**MENU:**

**WHILE (choice does not equal 9) {**

**Print Menu options;**

**Switch (choice)**

**Case 1:**

**loadCourses(csvPath, cours);**

**Print couses size;**

**Break;**

**Case 2:**

**printSortedCourses(courses)**

**Break;**

**Case 3:**

**printCourse(courses, courseNumber);**

**Break;**

**Case 9:**

**EXIT menu Print goodbye**

**END**

}

**printSortedCourses: (Courses)** {

SORT the courses in alphabetic order

FOR each course in sorted courses

PRINT Courses information

}

**Evaluation:**

**Vector:**

The advantages of using a vector for a data structure are that vectors are dynamic arrays, and most programming languages provide built-in support for vectors, making them simple to implement. Vectors are also great for ordered data, as well as for inserting and deleting data. They are also extremely efficient in accessing elements by index. However, using vectors also has its disadvantages, as linear search can be problematic in large data sets, making vectors not ideal for fast searches. Additionally, if you need to insert or delete elements in the middle, shifting elements will be required, leading to more time complexity.

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create vector** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create vector course item** | 1 | n | n |
| **While prerequisite exist** | 1 | n | n |
| **Append prerequisite** | 1 | n | n |
| **Push course item** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

**HashTable:**

The advantages of hash tables include their ability to facilitate rapid insertion, deletion, and lookup operations. This efficiency makes them particularly valuable for swiftly locating courses by course number. Additionally, hash tables are adept at managing large datasets, eliminating the necessity to sort data, as they utilize keys instead of a specific order. However, there are notable disadvantages: frequent collisions can significantly impair performance, and because hash tables rely on keys rather than ordering, there is inherently no arrangement among entries. Should there be a need to sort courses, extra steps will be required to implement sorting. Furthermore, hash tables may demand additional memory, especially when collisions occur and resizing becomes necessary.

| **HashTable** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create hash table** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create key for course** | 1 | n | n |
| **If no entry found for key** | 1 | n | n |
| **Assign node to key** | 1 | n | n |
| **While prerequisite exists** | 1 | n | n |
| **Append prerequisite** | 1 | n | n |
| **Insert course item** | 1 | n | n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |

**BinarySearchTree:**

The advantages of utilizing a binary search tree include the inherent organization of data in sorted order, which facilitates seamless printing and iteration through courses without the need for additional sorting. Furthermore, it offers efficient methods for searching, inserting, and deleting elements. However, one notable disadvantage is the potential for the binary tree to become unbalanced, which can result in suboptimal performance. In the worst-case scenario, its time complexity can be comparable to searching in vectors. Additionally, it may not be ideal for handling large datasets.

| **BinarySearchTree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Add node method** | 1 | 1 | 1 |
| **If root is null, add root** | 1 | 1 | 1 |
| **If node is less than root, add left** | 1 | n | n |
| **If no left node** | 1 | n | n |
| **If node is greater than root, add to right** | 1 | n | n |
| **If no right node** | 1 | n | n |
| **This node becomes right** | 1 | n | n |
| **For each line in file** | 1 | n | n |
| **Create course object** | 1 | n | n |
| **While prerequisite exists** | 1 | n | n |
| **Append prerequisite** | 1 | n | n |
| **Insert course item** | 1 | n | n |
| **Total Cost** | | | 11n + 2 |
| **Runtime** | | | O(n) |

**Recommendation:**

After analyzing all three data structures, Vectors, Hash Tables, and Binary Trees, I recommend using Hash Tables based on performance, memory, and implementation. The reason I recommend Hash Tables is that the primary requirement involves frequent lookups of course information by course number. Hash Tables are great for this, offering O(1) average time for lookups. Additionally, as the number of courses grows, the Hash Tables can scale while maintaining their great lookup speed. Hash Tables do use more memory due to collision handling; however, this is a good trade-off. Lastly, Hash Tables are widely supported, making them simple to implement. Given the advisors' requirements of fast lookups and a growing dataset, Hash Tables are the best fit.