|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **# Times Executives** | **Total Cost** |
| Create Vector | 1 | 1 | 1 |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Pushback course item | 1 | n | n |

|  |  |
| --- | --- |
| **Total Cost** | **5n+1** |
| ***Runtime*** | ***O(n)*** |

|  |  |  |  |
| --- | --- | --- | --- |
| **HashTable** | **Line Cost** | **#Times Executives** | **Total Cost** |
| Create hash table | 1 | 1 | 1 |
| Insert method | 0 | 0 | 0 |
| Create key for course | 1 | n | n |
| If no entry found for key | 1 | n | n |
| Assign node to key | 1 | n | n |
| Else | 1 | n | n |
| Assign old node key to UNIT\_MAX, set to key, set old node to course and old node next to null pointer | 4 | n | 4n |
| Else | 1 | n | n |
| Find the next open node | 1 | n | n |
| Add new newNode to end | 1 | n | n |
| For each new line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert course item | 1 | n | n |

|  |  |
| --- | --- |
| **Total Cost** | **16n+1** |
| ***Runtime*** | ***O(n)*** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tree** | **Line Cost** | **#Times Executes** | **Total Cost** |
| Add node method | 0 | 0 | 0 |
| If root is null, add root | 1 | 1 | 1 |
| If node is less than root then add to left | 1 | n | n |
| If no left node | 1 | n | n |
| This node becomes left | 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 vector course item | 1 | n | n |
| While prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert course item | 1 | n | n |

|  |  |
| --- | --- |
| **Total Cost** | **11n+2** |
| ***Runtime*** | ***O(n)*** |

**Analysis Explained:**

Each data structure has its own pros and cons based on the program's requirements. A drawback of using a vector is the need to search through the list for a specific course, which involves checking each item until a match is found. However, vectors are advantageous because they are the fastest method for reading files and adding course objects, making this approach straightforward. Among the three methods, vectors had the shortest runtime at Sn+1.

Hash tables excel in quickly searching lists by creating keys that allow easy searching and printing of locations. However, they are slower when initially creating the list and finding spots to insert each course. Additionally, hash tables don't allow for sorting the table itself; to print an alphanumeric list of courses, each value must be extracted, sorted, and then printed. For these reasons, hash tables may not be the best choice for this program.

Binary trees offer a sorting advantage over vectors due to their fast sorting capabilities. Although not as simple as hash tables, they are quicker than vectors. The search time for binary trees is O(h), where h is the height of the tree.

I recommend using a vector sort for this project. The ability to quickly sort and print the entire catalog is more valuable, and the time lost during searches is not as significant compared to the utility of sorting. In my opinion, vectors are the best option.