**Evaluations**

**Opening File, Closing File, and Tokens**

The opening, closing, and token acquisition would be pretty much the same for all data structures, with slight differences. Opening a file would execute once, so O(1), and parsing the file, reading lines, and splitting the lines all depend on the lines in the file, and that execution would be O(n). Overall, the operation for each data structure will be O(n).

**Vector Analysis**

Based on the advisor’s requirements, a vector would be at a disadvantage when searching for a course. Because in a worst-case scenario, you would have an unknown number of executions, and it could traverse the whole vector multiple times, giving us an O(n) runtime. Even searching would take a while, O(n), depending on the number of elements. But vectors are easy to access and work with in code.

**Hash Table Analysis**

A hash table would have an advantage over every other data structure when it came to storing and retrieving courses and their prerequisites on average. When inserting into a hash table, it’s an O(1) runtime. When searching through a hash table, it’s also an O(1) runtime. The only problem I see is the need to resize the table, which will consume extra memory.

**Binary Search Tree Analysis**

This process is more efficient than using a vector data structure, but less efficient than using a hash table. Inserting into the tree would take more than O(1) because it will have to go through loops until it finds its end position, O(log n), and at its worst, it would be O(n). Searching would also be O(log n) because it involves conditional loops, but every time the tree doubles in size, it only has to do one more step. Unless it’s a lopsided tree, and in that case, you’re looking at O(n).

**Recommendation**

My recommendation, which I plan to implement in my code, is the binary search tree data structure. This wasn’t what I had planned to use, and I had written out this portion entirely in favor of hash tables until I read the article you posted in the general discussion, which swayed me in favor of BSTs. As the article had stated, hash tables are faster than BSTs, but they use more memory than a BST data structure, and they don’t have a natural way to sort data (GeeksforGeeks, 2025). Even though we won’t be running these programs on our toasters, and memory won’t be much of a problem, I would like the ease of using recursives to sort the data. If I were dealing with a large amount of data and my application required real-time processing, such as a self-driving car, then I would use a hash table. The ease of sorting, the guaranteed O(log n) run time, and low memory usage make BSTs a strong recommendation.

|  |  |  |  |
| --- | --- | --- | --- |
|  | BST | Hash Table | Vectors |
| Search | O(log n) | O(1) | O(n) |
| Insertion | O(log n) | O(1) | O(n) |
| Deletion | O(log n) | O(1) | O(n) |
| Memory | Low | High | Low |

References

GeeksforGeeks. (2025, July 23). *Advantages of BST over Hash Table*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/advantages-of-bst-over-hash-table/