Shimin Chan

CS 300

Prof. Kraya

**Project One**

Runtime Analysis:

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Cost per line | Number of executions | Total Cost  (Big O) |
| Open input file stream | 1 | 1 | O(1) |
| Read line by line using getline function | 1 | n (once per line) | O(n) |
| Parse each line | 1 (per part) | k parts per line x n lines | O(n x k) -> O(n) |
| Check formatting errors | 1 | n (once per line) | O(n) |
| Assign fields and store prerequisites in object | 1 (per part) | k parts per line x n lines | O(n x k) -> O(n) |

k is a constant  
n = the number of courses in the input file that the program will read and process

Overall complexity for all these processes, even using vector, hash table and BST, is O(n), linear with respect to the number of courses.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Insert/ Append course | Avg. Runtime Complexity | Worst Runtime Complexity |
| Vector | O(1) | O(n) | O(n) |
| Hash Table | O(1) | O(n) | O(n^2) |
| BST | Balanced BST – O(log n)  Worst case – O(n) | O(n log n) | O(n^2) |

Advantages and Disadvantages of each data structure:  
**Vector**  
A vector is simple to implement and allows fast insertion by just appending elements. It works well when you plan to sort all courses later. However, searching for a course by courseNumber requires a linear scan, making it O(n) in time complexity. Additionally, sorting the entire list for printing takes O(n log n) time.

**Hash Table**  
A hash table offers fast average-case search and insertion operations with O(1) time complexity when searching by courseNumber. However, printing all courses in order requires an extra step of gathering all the entries and sorting them, which takes O(n log n). Hash tables can suffer poor performance in worst-case scenarios if many collisions occur. Also, they are slightly more complex to implement compared to vectors.

**Binary Search Tree**  
A binary search tree maintains the courses in sorted order as you insert them, assuming the tree remains balanced. Searching by courseNumber is efficient on average, with O(log n) time complexity. Printing all courses in order is straightforward using an in-order traversal, which takes O(n) time. However, if the BST becomes unbalanced, search and insertion can degrade to O(n). While BSTs are more complex than vectors, they are somewhat simpler than hash tables. Adding balancing logic increases complexity but improves performance guarantees.

Recommendation for data structure to use:  
I believe that a Binary Search Tree (BST) is the most suitable data structure for the Computer Science program at ABCU. This is because it effectively balances the need for fast searching and sorted data retrieval. While the course dataset is not excessively large, it is substantial enough that using a linear search in a vector could become inefficient. The BST allows for average-case insertion and search operations in O(log n) time, which is significantly quicker than the O(n) time required for a search in a vector. Additionally, the BST naturally maintains the courses in sorted order, making it easy to print the course list in alphanumeric order without the need for extra sorting, as would be required with a hash table or vector. This combination of efficiency and order preservation makes the Binary Search Tree the optimal choice for effectively managing course data.