**Data Structures**

**Vectors:** In Lab 4-2, we were tasked with finding different ways to sort vectors. We were able to implement both a quick sort and a selection sort. Both methods were able to effectively sort our vector with fixed values. Vectors become a problem when adding or removing entries. It can be a very slow process as all other values will need to be adjusted when adding or removing a value. Since we did not implement a function to add or remove bids from our data set, a vector was the correct choice.

**Hash Tables:** For Lab 5-2 we implemented a hash table for our bids. Hash tables assign a value to a key that is created using a uniquely identifiable aspect of that value. An ID, for example. In our program, we wanted to functionality to add or remove bids. A hash table is suited better for this than a vector, because other items in the table will not need to be moved to ensure the cohesiveness of the data structure.

**Tree Structures:** Lab 6-2 allowed us to experiment with binary search trees (BST), which are very common data structures used in computer science. BSTs have the advantage of being very fast to search. For this program, we wanted to be able to find a bid using the bid ID, therefore a data structure that could be searched quickly would be best and is why we chose the BST.

**Algorithms**

**Search:** As mentioned previously, our BST in Lab 6-2 was by far the best performing when it comes to searching a data structure. The maximum times a BST will iterate is equal to the height of the tree. So a BST with 7 items, but with a height of 3 will be able to be searched and return the correct value with a maximum of 3 iterations.

**Sort:** The methods used in Lab 4-2 best display the functionality of common sorting methods used today. Our selection sort method splits the vector into two sections: sorted an unsorted. It does this by comparing two values and swapping them if they are not sorted. The quick sort method recursively creates partitions of values within our data structure, sorts the values within them and then returns until it reaches the top level of our recursion with a sorted set.

**Hash/Chaining:** Hashing and chaining was, of course, used in our hash tables in Lab 5-2. To create a hash key, our program used a simple algorithm to obtain the bid ID and then performed a modulus function on it to generate a key. Sometimes when using hash tables, a value may end up with the same key and the program will try and store two values with the same key. To combat this, we can use chaining. In our program if a value is found with the same key, the next unoccupied bucket is located, and our value is stored there instead.

**Student Choice**

Before this course, I had no idea what types of data structures were available. I was beyond impressed at the incredible amount of high-level conceptualization that must have happened to come up with everything we learned this term.

By far, my favorite data structure was the binary search tree. It’s a simple, easy to understand concept, and it outperforms every other data structure/algorithm pair we learned about.

The structure itself is very simple, containing only a node pointer and some methods to add, remove, or sort nodes. Inserting nodes consisted of comparing a bid’s ID to the root bid’s ID moving down the tree to the left or the right depending on if the ID is less than or greater than the root’s ID. This is done recursively with the nodes next in succession until an available location is found. A method to remove nodes traverses our tree in a similar way but reorganizes our tree if a node is removed that has a child. Searching our tree is indeed similar as we traverse our tree in the same way. When an ID is matched to one of our bids, that bid is returned.

This code is modularly composed. Every function of our program has its own function and its logic separated from the main class. This is done in a way that is congruent to human thinking as each of the defined methods serve as a bigger part of our BST class.

Much of this code is reusable as well. Because how modular each method is, to take this same logic and apply it in a completely different application with a list of different types of objects wouldn’t be difficult. Some variable names would need to be changed for clarity if working with a different data set than bids, but the code would function perfectly regardless.

In this lab, every step of the way has been annotated as to what its following code does and why. In development, there are many ways to achieve a result. Creating a BST can be done in several ways and the choices made in creating this one will help facilitate future developers that look at this program understand how it works. As an example, the BST class is defined in an organized way with comments explaining its function, and then each method within that class is defined and commented in the same organized fashion immediately afterwards.

**Conclusions**

Over the past several weeks in this course, I have learned many new concepts. Before this class, I had no concept of the different types of data structures used in computer science. Data structures are important because they assist in organizing and storing data. Without them, software development would be much harder, and it would be almost impossible to collaborate with other developers on a project. That said, it is apparent that there are different tools needed for each job. A data structure that works for a project may not be suitable for something else, and it’s important to keep that in mind when choosing a one to use. A data structure can be too slow for an implementation or could be too complicated to get the program to release quickly.

The same could be said about algorithms. There will be different times to use the algorithms learned during this course, but it should be noted that the data structure will determine the algorithm in many cases.

During this term, I found that many of these concepts will take some time to get a full grasp on. A lot of these data structures have moving parts and when writing code, those moving parts will need to be accounted for in the logic. In order to get more practice with these concepts, I will implement more of these data structures and algorithms into my personal projects and into future schoolwork. Currently, I am attempting to learn application development that catalogs data. It runs with arrays in Java, however I will be taking some of the concepts learned back with me to create a better performing application.