**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Read each line from the file** | 1 | n | n |
| **Create course object** | 1 | n | n |
| **Insert into vector (push\_back)** | 1 | n | n |
| **Validate each course (search all others)** | 1 | n x n | n2 |
| **Total Cost** | | | n2 + 3n |
| **Runtime** | | | O(n2) |

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Read each line from the file** | 1 | n | n |
| **Create course object** | 1 | n | n |
| **Insert into hash table** | 1 | n | n |
| **Validate prerequisites (hash lookup)** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**BST**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Read each line from the file** | 1 | n | n |
| **Create course object** | 1 | n | n |
| **Insert into BST (O(log n) per insert)** | log n | 1 | n log n |
| **Validate prerequisites (log n lookup per prereq)** | log n | n | n log n |
| **Total Cost** | | | 2n log 2n + 1 |
| **Runtime** | | | O(n log n) |

**Pros and Cons of Data Structures**

One of the advantages of vector data structure is that programmers implement an efficient and simple system for storing data. It also allows for dynamic resizing for the inserted data. However, one disadvantage of a vector data structure is that it's inefficient in sorting data. The advisor needs a run of time that isn’t too slow for the purpose of transferring information quickly. Another disadvantage is vectors don’t allow for sorted data. This isn’t preferable to advisor because they are looking for a system to print out information in a structured order, which a vector isn’t capable of.

Now one pro about hash tables data structure offers quick lookup of information. Making it desirable in an on-demand environment requires information. It also allows for easy validation of prerequisites in the table. This is because all that needs to be checked is if the data exists within the table (O(1)). However, similar to a vector data structure, it’s inefficient and doesn’t organize data systematically. Another disadvantage of hash tables is that we’d be dealing with high usage of memory. Largely in part of the dataset potentially being large.

One of the advantages of the BST data structure is that it can systematically handle sorting data. Where travel of the tree is always handled in a structured order (i.e. inorder traversal). It also allows for resizing the inserted data, which can prove efficient as long as the tree remains balanced. However, one disadvantage of a BST is that in this case it's unbalanced; the BST can degrade into a linked list structure. It can’t also directly access elements in the structure... making it inherently slower.

**Recommendation**

Based on The Big O Analysis and analyzation on all three data structures, I recommend using a Binary Search Tree to print and store data in an alphanumeric list. Binary Search tree offers an algorithm that systematically handles sorting data. For instance, if we are trying to determine which course is printed first, CSCI100 and CSCI104, a binary search tree will compare which is smaller. Based on this algorithm, CSCI100 is less than CSCI104; therefore, CSCI100 will be inserted to the left most of CSCI104 (allowing it to be read first). BST structures also provide a solid middle-ground between fast and slow runtime. While a hash table is faster, it doesn’t sort its data conveniently. Lastly, BST structures. Therefore, BST structures assert themselves as the best data structure to solve the task.