**Data Structure Analysis**

The Advisor’s requirements for ABCU’s program includes functionality to first, load courses into the data structure, and then, print those courses using two different formats. First, the user can search for one specific course. If that search is fruitful, then the program will print the specific course number, course title, and it will also print any prerequisite course information (course number and title) relevant to this specific course only. If the search is unproductive, then an error message will display, and the user is redirected to the main menu. Printing the entire list in alphanumeric order is the second available format. For this selection we utilize a sort algorithm on the loaded courses before we print the list.

The advantages and disadvantages of the three data structures in question are greatly influenced by the program requirements and corresponding algorithms. Although the vector is a linear data structure with fast and simple insertion and deletion functionality (O(1)), pointers require additional memory and time. Additionally, the vector search runtime increases as the number of elements increase O(n). Being that there are only eight courses currently, a vector would suffice. However, we must also consider scalability. In time, ABCU may offer nearly 100 courses or more. This would alter time comparisons.

Having a worst-case runtime of O(n2), yet a memory complexity of just O(log(n)), the quicksort algorithm is my preferred choice for a vector or when sorting the hash table (an associative array). It is noteworthy that a sort algorithm cannot be applied to a hash table directly. We must convert the values into an array, apply the sort algorithm, and then print the sorted list. This does not alter the hash table.

The hash table data structure stores key value pairs. This means that it has a unique key for each value. Like the vector, the hash table has an insert and delete runtime of O(1), but the hash table can also maintain the O(1) when searching for a value. It can be implemented using an array; it can be linear or non-linear, and it can store large amounts of data. One disadvantage, however, may be that hash functions can be complex to implement properly.

Our final data structure of the three in question, the binary search tree, can store an arbitrary number of values which is an advantage. It always has space for one more. However, the binary tree deletion algorithm is complex. This data structure is non-linear, and the insert and delete algorithms increase in run time as the height of the tree increases O(log(n)). In terms of memory complexity, it is O(n).

For the binary search tree, tree sort can be used as a one-time sort, but it is equivalent to [quicksort](https://en.wikipedia.org/wiki/Quicksort) as both recursively partition the elements based on a pivot. Tree sort can be a combination of inorder traversal, preorder and postorder functions, and it is the preferred sort method for a binary search tree data structure.

Space complexity is defined as the total memory space required by the function for its execution. The vector has a space or memory complexity of O(log(n)). The hash table produces an O(n) for memory. However, there is no true sort algorithm for the hash table or hash map, per se. When space complexity is considered, the binary search tree also has an O(n) result.