Overview. The goal of the semester project is for you to identify, research, implement, and report on an algorithm of interest to you. The project is broken into multiple steps, each of which have either a check-in or deliverable that you will need to turn in. The project will be graded on the quality of your work and will be weighted based on the difficulty of your project. The project is worth a total of 60 points.

Selecting an Algorithm. In terms of choosing an algorithm you have two options. Option 1 is to take a problem we’ve discussed (or will discuss) and explore a more efficient algorithm for the problem not covered in class. A list of algorithms we will cover during the remainder of the semester is provided below. Option 2 is to identify a problem that we haven’t discussed (or won’t discuss) and select a standard “naive” (straightforward, not as efficient) algorithm as well as a standard, efficient algorithm for solving the problem (note for both Option 1 and Option 2 you are required to do a comparison between a naive and the efficient algorithm you choose as part of the project). For Option 2, the problems should be relatively broad in scope (with clear applications) and have algorithms that require some sophistication (i.e., are non-trivial and/or more advanced). Other areas to consider for Option 2 include approximation algorithms for NP-complete problems (where there are numerous interesting problems and algorithms to study) as well as “randomized” versions of algorithms that often have good performance characteristics. In the remainder of the course, we will be looking at the problems of max-flow, finding cliques, finding shortest weighted paths, finding minimum spanning trees, finding optimal schedules, knapsack and bin packing, basic compression, finding independent sets, and string alignment (matching). Algorithms we plan to look at for the previous problems include: the Floyd-Fulkerson method and Edmonds-Karp algorithm (max-flows); Prim’s and Kruskal’s algorithms (minimum spanning trees); Bron-Kerbosch (cliques); the Dijkstra and Bellman-Ford algorithms (shortest weighted paths); Huffman Codes (compression); Floyd-Warshall (all-pairs shortest paths); simple greedy algorithms (scheduling); and simple dynamic programming approaches (independent sets, string alignment).

Implementation. As part of the project, you will be required to implement your selected algorithm (or algorithms for Option 2) from scratch, perform unit testing (similar to the homework assignments), and perform a performance evaluation (also similar to the homework assignments). Performance evaluations will compare the naive/simple algorithm against the efficient algorithm you have chosen. For the project you are required to do each of the following (in addition to implementing the algorithms):

- Develop a large and robust set of smaller examples to test your algorithm implementation on (i.e., a suite of unit tests);
- Develop a set of benchmarks, i.e., increasingly larger and/or more complicated examples, that can be used as part of a performance evaluation; and
• Implement and run a performance evaluation of both algorithms, i.e., the simple/naive algorithm implementation versus the efficient algorithm implementation. The performance evaluation will be run over your benchmark examples and will collect data that you will graph and compare similar to the homework assignments.

Note that for your project, you are free to use whatever programming language you like (python, java, C/C++, javascript, etc.).

Final Report. The last part of the project is to write a research-style paper that describes the problem the algorithms you selected solve, the specific algorithms you looked at and compared (including an analysis of their complexity), the benchmark you developed for performance testing, your performance evaluation results, and the resources you used as part of your research and implementation. In addition to the final report, you will also need to submit all source code to your project repository (instructions regarding the project repository will be provided via Piazza).

Step 1 (due by Thurs, 11/3): Algorithm Selection. The first step is to identify two algorithms you are interested in exploring. Note that each student in the class will need to work on a different algorithm. Assuming neither of your choices have already been selected, you will receive your first choice, followed by your second choice (if the first choice is taken), or else be notified by me to select an additional alternative. You must select algorithms that will not be discussed in class (see above) and that you are not already familiar with. Note that since you will be doing a comparison of the algorithm you select to a naive solution, for each of the two algorithms you propose, you must state what you plan to compare it to (the naive algorithm). Turn in a hardcopy print out of your algorithm selections in class by the due date along with a list of the resources you used as part of your research. This step is worth 2 points.

Step 2 (due by Thurs, 11/17): Resources, Language, and Initial Tests. The second step is to identify a list of authoritative sources that describe the algorithms you plan to develop. At least one of the sources must be primary (ideally, the paper where the algorithm was originally published; however, a published textbook will also suffice). Note that you should not rely on non-authoritative web sites (which often provide inaccurate, overly-brief, or narrowly focused descriptions of implementations). While Wikipedia would not be considered authoritative, it is often a good resource to get a general idea of an algorithm, but more importantly, often provides links to authoritative and primary sources. For this step, you must also select the language you plan to implement the algorithms, unit tests, and performance tests in. Finally, you should describe ten or so initial unit tests for the algorithms (these do not need to be coded, but just described via examples). Turn in a hardcopy print out of the above in class on the due date. This step is worth 2 points.

Step 3 (due by Thurs, 12/1): Status Update. By this point, you should at a minimum have completed the implementation of the algorithms, have most if not all unit tests complete, and have started developing the benchmarks for your performance evaluation. Turn in a hardcopy print out
that briefly describes your progress in class on the due date. This step is worth 2 points.

**Step 4 (due by Thurs, 12/15): Final Report and Submission.** Your final project submission is due on the day of the final exam. As stated above, your final submission will consist of your source code and your final report. Note that your work must be “repeatable” in that instructions (as needed) must be provided so that I can run your code and obtain similar results. A template will be provided for your final report. The final report (as a PDF file), source code, and instructions must be submitted to your project repository (as mentioned above). The remaining points will be assigned based on your source code and final report (again, focused on overall quality of work and difficulty of your project).