**Chris Bridges**

**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pseudocode** | **Line cost** | **times executed** | **Total cost** |
| Open file | 1 | 1 | O(1) |
| If file can not be opened, print error and return | 1 | 1 | O(1) |
| While the file has lines | 1 | n | O(n) |
| Read line | 1 | n | O(n) |
| Split line by , into items | 1 | n | O(n) |
| If item size < 2, print error and continue | 1 | n | O(n) |
| Create course object | 1 | n | O(n) |
| Assign course number | 1 | n | O(n) |
| Assign course name | 1 | n | O(n) |
| Assign prerequisites | 1 | n | O(n) |
| Append course to course vector | 1 | n | O(n2) |
| Close file | 1 | 1 | O(1) |

0(n) would be the total complexity of reading and course creation using a vector.

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pseudocode** | **Line cost** | **Times Executed** | **Total cost** |
| Open file | 1 | 1 | 0(1) |
| If file can not be opened print error and return | 1 | 1 | 0(1) |
| While file has lines remaining | 1 | n | O(n) |
| Read line | 1 | n | O(n) |
| Split line by , into items | 1 | n | O(n) |
| If size of item < 2, print error and continue | 1 | n | O(n) |
| Declare course as Course | 1 | n | O(n) |
| Assign course.number item[0] | 1 | n | O(n) |
| Assign course.name item[1] | 1 | n | O(n) |
| Assign course. Prerequisites item[2:1] | 1 | n | O(n) |
| Insert course into hash table | 0(n) | n | O(n) |

0(n) would be the complexity with O(n2) being the worst case scenario if it encountered numerous hash collisions.

**Binary Search Tree**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pseudocode** | **Line cost** | **Times Executed** | | **Total Cost** |
| Open file | 1 | | 1 | 0(1) |
| If file can not be opened, print error and return | 1 | | 1 | 0(1) |
| While file has lines remaining | 1 | | n | 0(n) |
| Read line | 1 | | n | 0(n) |
| Split line by , into items | 1 | | n | 0(n) |
| If size of items < 2 print error and continue | 1 | | n | 0(n) |
| Declare course as Course | 1 | | n | 0(n) |
| Assign course.number to item[0] | 1 | | n | 0(n) |
| Assign course.name to item[1] | 1 | | n | 0(n) |
| Assign course.prerequisite to item[2:] | 1 | | n | 0(n) |
| Insert course into BST | 1 | | n | 0( log n) |

0(n2) would be the worst case scenario for a binary search tree

Vectors

Vectors work well with smaller datasets and can be altered easily. They are easily sorted, modified, and are objectively the simplest of the 3 choices to implement simply because they’re just one large array. The problem with one large array comes with scaling. When arrays become large they take more time to search, call, and modify, making their choice for a dataset of this size to be possible but not ideal.

Hash Tables

Hash tables can be quicker than vectors because of their smaller array size. If a hash table is optimized to reduce the risk of collisions, it could run much more efficiently than vectors or binary search trees. This would slow the process form O(1) to O(n).

Binary search trees

Binary search trees work great at 0( log n) and can be a reliable way to store large data sets. If the tree becomes unbalanced operations can slow to O(n2). The implementation of binary search trees is a bit more complicated and more memory is required to establish the tree structure with pointers needed for each node.

Recommendation

I would recommend a hash table for this particular set of data. The courses offered by a school is relatively static with the need for alteration occurring only once or twice a year at most. The biggest operation of this program would be searching so hash tables would be an ideal choice due to the faster searching speeds. The size of the hash table would need to be adequate to prevent frequent collisions and improve the process, but it would be the optimal way to manage this data.