Code Evaluation

In order to analyze the run time of a program we must first find out how many lines of code are executed per function. We can also use a Big O value to show the overall speed of the function in use. For this assignment, we first must load courses into a data structure of our choosing. In my pseudocode for loading structures there are two for loops, which we can interpret as N, for number of loops. These can vary depending on the size of the list. Every other line of code is represented as 1 line. Therefore, the amount of runtime for the function to load courses into a data structure would be 3 + N + 1 + N + 6, or 2N +10. The Big O value ignores all constants; therefore, the Big O value is O(N). The three different data structures that I have been using thus far have been, vectors, binary search trees (BST), and hash tables. Each have their respective advantage and disadvantages, as well as their own runtime speeds. In terms of run time speeds, I ordered them from slowest to fastest. So the Big O values are O(N), O(log N), and O(1), respectively.

Starting off with vectors, these are easy to use and implement. The insertion of a new data node just requires the knowledge of the current last node and add it in after said node. The disadvantage is that the runtime for searching a specific node can take longer than the other two options. Since having to search for a specific node will require the program to iterate through the entire list, in order to find said node. This can take a longer time if the node we are looking for is at the very end of the list.

The next fastest data structure would BST’s. These are a little faster than a vector since sorting the courses would include either being greater than or less than the root node. The advantage to using a BST is of course speed, but it is still easy to implement and use, and it is increasingly faster than a vector sorting. The downside I see of using BST’s is that it can get difficult to sort and keep track of BST’s the bigger they get. Otherwise, searching through courses with a BST will be faster since we cut the time to search a course by Log.

The final data structure and fasted would be the hash table. A hash table biggest advantage is its speed with a Big O value of 1. The reason for it is that we give every course a hash key that will place that entry into a bucket. Therefore, when we go look for an entry all we need is the key and it will go to that bucket and retrieve said entry. It is increasingly faster than the other two since it minimizes iterating through the list and directly goes to node we want. The disadvantage to using a hash table is that it may be difficult to implement, and if your data size is large, it may be detrimental. If a large data set exists wither the buckets size must increase or methods like chaining, which include linked lists, must be used.

In conclusion, loading data, sorting data, and searching through data have many different functions that complete said job. Which ever one is used can be advantages or detrimental to use. If speed is what you want, difficult implementation may be required. For this project I would like to use a Hash table to load courses and search through courses. My only concern is with organizing through the courses in alphabetical order. I may use a bucket list of 29, room for each letter of the alphabet and chain any matching keys. Then organize those chains so that when I print out all courses, we will start with bucket full of ‘A’ courses in order than move down the chain list of courses that start with A. Then move on to each letter in the alphabet. I may still look into BST’s, since for me they are a little easier to implement, but I will try a Hash table first.