Christopher Diaz

04/08/2025

CS300

Project 1- Runtime analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Load data | Lookup course by ID number | Print all(sorted) | Memory Usage |
| Vector | O(n) | O(n) | O(n log n)\* | O(n) |
| Hash Table | O(n) | O(1) | O(n log n)\* | O(n) |
| BST | O(n log n) | O(log n) | O(n) | O(n) |

The program to be developed is a program that involves a list of college courses. For some of these courses, there can be prerequisite courses required before a student is allowed to attend the course. The program must be able to read the data involving these courses and their prerequisites courses so that the program can either output the data as a sorted list, or output 1 specific course with course information included. For this type of program, I feel the hash table is most suitable for a quick look up, making it best if the program involves frequent search ups even though it does require sorting which can take longer. Alternatively, the binary search tree is best for traversing through the list and printing it out in a sorted order. The lookup time isn’t as good as with a hash table, but it doesn’t fall far behind. As for the vector, it feels like the weakest and slowest approach because it takes the most time for search ups, and it requires the list to be sorted before it can be output. Overall, the vector is the simplest but also the least suitable to implement for this program simply because it does have simple implementation and low memory use, but it falls short on the time it takes to sort and print out an ordered list. It can be beneficial for a small dataset, and for memory saving, but having to read from a file makes the size of the dataset unpredictable. As for the hash table, printing through the entire sorted list after sorting it can be a massive downside if the dataset is large, making speeds slow, especially if it is done frequently. The hash table also uses more memory as another downside, but its major advantage is that it can quickly look up a course by the ID. In the BST, it is most efficient in sorting and traversing through a list even if it is the most complex. Additionally, it does have very quick search ups, but as a downside, it uses more memory than the other options. Ultimately, if I had to choose among these methods for the program, I would go with using a binary search tree because it is the most stable and safe option to use for this instance. It provides a balance of speed, stability, and structure for the program, making it the most assuring when it comes to the program running correctly, efficiently, and quickly.