Johns Hopkins University, Whiting School of Engineering
605.621 Foundations of Algorithms
Programming Assignment Guidelines

The programming assignments in this course are designed such that you will present algorithms, analyze them theoretically, implement your proposed algorithm in Java, and then test the implementation on data sets of your choosing. The source code and supporting materials shall be zipped into an archive file with filename of form "lastname_assignmentNum.zip" (or ".tar.gz", as appropriate). Please submit:

- **Source code (i.e., the development part)**
  - A main file that runs within the Java project you create.
  - All supporting files for the main class should be submitted. Don't include compiled object files.
  - Code must be documented; please use methods and function names that are "self-documenting". For example, use "index" rather than "i" or "data[]" rather than "a[]".
- **Analysis (i.e., the science part)**
  - A README file that includes:
    - The version of your Java runtime, and how to run your code.
    - Trace runs that demonstrate proper functioning of your implementation (including inputs, outputs, and intermediate results demonstrating proper functioning).
  - A report that includes for each problem:[1]
    - Pseudocode, if the problem asks for an algorithm.
    - Proof of correctness of pseudocode.
    - Proof of theoretical runtime of pseudocode.
    - Empirical measurement of runtime, compared with the theoretical runtime that you proved from the pseudocode. **Note:** Do not measure runtime using wall clock time (i.e., minutes and seconds).[2] Instead, measure the unit of work that you bounded in your theoretical analysis (e.g., comparisons). A graph or two is often a great way to compare theoretical and empirical scaling, or even fitting the experimental data with the theoretical scaling function (e.g., n^2) and showing good fit.

[1] The report structure for programming assignments mirrors that approach that is used when one is performing research into new algorithms, and is a good pattern to use in general when analyzing novel computational approaches.

[2] Due to the amount of performance variability on modern processors, it is very hard to measure wall clock time and compare it to theoretical results. This is due to the behavior of CPU frequency throttling, context switches by the operating system, cache behavior, the action of prefetchers, and more. Measuring energy and power is often even tougher. Typically, algorithms researchers will choose a proxy unit of analysis (e.g., array accesses or floating point operations) that can be used for theoretical bounds *and* measured for empirical confirmation. Of course, an argument must be made for the suitability of the proxy metric (e.g., array accesses) in place of the target metric (e.g., wall clock time).