Felix Garcia

CS300

08/19/25

## Analysis

| Data Structure | Big O Runtime  (Loading) | Big O Runtime  (Search) | Big O Runtime  (Sort/Print) | Memory Usage |
| --- | --- | --- | --- | --- |
| Vector | *O(*𝑛*2)* | *O(*𝑛*)* | *O(*𝑛log𝑛*)* | *O(*𝑛*)* |
| Hash Table | *O(*𝑛*2)* | *O(1)*  *(average)* | *O(*𝑛log𝑛*)* | *O(*𝑛*)* |
| Binary Search Tree | *O(*𝑛*2)* | *O(*log𝑛*)*  *(balanced)* | *O(*𝑛*)* | *O(*𝑛*)* |

**Advantages and Disadvantages**

The vector data structure offers a straightforward approach for managing the course data, making it easy to implement with familiar library support. Its primary advantage is the simplicity of appending new courses, which is efficient at *O*(1), and it can sort the course list alphanumerically using built-in functions, meeting the requirement to print an ordered list. However, this comes with a significant drawback: searching for a specific course takes *O*(𝑛) time, which could slow down frequent lookups of individual course details, especially as the number of courses grows beyond the current 8. This inefficiency may frustrate advisors who need quick access to course information.

The hash table provides a robust solution with its constant-time average lookup capability (*O*(1)), allowing advisors to retrieve course titles and prerequisites almost instantly, which is ideal for frequent queries. Inserting courses is also efficient at *O*(1) on average, matching the vector’s loading speed. However, it lacks inherent ordering, requiring an additional O(log 𝑛) sorting step to print the alphanumeric course list, which could be a minor inconvenience. Despite this, its performance scales well with larger datasets, ensuring reliability as the curriculum expands.

The binary search tree excels in maintaining a natural alphanumeric order through in-order traversal, enabling an O(𝑛) print of the sorted course list without extra sorting, which aligns perfectly with the advisors’ need for an ordered display. Its search time of O(log 𝑛) in a balanced tree offers a middle ground between vector and hash table for individual course lookups. However, insertions take O(log 𝑛) per course, slowing down the initial loading compared to vector and hash table, and an unbalanced tree could degrade to O(𝑛) performance, adding complexity to ensure balance, which may complicate implementation.

**Recommendation**

Based on the runtime analysis and the advisors’ requirements, the hash table is the preferable choice out of the three. The loading time for all three structures is O(𝑛2) due to prerequisite validation, making them comparable in this phase for the current 8-course dataset. However, the hash table stands out with its *O*(1) average-case lookup time, enabling advisors to quickly access individual course titles and prerequisites, a task performed more frequently than showing all the courses. While the vector’s search and the BST’s search are slower, and the BST’s insertion adds overhead, the hash table’s efficiency in lookups outweighs the need for sorting the list alphabetically, which is less frequent. The hash table’s scalability and simplicity with standard libraries make it the best choice to meet the advisors’ needs effectively and reliably.