Lab 6 - Menagerie of Sorts

CSC 172 (Data Structures and Algorithms)

Due Date: Sunday, 04/10 11:59 PM

Every student is encouraged (but not strictly required) to have a lab partner. Every student must hand in their own work but also list the name of their lab partner, if any.
In this lab, we will work on our understanding of asymptotic analysis of sorting algorithms. We will run implementation of various sorting algorithms on different datasets to determine their runtimes and how closely result follows the asymptotic runtime for that algorithm. This lab uses files you can find in attachment to the post.

The *zip* file contains 7 data files
- 1Kints.txt: contains 1K integers
- 2Kints.txt: contains 2K integers
- 4Kints.txt: contains 4K integers
- 8Kints.txt: contains 8K integers
- 16Kints.txt: contains 16K integers
- 32Kints.txt: contains 32K integers
- 1Mints.txt: contains 1M integers

It also contains **`Sorting.java`** which you need to modify and submit. The program takes two arguments: the first is the input file name (e.g., *4Kints.txt*) and the second is an integer between 0 and 5 indicating the sorting algorithm to execute based on the following list.

- 0 implies Arrays.sort() (Java Default)
- 1 implies *Bubble Sort*
- 2 implies *Selection Sort*
- 3 implies *Insertion Sort*
- 4 implies *Mergesort*
- 5 implies *Quicksort*

Your final submission must contain your *code* and a lab *report* describing your finding.

Step 1

Open the input file given as the first argument to the function and generate four integer arrays from the integers the file contains. These four arrays (a, b, c, and d) must have the following properties:

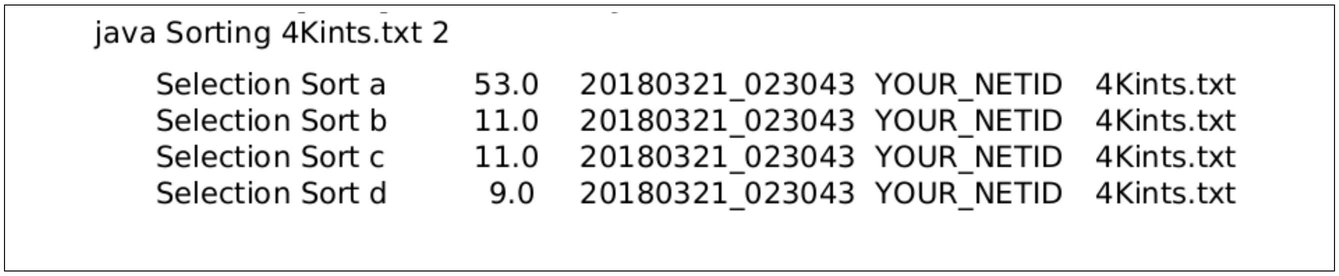
1. a: Random numbers. Same ordering as the input file
2. b: Sorted (using Arrays.sort() method)
3. c: Sorted in reverse order
4. d: Almost sorted. You need to perform (0.1 * length of the array) number of swaps to achieve this.

As a dummy example, if the input file had contained 10 random integers from 0 to 9, then these four arrays
would look like:

1. a: 3, 8, 0, 2, 5, 9, 4, 6, 7, 1
2. b: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
3. c: 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
4. d: 0, 1, 2, 3, 9, 5, 6, 7, 8, 4 (Note: only one swap (4 and 9) is performed as 10 * 0.1 = 1. Also note that for each run, the array d could be different)

Step 2
Read the second input parameter and based on the value (between 0 and 5 each representing a particular sorting algorithm as mentioned earlier), perform that sorting on four arrays namely, a, b, c, d those you have created as per **Step 1**. Print on console the name of the algorithm used, the array used (a, b, c, or d), time-elapsed (in milliseconds), timestamp, your NetID and the input file used.

Here is a sample output for the following run:



```
java Sorting 4Kints.txt 2
     Selection Sort a      53.0   20180321_023043  YOUR_NETID   4Kints.txt
     Selection Sort b      11.0   20180321_023043  YOUR_NETID   4Kints.txt
     Selection Sort c      11.0   20180321_023043  YOUR_NETID   4Kints.txt
     Selection Sort d       9.0   20180321_023043  YOUR_NETID   4Kints.txt
```

Take a screenshot of your output. Also, your program should save the sorted output in four different files as a.txt , b.txt , c.txt , d.txt for arrays a, b, c, and d respectively for each run. Note that you can overwrite these files for each run. This is just to check the correctness of your program. You do not need to submit these '.txt' files.

Important note:

1. You need to change the code so that it prints your *NetId*
2. You MUST provide screenshots (copying the output as text is not allowed.)
3. You MUST generate four (4) output files for each run.

Step 3

Now you will produce a report summarizing your findings.
1. Provide four tables (for four arrays a, b, c, and d respectively). Each row (or record) should provide the runtime for using any particular sorting algorithms, where the columns are for the size of the input.
Each table should have six (6) rows (representing six sorting algorithms) and seven(7) columns (representing seven data files). Save the screenshots and tables in your report *Lab6Report.pdf*

2. (3 *extra points*) Generate four plots for four arrays. Use the data from the earlier table to plot the relative performance of each algorithm. Use logarithmic scale for x-Axis depicting input size. Include plots in your report.

Note: If any of these algorithms for any particular array runs for more than 10 minutes, you can terminate the process and state that in your report.

Submission

Submit a single zip file Lab6.zip containing the source code (Sorting.java) and the lab report Lab6Report.pdf at the appropriate location on Blackboard. No other files or folders are allowed.
Grading (10 pts + 3 extra points)
Your lab will be graded solely on the source code and the pdf file you have submitted.
Lab Report: 4 pts + 3 pts extra for plots
Code: 6 pts

Testing Notes (important)

Our testing script will test the methods as follows:

1. `Sorting.defaultSort(int[])`
2. `Sorting.bubbleSort(int[])`
3. `Sorting.selectionSort(int[])`
4. `Sorting.insertionSort(int[])`
5. `Sorting.mergeSort(int[])`
6. `Sorting.quickSort(int[])`

Please use the same names for methods in order to facilitate the process and to avoid point deduction. Declare the methods as `public` to avoid access issues.

Notes on collaboration and use of external sources:

All labs are open book. You can use the code given in lectures with proper citation. Use of other sources is not permitted.