Algorithms and Data Structures Narrative

Artifact Description: The artifact used to demonstrate algorithms and data structures is an assignment to create a binary search tree from CS-300 Data Structures and Algorithms – Analysis and Design, completed in the summer of 2023. This project uses C++, a .csv file containing a list of bids and information on them, and a header file designed to parse the .csv file. The original program:

1. Parses through the .csv file of bids and adds each bid to a binary search tree.

2. Does an in-order traversal of the binary search tree and print out each bid in order of bid number.

3. Searches the binary search tree for a specific bid and prints out the bid.

4. Searches the binary search tree for a specific bid and removes it.

5. Exit the program.

Justify the inclusion of the artifact: The project was selected for inclusion in my ePortfolio because it represents a fundamental aspect of computer science and showcases my understanding and application of key concepts in data structures and algorithms. Initially, the BST implementation was straightforward but lacked efficiency and proper balancing. This led to performance issues, especially with large datasets. Ensuring a balanced tree is crucial for maintaining efficient search and delete operations, which is why this project was a perfect candidate to demonstrate these principles.

Improvements and Enhancements: To improve the artifact, I initially created a wrapper function to balance the tree after it was created. However, I realized that over time, with

continuous insertions and deletions, the tree would become unbalanced again. I did some research and came across Red – Black trees, B- trees, and a few others but I decided to implement an AVL tree, which is a self-balancing binary search tree, from the start. This involved:

1. Introducing rotation operations (left, right, left-right, and right-left) to maintain tree balance.
2. Modifying the insertion and deletion functions to ensure the tree remains balanced after each operation.
3. Using the AVL tree structure from the beginning to handle all insertions and deletions, ensuring the tree remains balanced at all times.

These enhancements significantly improved the efficiency of search and delete operations, ensuring they consistently perform at O(log n) time complexity. I wanted to test the runtime of the different operations using all 3 methods to make sure that I was correct in my pick of the AVL tree. I did change the menu slightly so that I could input a new bid and be able to specify a specific bud number for delete and search instead of having it hard coded.

One other enhancement I made was to pass a function object to the in order traversal. This way I can use it for anything that needs the bids in order, such as the print function or the add to vector function that I used in the first attempt.

```
void AVLBinarySearchTree::inOrder(Node* node, std::function<void(Bid)> func) {
    if (node != nullptr) {
        // Traverse left subtree
        inOrder(node->left, func);
        // Process node
        func(node->bid);
        // Traverse right subtree
        inOrder(node->right, func);
    }
}
```

Course Outcomes: By enhancing and modifying this artifact, I successfully met the course outcomes I had planned for. The second outcome - Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts – is being demonstrated with the code review and narrative provided with the artifact. The third outcome - Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices – is the main focus of this artifact. By researching first I was able to sort through all the different binary tree structures and find one that best worked for this particular problem with the AVL binary tree. The fourth outcome - Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry- specific goals – is accomplished because the improvements add value to the code by creating an application that runs faster.

Reflection on Process: At first, I had just wanted to create a balanced Binary Search Tree to reduce the time it would take to search for the bids in the tree structure. I employed this by creating a wrapper to change the initial tree into a balanced tree but felt that it didn't improve speed much except when searching and inserting and deleting unbalance the tree. Also, the load

time was ridiculous since I am creating a tree, doing a in order traversal to add the bids to a vector, and then building a balanced tree after. I looked for another method and found the AVL tree, named after its inventors, Georgy Adelson-Velsky and Evgenii Landis, that does everything that I wanted the tree to do (Jha, 2023). It has fast insertion, search, and deletion which are all O(log n) runtimes. It is a little bit complicated to implement but is far superior to the method used before. I did some testing on the speed differences and the AVL binary tree was the best around. The csv file contained a little over 12000 bids, and I based the speed measurements on clock ticks.

	Original	Balanced Original	AVL
Load Bids	10,584 – 10,638	10,876 – 10,971	3,144 – 3,197
Search Bid	5 - 6	0 - 1	0 – 1
Remove Bid	4 - 6	0 – 1	0 - 1
Insert Bid	3 - 4	0 - 1	0 – 1

The AVL outperformed substantially in all aspects compared to the original. While the balanced is better than the original in most aspects, it does not rebalance after inserts and deletes so the performance will decay over time until it is balanced again.

Jha, A. (2023, November 2). *AVL Tree Data Structure*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-avl-tree/