Evaluation and Analysis

Runtime breakdown by data structure:

| **Operation** | **Vector** | **Hash Table** | **Binary Search Tree** |
| --- | --- | --- | --- |
| Open file | O(1) | O(1) | O(1) |
| Read and parse file lines | O(n) | O(n) | O(n) |
| Validate prerequisites | O(n²) | O(n) (hash lookup) | O(n²) (list lookup) |
| Create course objects and insert | O(n) | O(n) | O(n log n) - average//O(n²) - worst case if unbalanced |
| **Total Worst-Case Runtime** | **O(n²)** | **O(n)** | **O(n²)** |

Advantages/Disadvantages by data structure:

Vector:

Advantages:

* Simple to implement
* Easy to sort and iterate throughout the structure

Disadvantages:

* Slow searching algorithms and validation
* Poor scalability

Hash Table

Advantages:

* Fast lookups (O(1) average)
* Great structure for checking prerequisites
* Great structure for accessing individual courses

Disadvantages:

* No natural order
* Requires extra sorting logic to print in order

Binary Search Tree

Advantages:

* Automatically stores data in sorted order(more or less depending on randomness of data loaded)
* In-order traversal naturally outputs sorted data for printing

Disadvantages:

* Performance worsens if tree is unbalanced
* Validation is still linear (O(n^2))

Recommendation:

Based on the runtime Big O analysis, the hash table is the best data structure for this application. It provides efficient loading and lookups with an O(n) worst case for total load. The hash table also provides very fast validation of prerequisites due to constant time lookups in the course index, as well as scalability for larger catalogs if necessary. The only downside to this structure is that it does not maintain order natively, however this was easily solved with the additional vector sorting steps included in the pseudocode above. In conclusion, the hash table offers the best balance between performance and simplicity for ABCU’s advising program. It ensures fast access to course data and scales well with increased data size.