will probably appear in your home directory)
5. Select the file **lab4.zip** and click **OK**
6. Click **Finish**.

Explore the existing methods and the test cases, try to understand the purpose of each line. For example, how the generic types were used and how the `Comparator` types were specified.

Part 2: Sorting Implementation

Open the class `Lists` and implement the two methods with TODO sections.

Insertion sort is somewhat similar to selection sort shown in class. However, it's a different algorithm. In particular, for a list that is almost sorted, insertion sort takes linear time (i.e., one pass over the list being sorted, rather than quadratic time for selection sort. For a short description, see https://en.wikipedia.org/wiki/Insertion_sort#Algorithm

For the iterative solution, your implementation will basically contain two nested loops: for each element *i* of the input, you search for an appropriate location for it in the already sorted portion of the list.

For the recursive solution, you may base the implementation on the ideas described in the notes on recursion. Similar to the selection sort example, where you would need to have a helper method to find a maximum value, in your assignment you will probably need a helper (a recursive one) to do the insertion[1]. Think about what should be the base case, think about where to insert elements (e.g., at the beginning or at the end of the list), and so on. You can assume that an `ArrayList` implementation is going to be used with your sorting method.

`JUnit` testers for your classes are available in the project that you downloaded. Note that the testers are not very thorough, and it may not catch all errors that you might make. Also note that

[1] The recursive implementation described in https://www.geeksforgeeks.org/recursive-insertion-sort/ is not suitable, as it is not completely recursive and contains a loop.

passing all of the tests in this tester *does not* guarantee a good solution (in other words, you should think critically about your implementation for each method).

Part 3: Experiments

Complete the implementation of the `sortExperiment` class. Use the existing pattern to add the other sorting algorithms into the experimental procedure, and format the output in a similar fashion. E.g.,

```
N=1000
Default Sort, Time spent: 1.76309 ms
Insertion-Iterative, Time spent: XXX ms
Insertion-Recursive, Time spent: YYY ms

N=2000
Default Sort, Time spent: 3.848191 ms
Insertion-Iterative, Time spent: XXX ms
Insertion-Recursive, Time spent: YYY ms
```

<repeat for larger values of N, doubling every time>

NOTE: you will probably discover that large values cause the recursive solution to fail due to stack overflow, and make the insertion sort algorithms take a very long time. Reduce the maximum value of *N* in the `sortExperiment` accordingly.

Your program should print the results above to the console using appropriate method calls in your `main` method, or in additional static helper methods. Save the result of your program execution in a file `testrun.txt` and submit it together with your other files. The `testrun.txt` file can be produced by either manually copying the output of your program from the console window into a text editor and saving it, or by using *output redirection*.

If you have questions, don't hesitate to post your questions on the course forum on eClass.

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[2]. We look at whether the code passes the unit tests, satisfies the requirements of this document, and whether it conforms to the code style rules.

Submission

Find all the `java` files in your project and submit them electronically via eClass (no zipping is required). There should be four files in total: three classes and the `testrun.txt`.

If working in a group, make only one submission and include a `group.txt` file containing the names and the student numbers of the group members. The deadline is firm.

Academic Honesty

Direct collaboration (e.g., sharing your work results across groups) is not allowed (plagiarism detection software may be employed). However, you're allowed to discuss the assignment requirements, approaches you take, etc. Also, make sure to state any sources you use (online sources – including web sites, old solutions, books, etc.). Although using outside sources is

[2] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/

allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced.