DECLARE var menuChoice

INITALIZE menuCoice == 0

CREATE orderNames

SORT courseName

PRINT

WHILE menuChoice != 9

PRINT menu

“Option 1: Load File Data

Option 2: Print all courses alphanumerically

Option 3: Print Course Titles and Prerequisites per course

Option 9: Exit “

Case 1:

loadCourses()

Case 2:

Print courseNum

orderNames

Case 3:

PROMPT courseNum

PRINT courseNum, courseName, prerequisite

Case 9:

EXIT Program

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **#Times Executed** | **Total Cost** |
| CREATE vector | 1 | 1 | 1 |
| FOR each line | 1 | N | N |
| Create course vector | 1 | N | N |
| WHILE prerequisite exists | 1 | N | N |
| ADD/append prerequisite | 1 | n | N |
| PUSH back course item | 1 | N | N |
| Total Cost | | | 5n + 1 |
| Runtime | | | O(n) |
| **HashTable** |  |  |  |
| create hashtable | 1 | 1 | 1 |
| Insert method | 0 | 0 | 0 |
| CREATE course key | 1 | n | n |
| For all courses | 2 | n | N |
| IF course = coursenumber | 1 | N | n |
| Print course info | 1 | 1 | 1 |
| FOR each course prerequisite | 2 | N | N |
| Print prerequisite info | 4 | n | N |
| Total Cost | 1n + 1 | | |
| Runtime | O(n) | | |
| **Tree** |  |  |  |
| ADD node | 0 | 0 | 0 |
| Add root if root = null | 1 | 1 | 1 |
| If node < root, add node to left | 1 | N | n |
| If no node | 1 | N | N |  |  |
| Node – left | 1 | N | N |  |  |
| If node > root, add node to right | 1 | N | N |  |  |
| If no node  Node = right | 1 | N | N |  |  |
| For each line in file | 1 | N | N |  |  |
| Create course item vector | 1 | N | N |  |  |
| While prerequisite = true | 1 | N | N |  |  |
| Append prerequisite | 1 | N | N |  |  |
| Insert courseitem | 1 | N | N |  |  |
| Total Cost | 11n +1 |  |  |  |  |
| Runtime | O(n) |  |  |  |  |

Each structure holds distinct advantages and disadvantages to this project. Using a vector is the fastest method of adding and reading files. Out of the three methods, vectors had the shortest runtime of 5n + 1. Vectors would also use less data. However, they would have to search for each item in the list. Additionally, vectors cannot delete data types or hold multiple data types. On the other hand, hash tables can quickly soften through data or a list. Hash tables allow for the creation of a key, allowing us to search for values easily. However, hash tables cannot be printed. Although you can insert and delete items, sorting the list would be difficult and inefficient. To do this, each item must be removed from the list, sorted, and printed. From this, I believe it is not the most effective path for the program. Binary search trees retrieve items in the order they are placed and can be deleted or added. However, the search tree must be balanced. Overall, I believe that a vector is the most efficient method given the project.