Ryan Cocuzzo
Professor Zhupa
CSC172
4 November 2018

Lab 5 Report

Overview

This lab is a culmination of the comparison-based sorting algorithms we have been covering as a class. Having covered as a class the proof that we cannot do better in a comparison-based sorting algorithm than $\boldsymbol{O}(n\log n)$, sorting algorithms covered in this lab carry that innate limitation. The algorithms covered are: Bubble Sort, Insertion sort, Selection sort, Quick Sort, Merge Sort, as well as the default Arrays.sort() function in Java, all of which deviate somewhere between $\boldsymbol{O}(n\log n)$ and $\boldsymbol{O}(n^2)$. This project will take an inputted file path/file name to a file containing integer inputs and an integer representation of the desired sorting algorithm to use and it will output the amount of time taken to sort the array.

Process and Output Consideration Factors

In the making of this lab, maintenance of developer-friendly and human-readable algorithms was given high consideration and priority. Each algorithm was designed to specification of each algorithm's main properties and goals but was also highly tailored to allow a more readable and simplistic experience when looking over the code. This was supported by the usage of many custom-made helper functions to off-load code when performing minute tasks and this process, in my personal opinion, greatly enhanced the legibility of the project.

In this project, however, a setback that was given high priority was runtimes not displaying adequately. As can be noticed in the data being displayed at a later part of this lab report, only portions of the runtimes that come out appear to be feasible numbers, and, cumulatively, this inhibits the quality of information we can gather as to Big-O runtimes of this project. The correctness of the algorithms both in theory and in implementation imply trends of their respective runtimes, which is reflected partially, but still, to some extent, illegibly in the outputted data. I do hope for greater consideration in the correct and well-outlined code and surrounding elements of the project as a reflection of the quality of the project as the project was completed with great effort and care.

The inputs, for the greater part, worked perfectly well for my personal computer with no issue, however, the 1M.txt input file could not be used as it would take an enormous amount of time and computing power, overclocking the CPU and making testing incredibly difficult. This being the case, I omitted it from the data.

Files included are only ones obtained from the source provided in the assignment, no more nor less. Simply using the Sorting.java file was possible had the assignment not come with the In.java, StdRandom.java, Stopwatch.java, and StdOut.java files as well as partial implementations for use, therefore those files were included in my implementation.

Output

The following output was created to illustrate each possible case of both input and sorting algorithm. Additional permutations can be derived from the two sets, this is simply sample output that illustrates the correctness of the program in achieving the desired goal in a timely manner.

```
Last login: Sun Nov  4 18:40:52 on ttys000
[dhcp-10-5-31-132:src Ryan$ javac Sorting.java
[dhcp-10-5-31-132:src Ryan$ java Sorting 4Kints.txt 2
Selection Sort a     26.0   20181104_204251  RCOCUZZO  4Kints.txt
Selection Sort b     10.0   20181104_204251  RCOCUZZO  4Kints.txt
Selection Sort c      5.0   20181104_204251  RCOCUZZO  4Kints.txt
Selection Sort d      2.0   20181104_204251  RCOCUZZO  4Kints.txt
[dhcp-10-5-31-132:src Ryan$ java Sorting 4Kints.txt 3
Insertion Sort a     33.0   20181104_204458  RCOCUZZO  4Kints.txt
Insertion Sort b     13.0   20181104_204458  RCOCUZZO  4Kints.txt
Insertion Sort c      3.0   20181104_204458  RCOCUZZO  4Kints.txt
Insertion Sort d      5.0   20181104_204458  RCOCUZZO  4Kints.txt
[dhcp-10-5-31-132:src Ryan$ java Sorting 16Kints.txt 2
Selection Sort a    114.0   20181104_204515  RCOCUZZO  16Kints.txt
Selection Sort b     74.0   20181104_204515  RCOCUZZO  16Kints.txt
Selection Sort c     90.0   20181104_204515  RCOCUZZO  16Kints.txt
Selection Sort d     38.0   20181104_204515  RCOCUZZO  16Kints.txt
[dhcp-10-5-31-132:src Ryan$ java Sorting 1Kints.txt 4
Merge Sort a      1.0   20181104_204546  RCOCUZZO  1Kints.txt
Merge Sort b      0.0   20181104_204546  RCOCUZZO  1Kints.txt
Merge Sort c      1.0   20181104_204546  RCOCUZZO  1Kints.txt
Merge Sort d      0.0   20181104_204546  RCOCUZZO  1Kints.txt
[dhcp-10-5-31-132:src Ryan$ java Sorting 32Kints.txt 0
.sort() Sort a      2.0   20181104_204617  RCOCUZZO  32Kints.txt
.sort() Sort b      1.0   20181104_204617  RCOCUZZO  32Kints.txt
.sort() Sort c      1.0   20181104_204617  RCOCUZZO  32Kints.txt
.sort() Sort d      2.0   20181104_204617  RCOCUZZO  32Kints.txt
[dhcp-10-5-31-132:src Ryan$ java Sorting 8Kints.txt 4
Merge Sort a      3.0   20181104_204645  RCOCUZZO  8Kints.txt
Merge Sort b      2.0   20181104_204645  RCOCUZZO  8Kints.txt
Merge Sort c      2.0   20181104_204645  RCOCUZZO  8Kints.txt
Merge Sort d      2.0   20181104_204645  RCOCUZZO  8Kints.txt
[dhcp-10-5-31-132:src Ryan$ java Sorting 16Kints.txt 1
Bubble Sort a    277.0   20181104_204705  RCOCUZZO  16Kints.txt
Bubble Sort b    109.0   20181104_204705  RCOCUZZO  16Kints.txt
Bubble Sort c     59.0   20181104_204705  RCOCUZZO  16Kints.txt
Bubble Sort d    163.0   20181104_204705  RCOCUZZO  16Kints.txt
[dhcp-10-5-31-132:src Ryan$ java Sorting 2Kints.txt 5
Quick Sort a     12.0   20181105_002433  RCOCUZZO  2Kints.txt
Quick Sort b     10.0   20181105_002433  RCOCUZZO  2Kints.txt
Quick Sort c     10.0   20181105_002433  RCOCUZZO  2Kints.txt
Quick Sort d      8.0   20181105_002433  RCOCUZZO  2Kints.txt
dhcp-10-5-31-132:src Ryan$
```

Data Collection Results

As mentioned earlier, some of the output numbers appear to look quite odd. The sorting algorithms work with complete and testable correctness and the process of timing them is done not only clearly and separate from sorting, but was done using the code provided in the source. This being the case, the data is legitimate and thorough, but certainly does show room for improvement on a code level. As can be seen, however, in the data visualizations, some of the data collected is what should be estimated and illustrate textbook trends that are looked for when assessing runtime complexities of these algorithms as they scale, which is exactly what this project aims to do.

A

Algorithm	1K	2K	4K	8K	16K	32K	1M
.Sort()	0	0	0	0	2	3	N/A
Bubble	4	3	9	39	165	694	N/A
Selection	3	1	3	9	28	112	N/A
Insertion	4	1	7	21	87	349	N/A
Merge	1	0	0	1	2	4	N/A
Quick	5	8	24	166	418	2330	N/A

B

Algorithm	1K	2K	4K	8K	16K	32K	1M
.Sort()	0	0	0	0	1	1	N/A
Bubble	1	1	3	12	48	195	N/A
Selection	2	1	2	8	28	117	N/A
Insertion	1	2	9	29	116	471	N/A
Merge	1	1	0	1	1	1	N/A
Quick	2	10	39	160	705	2841	N/A

C

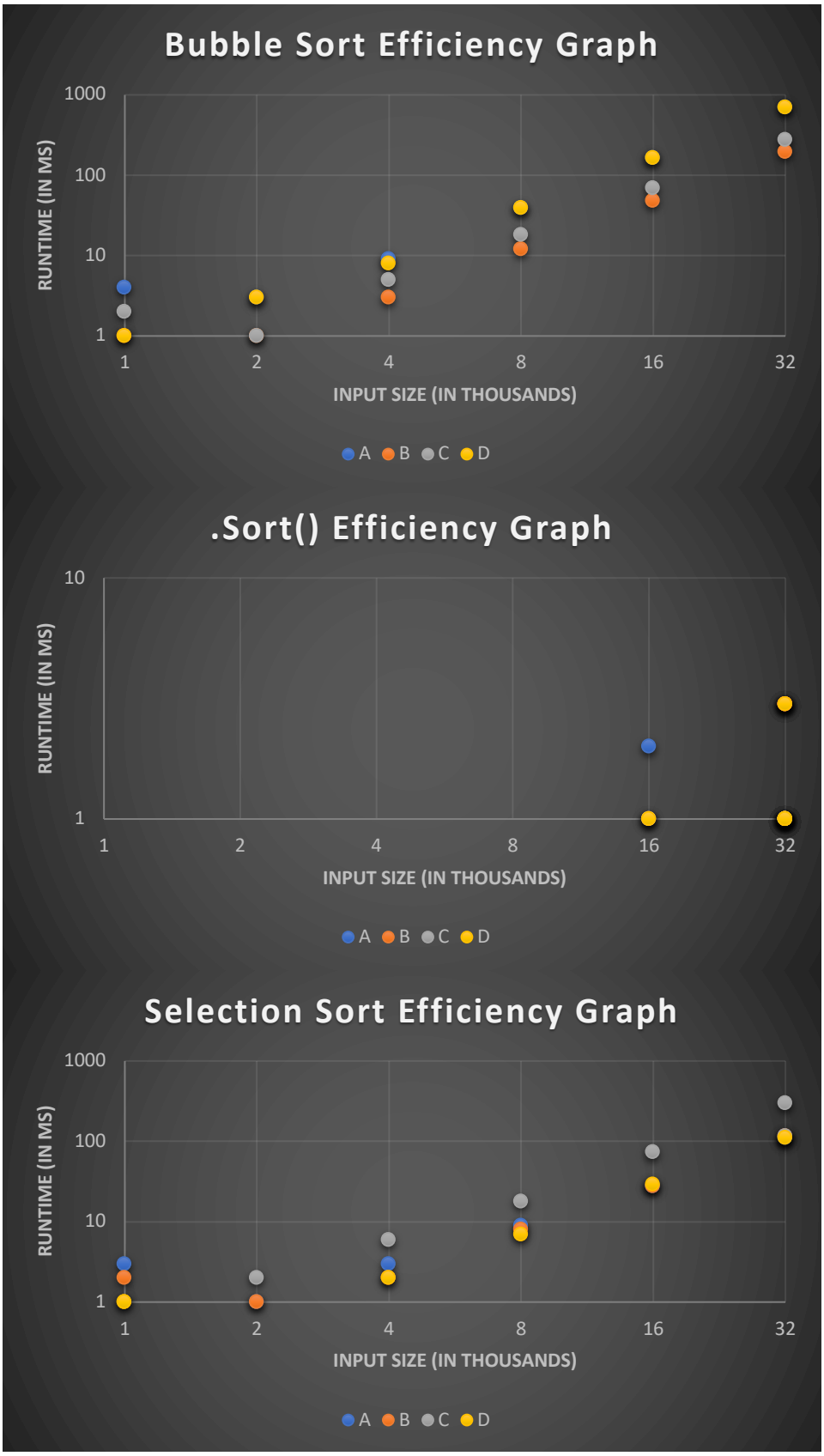
Algorithm	1K	2K	4K	8K	16K	32K	1M
.Sort()	0	0	0	0	1	1	N/A
Bubble	2	1	5	18	69	277	N/A
Selection	0	2	6	18	74	302	N/A
Insertion	1	1	4	16	57	220	N/A
Merge	1	0	0	1	1	2	N/A
Quick	2	10	38	168	676	2899	N/A

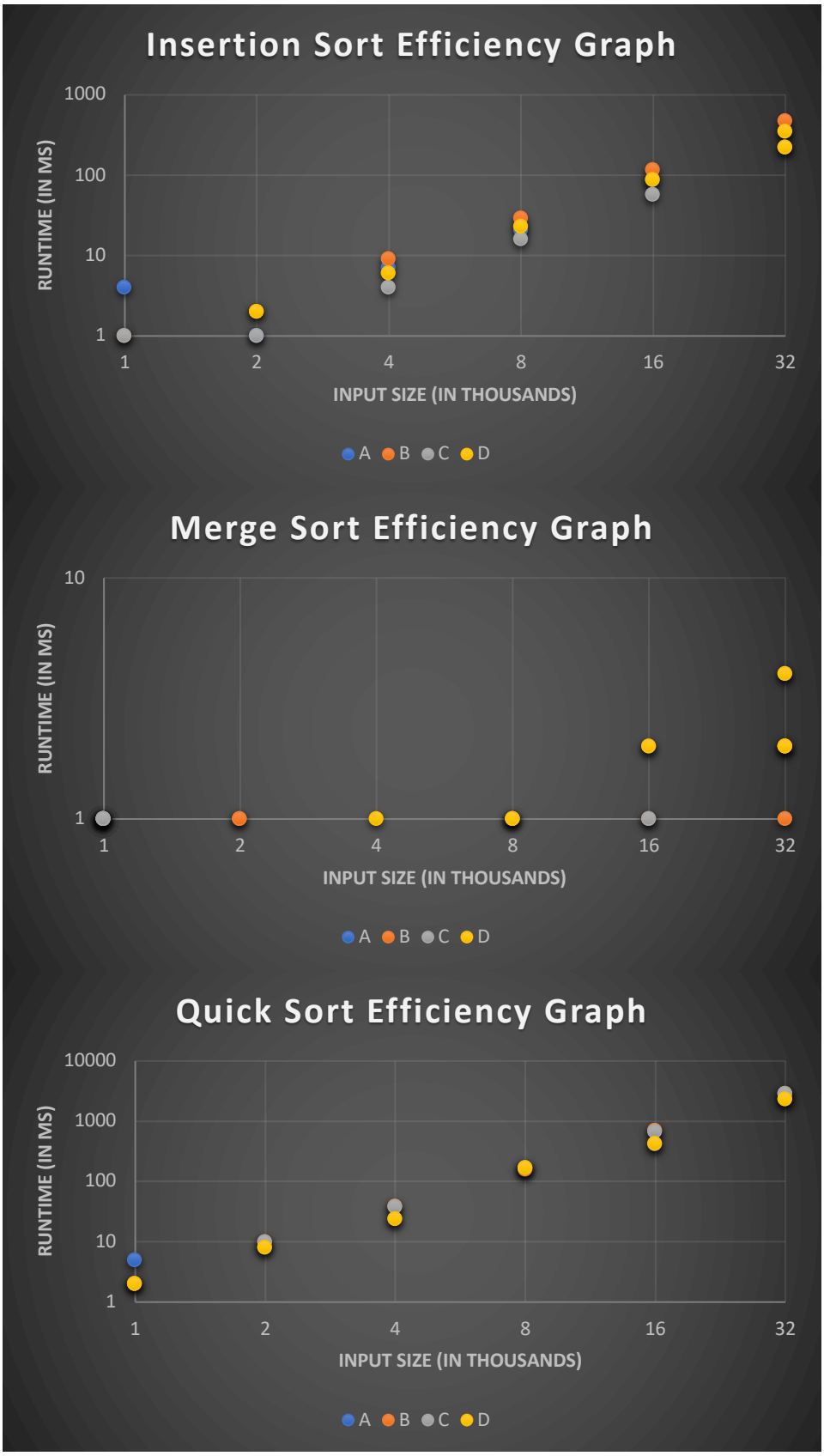
D

Algorithm	1K	2K	4K	8K	16K	32K	1M
.Sort()	0	0	0	0	1	3	N/A
Bubble	1	3	8	39	165	695	N/A
Selection	1	0	2	7	29	111	N/A
Insertion	0	2	6	23	88	348	N/A
Merge	0	0	1	1	2	4	N/A
Quick	2	8	24	170	425	2349	N/A

Data Visualization / Extrapolation (Extra Credit)

As per the extra credit opportunity provided in the assignment, below are many beautiful and concise visualizations of the data in question. These visualizations, as per the extra credit section's notes, was done in logarithmic scale to allow the data to show a more clear linear relationship. While data collection had troubles outlined earlier, many of these findings show trends that are exactly the prescribed and desired goal when reflecting on their respective algorithm's efficiency.


Bubble Sort Efficiency Graph
1000
100
10
1
RUNTIME (IN MS)
1
2
4
8
16
32
INPUT SIZE (IN THOUSANDS)
A B C D
.Sort() Efficiency Graph
10
1
RUNTIME (IN MS)
1
2
4
8
16
32
INPUT SIZE (IN THOUSANDS)
A B C D
Selection Sort Efficiency Graph
1000
100
10
1
RUNTIME (IN MS)
1
2
4
8
16
32
INPUT SIZE (IN THOUSANDS)
A B C D


Insertion Sort Efficiency Graph
RUNTIME (IN MS)
1000
100
10
1
1
2
4
8
16
32
INPUT SIZE (IN THOUSANDS)
A B C D
Merge Sort Efficiency Graph
RUNTIME (IN MS)
10
1
1
2
4
8
16
32
INPUT SIZE (IN THOUSANDS)
A B C D
Quick Sort Efficiency Graph
RUNTIME (IN MS)
10000
1000
100
10
1
1
2
4
8
16
32
INPUT SIZE (IN THOUSANDS)
A B C D

Compile and Run

Place an input file into the project's directory and open the project's directory via the terminal. From the terminal session enter the following commands:

$ javac Sorting.java
$ java Sorting <input-file> <sorting algorithm # (0-5)>

Ex.
$ javac Sorting.java
$ java Sorting 2Kints.txt 3