CS 171 (Summer 2022)
Assignment 4: Bubble Sort and Mergesort
Due: 7/28/2022 (11:59pm)

Goals

- Implement bubble sort and a non-recursive version of mergesort.
- 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.

Requirements:
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 are 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 an 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. Comparing Sorting Algorithms

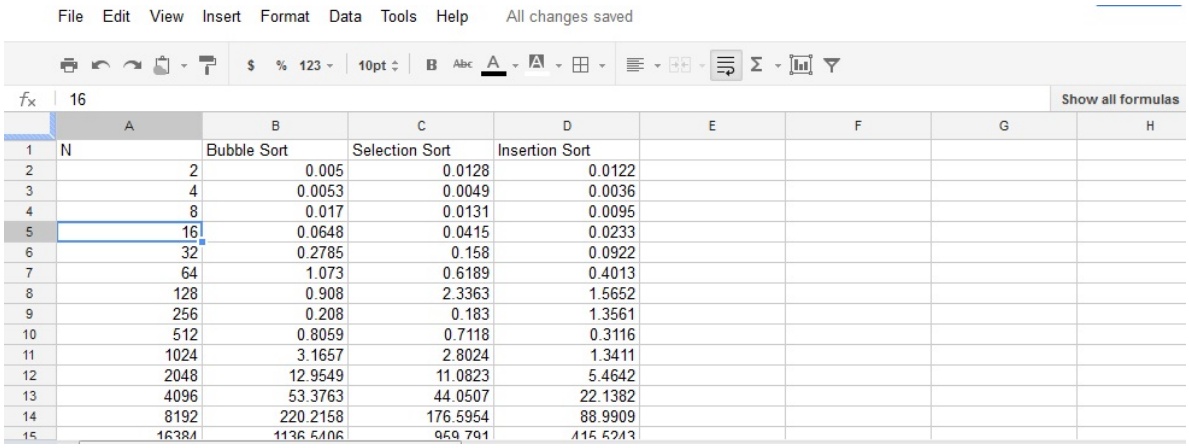
You will compare the average performance of five different sorting algorithms: bubble sort, selection sort, insertion sort, recursive mergesort, and nonrecursive mergesort. The selection sort, insertion sort and recursive mergesort are provided in the starter code. The code for running the comparisons 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. 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.

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. So watch for such messages. Points will be deducted if your sorting result is incorrect.

2.1 Plotting the running times

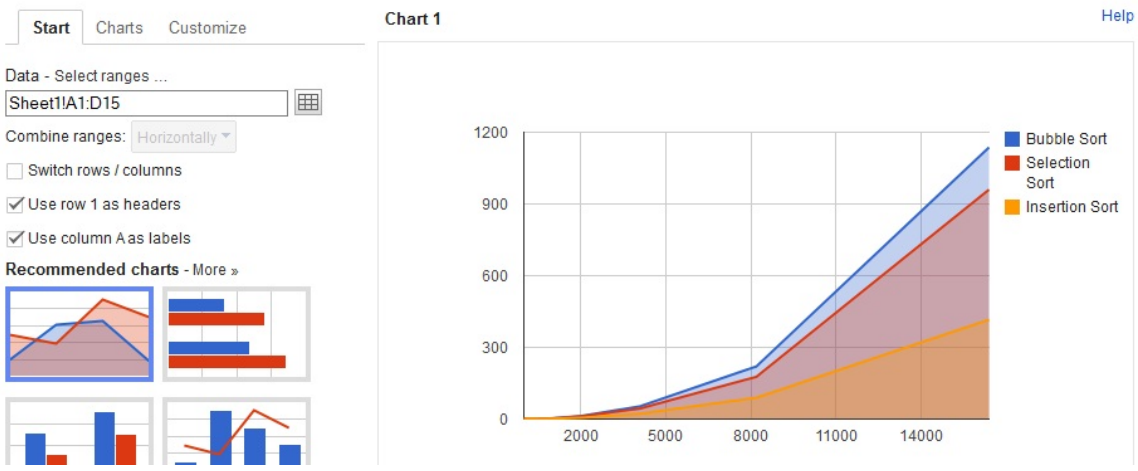
After the program prints out the table, plot the results in graphs. You can use Google SpreadSheet, Microsoft Excel, OpenOffice, or other tools. The following are the steps using Google Spreadsheet (which plots the results for 3 algorithms). If you use Microsoft Excel, or OpeenOffice Calc, the steps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

(a) Copy the running times reported to a spreadsheet.



	A	B	C	D	E	F	G	H
1	N	Bubble Sort	Selection Sort	Insertion Sort				
2	2	0.005	0.0128	0.0122				
3	4	0.0053	0.0049	0.0036				
4	8	0.017	0.0131	0.0095				
5	16	0.0648	0.0415	0.0233				
6	32	0.2785	0.158	0.0922				
7	64	1.073	0.6189	0.4013				
8	128	0.908	2.3363	1.5652				
9	256	0.208	0.183	1.3561				
10	512	0.8059	0.7118	0.3116				
11	1024	3.1657	2.8024	1.3411				
12	2048	12.9549	11.0823	5.4642				
13	4096	53.3763	44.0507	22.1382				
14	8192	220.2158	176.5954	88.9909				
15	16384	1136.5406	959.791	415.5243				

(b) 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.



Getting Started

1. Get the starter code `Sorting.java` from `~cs171001/share/hw4` directory and understand it.
2. Complete the method BubbleSort and MergeSortNonRec(), which should implement the bubble sort and non-recursive version of mergesort.
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 charts. You should create a plot with the running time of all algorithms plotted, and a second plot with only the recursive and nonrecursive mergesorts.

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_
*/
```

Submission:
Place your completed Sorting.java file directly under your `~/cs171/hw4` directory. Then use the turnin command:

```
~cs171001/turnin Sorting.java hw4
```

Upload your chart and plots to Canvas as Homework 4.

Grading:

- Correctness and robustness: your bubble sort and non-recursive mergesort works correctly for any input array (that has a size of a power of 2). (60 pts.)
- Efficiency: your non-recursive mergesort is efficient, and the running time is less than the recursive mergesort in most cases when N $>$ 512 (20 pts.)
- Code clarity and style (5 pts)
- Correctness of the plot: you included a running time chart and the data is correctly plotted. (15 pts.)