**Big O Analysis :**

Vector

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cost of Line | Times Executed | Total Cost |
| Define Course | 1 | 1 | 1 |
| Define Variables | 1 | 1 | 1 |
| Open Course Information | 1 | 1 | 1 |
| Declare Vector | 1 | 1 | 1 |
| Split lines | 1 | n | n |
| Parse Courses | 1 | n | n |
| Create Courses | 1 | n | n |
| Store Courses | 1 | n | n |
| Close File | 1 | 1 | 1 |
| Total Cost: | | | 4n + 5 |

Hash

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cost of Line | Times Executed | Total Cost |
| Define Course | 1 | 1 | 1 |
| Define Variables | 1 | 1 | 1 |
| Open Course Information | 1 | 1 | 1 |
| Declare Hash | 1 | 1 | 1 |
| Split Lines | 1 | n | n |
| Assign First Parameter | 1 | n | n |
| Assign Second Parameter | 1 | n | n |
| Assign Remaining Parameters | 1 | n | n |
| Create Course | 1 | n | n |
| Set Course Number | 1 | n | n |
| Set Course Title | 1 | n | n |
| Set Prerequisites | 1 | n | n |
| Add Course | 1 | n | n |
| Close File | 1 | 1 | 1 |
| Total Cost: | | | 9n + 5 |

Binary Tree

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cost of Line | Times Executed | Total Cost |
| Define Course | 1 | 1 | 1 |
| Define Course Variables | 1 | 1 | 1 |
| Define TreeNode | 1 | 1 | 1 |
| Define TreeNode Variables | 1 | 1 | 1 |
| Define BST | 1 | 1 | 1 |
| Define BST Variables | 1 | 1 | 1 |
| Insert Function | 1 | 1 | 1 |
| Recursive Insert Function | 1 | 1 | 1 |
| Traverse Function | 1 | 1 | 1 |
| Recursive In Order Function | 1 | 1 | 1 |
| Declare BST Instance | 1 | 1 | 1 |
| Split Lines | 1 | n | n |
| Assign First Parameter | 1 | n | n |
| Assign Second Parameter | 1 | n | n |
| Assign Remaining Parameters | 1 | n | n |
| Create Course | 1 | n | n |
| Set Course Number | 1 | n | n |
| Set Course Title | 1 | n | n |
| Set Prerequisites | 1 | n | n |
| Call BST Insert | 1 | n | n |
| Close File | 1 | 1 | 1 |
| Total Cost: | | | 9n + 12 |

**Vector**

A vector is a simple and straightforward data structure that stores elements in a continuous block of memory. This makes it very fast for iterating through elements, but inserting or deleting elements, especially in the middle, can be slow because it may require shifting other elements. Vectors are easy to use and efficient for small or append-only datasets but have slower search times compared to other structures.

**Hash Table**

A hash table is designed for fast lookups, insertions, and deletions, often taking constant time on average. This makes them very efficient for search-heavy operations. However, hash tables are more complex to implement and manage, requiring a good hash function to minimize collisions, which can slow down performance. They also use more memory than vectors due to additional storage for pointers and empty slots.

**Binary Tree**

A binary tree stores data in a hierarchical structure, which keeps elements sorted and allows efficient searches. Operations like insertions, deletions, and lookups are typically fast, but the tree must be balanced to maintain this efficiency, which adds complexity. Binary trees are good for scenarios where data needs to be in sorted order, but they use more memory and are more complex to implement compared to vectors.

**Recommendation:**

For managing course data, a Hash Table is the recommended choice. It provides fast average-case performance for lookups, insertions, and deletions, which is essential for efficiently accessing course information. Despite its complexity and potential for collisions, the benefits of quick data retrieval outweigh these drawbacks, especially if a good hash function is used. Hash tables are particularly advantageous when dealing with large datasets where quick access is critical, making them well-suited for the task of managing and searching through numerous courses. If maintaining sorted order is a key requirement, consider using a Binary Tree for its efficient in-order traversal capabilities. However, for general use and optimal performance, the hash table is the best option.