**Algorithms and Data Structures Narrative**

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

This artifact is an Employee Management System originally developed as a Binary Search Tree implementation for my Data Structures and Algorithms course about a year ago. The original version managed college course catalogs using basic BST operations for insertion, search, and in-order traversal. The enhanced version transforms this into a more advanced and professional-grade employee management system using a self-balancing AVL tree for guaranteed O(log n) performance.

***Justify the inclusion of the artifact in your ePortfolio. Why did you select this item?*** ***What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?***

I selected this artifact for three primary reasons. First, it was the most algorithmically rich project I completed during the program, directly involving core data structure operations. Second, it allowed me to demonstrate an advanced concept—the AVL tree—which adds meaningful complexity and depth beyond standard structures like vectors or hash maps. Finally, the project lends itself to real-world application: by transitioning from a course catalog to an employee directory, the system better reflects modern data management use cases in business or HR systems.

The key enhancements to this Employee Management program include:

* Conversion from a basic Binary Search Tree to a self-balancing AVL tree
* Implementation of proper memory management using a destructor, deep-copy constructor, and assignment operator
* Robust error handling and input validation for both file and user input
* Refactoring of previously monolithic code into clean, modular functions
* Enhanced CSV parsing to support quoted fields, whitespace trimming, and malformed data handling
* Addition of a comprehensive README file outlining usage, performance benchmarks, and technical documentation

***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?***

In my code review, I identified three core outcomes I planned to target with this enhancement:

* Design and evaluate computing solutions
  + I believe this was fully achieved. Replacing the original BST with an AVL tree required evaluating the performance limitations of unbalanced trees and designing a solution that maintained logarithmic performance under all insertion scenarios. The final implementation supports balanced tree growth with all four rotation cases (LL, RR, LR, RL), demonstrating algorithmic design, mathematical reasoning, and practical application.
* Demonstrate well-founded techniques in computing practices
  + This outcome was also met. In addition to implementing core algorithms, I reinforced proper C++ practices through memory-safe programming. This included adding a destructor, a deep-copy constructor, and a correctly defined assignment operator to prevent memory leaks and ensure correct tree behavior during copies and assignment.
* Deliver professional-quality communication
  + While I did not complete a formal test suite, I believe this outcome was still reasonably addressed through the polished modular function design, inline documentation, and the addition of a professional-grade README. The README outlines usage, system structure, time complexity, and file format expectations, making the project easy to understand and extend.

***Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?***

One of the most eye-opening parts of this enhancement was recognizing the dramatic difference in performance between balanced and unbalanced trees. Initially, I underestimated the importance of this distinction, especially since the original BST worked fine with small data sets. But once I started testing larger files and analyzed the theoretical behavior of worst-case insertions (e.g., sorted input leading to O(n) degradation), the value of AVL balancing became obvious.

The most difficult part of this project was implementing the AVL tree itself. AVL trees and tree rotations weren’t covered in my original coursework, so I had to research the underlying logic on my own. I spent a lot of time wrapping my head around how each of the four rotation cases worked, and how to correctly update node heights during recursive inserts. I also had to debug tricky issues related to imbalance detection and recursive pointer updates, which pushed me to better understand how trees behave under the hood.

This enhancement forced me to think like an engineer, not just a student. It wasn’t about getting the program to “work”—it was about making it scalable, safe, and maintainable. That mindset shift was probably the biggest gain from this process.