**Runtime Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create vector** | 1 | 1 | 1 |
| **WHILE prerequisite exists** | 1 | n | n |
| **Append prerequisite** | 1 | n | n |
| **Push back course item** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Hash Table | Line Cost | # Times Executes | 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 line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prerequisite exists | 1 | n | n |
| Append prerequisite | 1 | n | n |
| Insert course item | 1 | n | n |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Tree | Line Cost | # Times Executes | Total Cost |
| Create tree | 1 | 1 | 1 |
| 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 prerequisite exists | 1 | n | n |
| Append prerequisite | 1 | n | n |
| Insert course item | 1 | n | n |

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

Every data structure has its own advantages and disadvantages. The vector method is fastest for reading and adding course objects, but searching for a specific course means that it must check each item in the vector until the match is found. The runtime is shortest at 5n+1, with all of them having the same O(n) runtime.

Hash tables are able to search very quickly by creating a key, but has a much slower running time creating the initial hash table, and do not do well when sorting. It has the slowest total cost of 16n+1, and would not be recommended.

Binary trees are faster at searching than the vector, but not as fast as a hash table. If the tree only had left leaves, the search time would be O(h), with h being the height of the tree. It has a total cost of 11n+2.

Overall, the vector is the best way to set up this project. The quick sort and printing of the entire catalogue would be best for the client, and the search function does not have the worst total cost. I would recommend a vector for the client.