CS 171: Introduction to Computer Science II
Assignment #6: Bubble Sort and Merge Sort

(Due: April 6th at 11:59 PM via turnin)

IMPORTANT NOTE: This homework is due at 11:59 PM on Friday, April 6th, but it requires the use of lab machines and the lab will not be open at that time. For lab hours, see (`http://www.mathcs.emory.edu/site/about/computinglab/`). Also, this assignment requires you to run some code that can take a bit of time to execute. Make sure you allow plenty of time to run your code and complete your write-up.

1 Goals

For this assignment, you will implement bubble sort and a non-recursive version of mergesort. You will also compare the average running times of various sorting algorithms on inputs of different sizes, learn and practice how to run scientific experiments, and plot the running time as a chart. Lastly, you will become familiar with using a lab machine (either in person or remotely).

2 Requirements

2.1 Implementation of bubble sort and non-recursive mergesort

The starter code contains a method called `BubbleSort(long[] a)`. You must fill in this method with code that implements a bubble sort to sort an array of long integers.

The starter code contains a method called `MergeSortNonRec(long[] a)`. You must fill in this method with code that implements a non-recursive version of mergesort. In a nonrecursive mergesort, we start by a pass of 1-by-1 merges (merge every two adjacent elements, subarrays of size 1, to make subarrays of size 2), then a pass of 2-by-2 merges (merge subarrays of size 2 to make subarrays of size 4), then 4-by-4 merges and so forth, until we do a merge that encompasses the entire array. You may call the `merge()` that is already implemented in the starter code to accomplish merging. Similar to recursive merge sort, you need to create auxiliary arrays in order to call this `merge()` method. In addition, you need to correctly calculate the `lo`, `mid`, and `hi` indices and pass them to the `merge()` method. Alternatively, you may implement your own version of the `merge()` method if you wish. We will assume that the size of the input is always a power of 2. This reduces the complexity of the solution. If your implementation is correct, it should in fact run slightly faster than the recursive mergesort, because there is no overhead of recursive method calls.

2.2 Comparing Sorting Algorithms

You will compare the average performance of five different sorting algorithms: (1) bubble sort; (2) selection sort; (3) insertion sort; (4) recursive mergesort; and (5) non-recursive mergesort. The selection sort, insertion sort and recursive mergesort are provided in the starter code. The code for running the comparisons for different input sizes are provided in the `main` method. For each algorithm, it tests different input sizes (default is 15 which tests the first 15 powers of 2), and the input data are randomly generated. It performs each test multiple times (default is 5) in order to get an average running time. Afterward, it will print the results as a table: each row reports the input size (N), and the average running time for each algorithm in milliseconds. Familiarize yourself with the code in the `main` method and make sure you understand how the experiments are carried out.

Note that the starter code checks after each sorting algorithm whether the sorting result is correct. If your sorting implementation produces incorrect sorting results, a message 'The sorting is INCORRECT!' will be printed out. Watch for such messages. Points will be deducted if your sorting result is incorrect.

Run the `main` method and examine the table to compare the run times for the various algorithms. You can increase the value of the max variable, but be aware that bubble sort will take quite a bit of time. For consistency, we ask that you run your scientific experiments on a lab computer. You are free to perform the experiment on your own computer to debug, but the final reported times and associated plot needs to be from a lab computer.

2.2.1 Remote Access to Lab Machines

As you need to run the file on a lab machine, you will need to copy files from your home computer to the system at school. You MUST run this homework in the lab to make sure it is correct. This section describes how you can run the experiments even from your own personal computer.

If you are off campus, or not using the Emory Unplugged wireless network: You will need to use the VPN. Detailed instructions about how to access Emory's lab remotely can be found here: (`http://www.mathcs.emory.edu/~cheung/Courses/RemoteAccess/`). Note that if you have never used the VPN before, you have to register your username to be recognized by the VPN. This has to be done during normal business hours. I suggest starting with this step, as to not wait too long to register your name. Even if you are on campus, the information found in the link is helpful.

Copying Files to Account

Mac OS X, Unix and Linux come with the necessary software to do this. The recommended technique is to use the terminal and the `scp` command. The terminal application on Mac OS X can be found in the Applications/Utilities folder. On Linux/Unix you need to just open new terminal. Here is an example of the `scp` (secure copy) command to to copy files from your Mac to the school system into your cs171 folder with the hw4 subfolder (assumes you are already in the directory where the file is located):

```
scp Sorting.java myusername@lab0z.mathcs.emory.edu:cs171/hw6/
```

For Windows users, you can download tiny program called pscp (`http://the.earth.li/~sgtatham/putty/latest/x86/pscp.exe`), which has the same syntax as `scp` for Mac OS X / Unix / Linux. To open a command line window on Windows system, go to Start− >Run... and type cmd.

Remote login to lab machines

MAC OS X / Unix / Linux comes with all the software that you need. Just open the terminal and type what follows:

```
ssh -X myusername@lab0z.mathcs.emory.edu
```

Once you press return, you will be prompted for password. You might also be prompted to verify the key. If so, type yes and press return.

For Windows users, you will need to download an external software (that is free) called PuTTY (`http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe`). All you need to do is to save this file on your computer and run it. Once you start the program just use the following:

1. In Host type: lab0z.mathcs.emory.edu
2. Do not change other default options
3. Click Open (bottom of window)

If PuTTY gives a security alert, click Yes (this might happen, but does not have to). Type your user name and password when prompted (the same as you use to login in lab). Then you will be logged in.

Extremely important: *DO NOT use lab0z.mathcs.emory.edu to run the experiments*, it is only used as a login server. You will need to login again! After you login to lab0z, you **must** ssh to a lab machine using this command (where A = one of the digits 1,2,3,4,5,6,7,8; and B = one of the letters a,b,c,d,e,f,g,h):

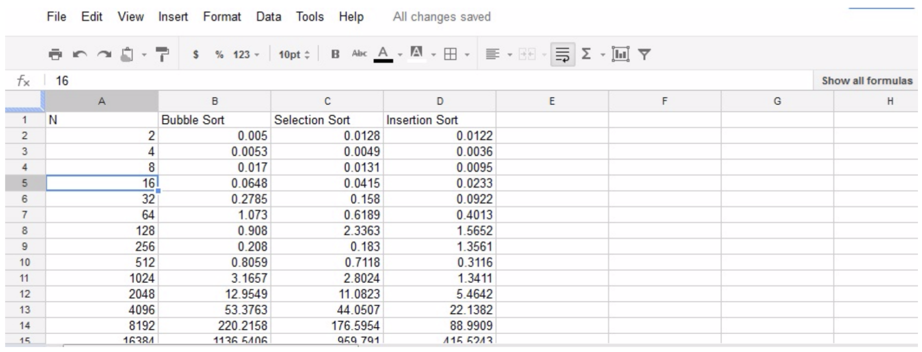
```
ssh -X labAB
```

If you want more information, please see Dr. Cheung's website (`http://www.mathcs.emory.edu/~cheung/Courses/RemoteAccess/index.html`) for more details.

2.2.2 Plotting the Running Times

After the program prints out the table, plot the results in graphs. You can use Google SpreadSheet, Microsoft Excel, OpenOffice, or other tools. The following are the steps using Google Spreadsheet (which plots the results for 3 algorithms). If you use Microsoft Excel, or OpenOffice Calc, the steps should be similar. Just make sure that you select "Line chart" as your chart type and save your chart as an image. In order to use the turnin program, your image file must be a .png with a name "Sorting.png".

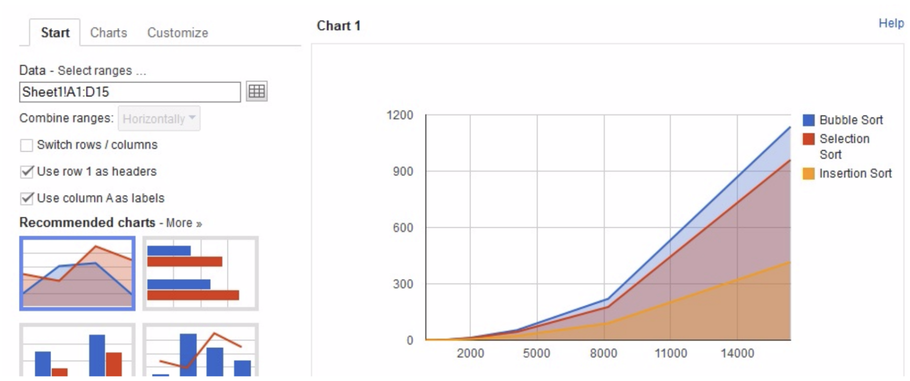
1. Copy the running times reported to a spreadsheet.



N	Bubble Sort	Selection Sort	Insertion Sort
2	0.005	0.0128	0.0122
4	0.0053	0.0049	0.0036
8	0.017	0.0131	0.0095
16	0.0648	0.0415	0.0233
32	0.2785	0.158	0.0922
64	1.073	0.6189	0.4013
128	0.908	2.3363	1.5652
256	0.208	0.183	1.3561
512	0.8059	0.7118	0.3116
1024	3.1657	2.8024	1.3411
2048	12.9549	11.0823	5.4642
4096	53.3763	44.0507	22.1382
8192	220.2158	176.5954	88.9909
16384	1136.5406	959.791	415.5243

2. Click on "Insert" → "Chart", and select "Use row 1 as headers" and "Use column A as labels." Select "Line" chart, and optionally, click on "Customize" and select "Medium" for Point Style. Feel free to add labels (such as chart title), to make the chart more informative. Examine the chart. This chart will help you visualize the difference of the running times of all the sorting algorithms, and also the trend when N grows large. Then click on "Insert" to

insert the chart. Click on the drop down menu from the chart and select "Save image" to save the image to a disk file, and name it Sorting.png. In the future, when you perform scientific experiments, you will frequently plot your data as charts. So take your time to experiment with different types of charts and different ways of plotting them. Make sure that somewhere on your plot, you put what lab machine the experiments were performed on.



3 Getting started

1. Download the starter code `Sorting.java` on Canvas and understand it.
2. Complete the methods `BubbleSort` and `MergeSortNonRec`, which should implement the bubble sort and non-recursive version of merge sort, respectively.
3. Run the starter code to compare the running times for different sorting algorithms (you can change the parameters "max" and "runs" for testing purposes).
4. Plot the performance chart. Make sure the numbers you use are from a lab machine.

4 Submission Instructions

The programming assignment should be submitted electronically by the turnin program. The instructions are from a terminal run the following commands:

```
>cd cs171
>/home/cs171003/turnin Sorting.java hw6a
>/home/cs171003/turnin Sorting.png hw6b
```

Make sure you submit **both** the `Sorting.java` file and the `Sorting` plot. On your plot, be sure that it is clear the name of the lab machine this was run on.

5 Grading

- Correctness and robustness: your bubble sort and non-recursive merge sort works correctly for any input array (that has a size of a power of 2) (60 points)
- Efficiency: your non-recursive mergesort is efficient, and the running time is less than the recursive mergesort in most cases when $N > 512$ (20 points)

- Correctness of the plot: you included a running time chart and the data is correctly plotted (15 points).
- Coding style (5 points)

Warning: If you *fake the numbers* so that your non-recursive merge sort is more efficient when it is not, you will *not only lose points on efficiency but also correctness of the plot (~35 points).* If your implementation is not as efficient, but your plot correctly reflects your code, you will get full points on the correctness of plot and lose points only on efficiency (≤ 20 points).

6 Honor Code

The assignment is governed by the College Honor Code and Departmental Policy. Please remember to have the following comment included at the top of the file.

```
/*
THIS CODE WAS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING
CODE WRITTEN BY OTHER STUDENTS. _Your_Name_Here_
*/
```