**EVALUATION**

**LINKED LIST**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for all courses | 1 | n | n |
| add course to vector | 1 | n | n |
| for each element of vector | 1 | n | n |
| add element and prerequisite to linked list | 1 | n | n |
| for all courses | 1 | n | n |
| if course is the same as courseNumber | 1 | n | n |
| print course information | 1 | 1 | 1 |
| for each prerequisite of course | 1 | n | n |
| print prerequisite course information | 1 | 1 | 1 |
| Total Cost | | | 8n + 2 |
| Runtime | | | O(n) |

**HASH TABLE**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for all courses | 1 | n | n |
| add course to vector | 1 | n | n |
| for each element of vector | 1 | n | n |
| add element and prerequisite to hash table | 1\* | n | n |
| for all courses | 1 | n | n |
| if key matches course key | 1 | n | n |
| print course information | 1 | 1 | 1 |
| for each prerequisite of course | 1 | n | n |
| print prerequisite course information | 1 | 1 | 1 |
| Total Cost | | | 8n + 2\* |
| Runtime | | | O(n)\* |

**BINARY TREE**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for all courses | 1 | n | n |
| add course to vector | 1 | n | n |
| for each element of vector | 1 | n | n |
| add element and prerequisite to linked list\* | 1\* | n | n |
| for all courses | 1 | n | n |
| if key matches course key | 1 | n | n |
| print course information | 1 | 1 | 1 |
| for each prerequisite of course | 1 | n | n |
| print prerequisite course information | 1 | 1 | 1 |
| Total Cost | | | 8n + 2\* |
| Runtime | | | O(n)\* |

All data structures (roughly) evaluate to O(n) runtime.

The linked list data structure is the most open to sorting and is very easily modified compared to other data structures. However, the data structure becomes more difficult to manage and sort the larger it becomes. Since this particular test only has it storing courses for a single major, it should be manageable.

The hash table structure is the easiest to add elements to at the cost of being almost entirely impossible to sort and modify. Inserting data also becomes more difficult and takes longer the larger the hash table becomes. Resizing would also come into question if the table were to get large enough.

The binary tree offers the greatest viability by being self-sorting, easy to insert and delete data from, and scalable to a sizable degree all the while maintaining a somewhat reasonable runtime. The larger the tree grows; the insertion or deletion of an internal node would become more difficult and more time consuming.

**RECOMMENDATION**

As such I would personally suggest that a binary tree be employed in storing this information. It runs the risk of slowing down the larger the data pool gets, but all the data structures bear that risk. It’s self-sorting nature and its search efficiency are too great boons to pass up for this data sorting need.