Lab 3 (Analysis of Sorting Algorithms)

CSC 172 (Data Structures and Algorithms)
Fall 2024
University of Rochester
Due Date: Sunday, September 22 @ End of Day

Introduction

The labs in CSC172 will follow a pair programming paradigm. 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 on all labs.

In this lab, we will work on our understanding of asymptotic analysis of 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 the same files you used for Lab 2. For your convenience, I have uploaded all the necessary files again. You can download the zip files from `http://www.cs.rochester.edu/courses/172/spring2018/labs/Lab03.zip`.

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 (for example, 4Kints.txt) and the second is an integer between 0 and 5 indicating the sorting algorithm to perform based on the following list. You may include Insertion Sort and Mergesort for extra credit, but they are not required for this assignment.

- 0 implies Arrays.sort() (Java Default)
- 1 implies Bubble Sort
- 2 implies Selection Sort
- 3 implies Quicksort

Your final submission must contain your code and a lab report describing your finding. More details to follow.

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) 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 perform 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 millisecond), timestamp, your classID, 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.

Note:
1. You need to change the code so that it prints your NetId (not mine!)
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. Please refer: `http://lti.cs.vt.edu/LTI_ruby/Books/CS172/html/SortingEmpirical.html` Table 9.15.1 for the format. Each table should have six (6) rows (representing six sorting algorithms) and seven(7) columns (representing seven data files).
2. 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. Comment on your finding. Save the screenshots, tables, plots, and your findings as `Lab7Report.pdf`.

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

Hand in a single zip file `Lab3.zip` containing the source code (Sorting.java) and the lab report `Lab3Report.pdf` at the appropriate location on the Blackboard system. No other files or folders are allowed. We will deduct points for not following the instructions.

Grading (100 pts)

Your lab will be graded solely on the source code and the pdf file you have submitted.

Lab Report: 50 pts
Code: 50 pts

All labs are open book. You can get code snippets from the internet if you want (make sure you cite those properly). But that is not the purpose. We want you to sit together, think about an algorithm, and then implement it together with your partner (that sounds fun! isn't it?)