Project 1 – Experimental Analysis of Sorting Algorithms

# Project 1A is due on Sept 11 and Project 1B is due on Sep 25. Project 1C later.

## General Notes

* For this project, by sorting we mean sorting in **ascending** order.
* Submit all source code, program output, and project report electronically on Canvas as a Zip file.
* Allowed programming languages: Python, Java, C++/C, or others *with prior approval from the instructor*.
* Implementations should be as optimized as possible.
* When measuring execution time, do not include the time for reading the input data from disk. Only include the time for the actual algorithm to execute!
* **You are not allowed to use online resources when writing source code solutions.**

## Academic Honesty Policy

For this project, you are not allowed to discuss with any other students currently in the course or with any person outside the course. **You are not allowed to use online resources when writing source code solutions.**  The only resource you may utilize is the pseudo-code given in the textbook or the lecture notes.

We use a plagiarism detection tools on all submitted source code. It is capable of finding portions of copied code from other students in the course, other sections of the course, previous offerings of the course, as well as Internet sources.

## Instructions

In this project you will implement several sorting algorithms and then analyze the performance of those algorithms on sample input data. You will then attempt to infer the complexity of each algorithm using that analysis and provide a report describing each algorithm’s observed performance and a comparison to the theoretical/expected performance.

The following sorting algorithms **must** be implemented:

1. Insertion sort (as described in the textbook, Page 18)
2. Merge sort (as described in the textbook, Page 34)
3. Heapsort (as described in the textbook, Page 160)
4. [**CS** **5120 ONLY**] 4-way Merge sort.

The emphasis of the project is on experimental analysis of running time ***and*** number of comparisons made. You need to track the time ***and*** the number of comparisons for sorting each data set and record them in units that are fine-grained enough to allow meaningful comparison. For example, the unit for time should be micro- or possibly even nano-seconds on fast machines.

You should submit exactly 3 (or 4 for CS 5120) main methods (in Java, 4 classes [source file and class file] with main methods, in C++ 4 different projects [with source code and executable], in Python 4 different [source] files with a main, etc)! If you fail to follow these instructions, you will lose 20% of your points!

Your sorting algorithms will be tested for correctness, so please ensure they work! This includes edge cases like inputs with a single value, odd number of values, even number of values, etc.

## Expected Input Data Format

**Do not hard code file paths in your programs!** Your program should take a single argument on the command-line, which is the path to a file to read as input. **We will use our own files to test your programs.** Failure to follow these instructions will lose you 20% of your points!

You must first create the following types of input data, of sizes 15,000, 30,000, 60,000, 120,000, and 240,000 and store them in separate files:

1. random data
2. sorted random data
3. reverse (descending) sorted random data
4. all identical data

Thus, there are 5 sizes in the above 4 categories yielding a total of 20 input data sets (files). Create these 20 sets of input data once and use the same data for each algorithm.

Files should be text files and contain 1 number per line. Numbers should be integers in the **range of 0 through 1 million**. Your algorithm implementations should expect and accept input in this format**.**

We provide a sample Python program to generate such datasets. The program can be called as follows:

* python3 genDataset.py 30000
  + generates a random list of 30000 integers
* python3 genDataset.py 30000 --sorted
  + generates a random list of 30000 integers, and outputs them sorted
* python3 genDataset.py 30000 --rsorted
  + generates a random list of 30000 integers, and outputs them reverse sorted
* python3 genDataset.py 30000 21345
  + generates a list of 30000 copies of the number 21345

This is just an example program but its usage is not required. If you use another program to generate the data sets, please submit it. Your generator must conform to the specified data format above.

## Expected Algorithm Output Format

The output of your program should be three values on a **single line, comma separated**: size of the input, number of comparisons made while sorting the input, and the execution time (in nano/milliseconds). There should be **no other output**. This helps ease your data collection, allowing you to easily script runs and copy/paste the results into Excel. For example, the output might look like the following:

30000, 213455, 68379  
30000, 214732, 69038  
30000, 212818, 69287  
30000, 213044, 68785  
. . .

**Our testing scripts will also expect this form of output and break on anything else.**

## What to Submit

There are three parts (Project 1A, 1B, and 1C) of the project with different due dates. See the rubric file for details.

The general guideline is as follows.

You should submit an archive (ZIP, TAR/GZip, etc) of all files, including the following:

1. All source code for all sorting algorithms. Make sure each algorithm can be executed as its own executable/main method and accepts a path to a file to use as input.
2. Makefile (or simple shell scripts) to compile the source code and execute the algorithms.
3. A README.txt file including instructions on how to compile and run your source code, any non-standard libraries used, etc.
4. The 20 input data sets generated and used in your analysis and report.
5. Your input dataset generator (if not using our python program).
6. Runs showing output (not the sorted output, but the results, namely the running time and the number of comparisons) for each sorting routine for each data set of each type. This can be saved into either a text file (with file extension .csv) or an Excel worksheet.
7. A project report (Word or PDF) containing (at a minimum) the following:
   1. Expected/theoretical performance (e.g., theta notation) for each algorithm.
   2. A table summarizing the observed results of each algorithm.
   3. A graphical representation of the data (graphs).
   4. Analysis and Discussion of ***your results*** vis-à-vis the expected results.
      1. Are the theoretical and actual performance results consistent?
      2. Any anomalies?
      3. What surprised you?

In Item 7, make sure you tell a story and cover all points.