Big O Analysis

In my evaluation of the runtime and memory of data structures for course data, I made sure to consider the efficiency of operations, including file parsing, object creation, and data storage. The pseudocode outlines my approach on processing course data, involving tasks like opening files, reading lines, parsing data, and formatting. Each operations impacted runtime complexity and memory usage differently, depending on the chosen data structure. Generally, the process exhibits a runtime complexity of O(n).

In the pseudocode, courses are created by extracting relevant information from the given file. These objects hold course data including course numbers, titles, and prerequisites, which enables the user to have easy access and control of course information. Creating the course objects involves a constant number of operations per line of code, making efficient runtime complexity. However, memory usage for storing course objects depends on the size of the data structure and the attributes of each course object.

Analyzing the runtime complexity of each data structure highlighted advantages and disadvantages. Vectors offer efficient random access and dynamic resizing but may experience slower insertion and deletion operations, especially during resizing. Hash tables provide fast case insertion, deletion, and search but have increased runtime complexity in cases of collisions. In contrast, trees, like balanced binary search trees, are better at finding things quickly and keeping them organized. However, they need extra effort to make sure everything stays balanced, which can be less efficient for small data sets.

Considering these advantages and disadvantages a balanced binary search tree would be my recommended data structure for handling course data. Although more effort is put in initially creating the balanced tree compared to vectors or hash tables, its sorting capabilities and efficient search make it a better choice for organizing and accessing course information. The Big O analysis results show better runtime complexity for common operations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Structure | Function | Line Cost | Total Cost | Runtime |
| Vector | LoadCourses (parse and add data to vector) | 1 | n | O(n) |
|  | printSampleSchedule (print vector contents) | 1 | n | O(n) |
|  | printCourseInformation (search in vector) | 1 | n | O(n) |
| Hash Table | LoadCourses (parse and add data to hash table) | 1 | n | O(n) |
|  | printSampleSchedule (print hash table contents) | 1 | n | O(n) |
|  | printCourseInformation (search in hash table) | 1 | n | O(n) |
| Tree | LoadCourses (parse and add data to tree) | Log(n) | nlog(n) | O(log(n)) |
|  | printSampleSchedule (print tree contents) | Log(n) | nlog(n) | O(log(n)) |
|  | printCourseInformation (search in tree) | Log(n) | nlog(n) | O(log(n)) |