CS 499 Milestone Three – Enhancement Two: Algorithms and Data Structure

Student: Nurlan Gasimzade

**Narrative**

Briefly describe the artifact. What is it? When was it created?

The artifact is my CS300 Course Planner, a graph-based console program that ingests a list of courses and their prerequisite relationships, validates the graph, and outputs an ordered plan of study. It was originally created in CS300 and comprises: (1) a directed-graph data structure (adjacency list), (2) parsing/validation of input data, and (3) topological sorting to produce a valid course order. A companion set of data-structure exercises (hash table implementation with chaining and a binary search support routine) informed the approach to efficiency.

Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What components showcase skills and how was it improved?

I selected this artifact because it highlights algorithmic thinking and data-structure design beyond framework use. It showcases trade-offs among representations (adjacency list vs. matrix), algorithm selection (Kahn’s algorithm vs. DFS-based topological sort), and complexity analysis. For this milestone, I performed enhancements that make the artifact production-ready and pedagogically clear:  
• Refactored to a clean Graph API (`add\_vertex`, `add\_edge`, `in\_degree`, `neighbors`).  
• Implemented Kahn’s algorithm with an `O(V + E)` runtime, plus cycle detection and a precise error message when prerequisites form a cycle.  
• Added input schema checks, safe parsing, and helpful diagnostics for missing courses or self-loops.  
• Provided asymptotic analysis in code comments and a short README with example inputs/outputs.  
• Wrote unit tests for cycle detection, multiple valid orders, and disconnected subgraphs.  
• (Optional) Added a small hash table utility to map course codes to integer IDs for faster lookups and compact storage.

Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?

Yes. This enhancement directly supports the outcome of designing and evaluating computing solutions using algorithmic principles. It demonstrates well-founded techniques by using linear-time topological sorting, appropriate data structures, and clear trade-off discussion. It also advances professional communication by adding a README, test plan, and complexity notes that non-specialists can follow. Next, I plan to add a simple CLI flag to emit DOT/Graphviz so the plan and cycles can be visualized, and to package the code with a starter dataset.

Reflect on the process of enhancing and modifying the artifact. What did you learn? What challenges did you face?

Enhancing the Course Planner reinforced the value of isolating pure algorithmic logic from I/O. Writing unit tests around cycle detection made the corner cases visible—especially diamond-shaped graphs and courses that are listed as prerequisites but not defined. I learned that precise error messages are part of algorithmic engineering: they make the program usable for non-experts. Challenges included keeping the implementation both efficient and readable, and ensuring stable behavior when multiple valid topological orders exist. Documenting `O(V + E)` reasoning, plus adding a hash-based index for courses, helped keep performance predictable as inputs scale.

**What to Submit**

Submit a zip containing all original and enhanced code files (graph implementation, tests, example input files, README) and this narrative document. Ensure the README includes instructions to run tests and an example run that demonstrates cycle detection and a valid plan.