General Submission Guidelines

Files. C or C++ code files, and a PDF report file should be submitted on Gradescope.

Submission Code Structure. Make a different sub-directory for each question with the name question#. Every question sub-directory should be independently buildable. You can re-use files through symlinks or relative paths in your Makefile. A simple flat structure is preferred. For example,

```
hw3
|--- question1
     |--- main.cpp
     +--- Makefile
+--- question2
     |--- main.cpp
     +--- Makefile
```

Development and Testing. You can develop and test your code on your local machines with gcc. However, before submitting your homework, make sure your programs compile and run on mc19.cs.purdue.edu.

Grading. mc19 machine will be used for testing your code with GNU compiler. Output your results to `stdout` using `printf()` or `cout`. If your code needs a different build process other than the one we will use as default, you will have to include a Makefile in the question sub-directory with the default target building the executable.

Extensions. No late submissions will be accepted, unless students have prior extensions from the instructors.

Homework Specific Guidelines

Build Command. By default, we will compile all of your C/ C++ files in each question sub-directory with the command `mpic++ *.cpp -o ./run`. All of your source files in the question's sub-directory will be built into an executable that outputs to `stdout`. If you do not want to include a faulty source file for grading, remove it from the sub-directory; otherwise it will fail the default build command.

If you do not want to us to use the default build command, include a Makefile in each question's sub-directory. However, it is recommended to use the simplest flat-file structure for your code as shown in the general submission guidelines and thus focus on experiments and results, not software engineering. You will still need to create sub-directories for questions.

Tools. You cannot use any language other than C/ C++ for implementing the algorithms. Data analytics/ plotting can be done with other tools. Excel or Google Sheets are the easiest to use for plotting graphs.

Execution. If your program uses command line arguments or some sort of input, you need to explain how to use your program in your report PDF. It is preferable if you avoid command line arguments, or automate the inputs to these arguments in a Makefile target or shell script included with your submission (e.g. you can include a shell script called `run.sh` that runs the program with different command line arguments).

Output. We will execute your programs on mc19. Make sure your code compiles, runs, and does not throw errors like segfaults on the test environment.

Question 1 (50 points) Implement the quicksort algorithm of Slides 60, 61 of sorting slide-deck on the website.

Test your implementation on a randomly generated set of integers, starting from a $p = 1$ case of $n = 10M$ entries.

Draw efficiency curves for the following cases as you increase p:

1. constant problem size (W)
2. linearly increasing number of elements (n)
3. problem size increasing at the rate of isoefficiency

(Remember, we define problem size as the asymptote of W, which is $n \log n$, not as input size n)

You can use mc servers or your local machines.

Question 2 (50 points) Implement sample sort algorithm starting slide 63. Repeat the experiments and efficiency curves above.

Warning Range p from 1 to maximum available slots on your machine (64, if possible). Step size can be 1 or 2. Do **NOT** use sizes of p that are powers of 2 (2, 4, 8, 16, 32). This will lead to deduction of 25 points. These graphs should have a lot of data points.