**6-2: Project One Analysis and Recommendation**

**Opening File Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| fstream open sampleFile | 1 | 1 | 1 |
| If (return values is negative) | 1 | n | n |
| Output error | 1 | 1 | 1 |
| Else { | 1 | n | n |
| While (not end of sampleFile) | 1 | n | n |
| Read sampleFile line by line | 1 | n | n |
| If (three or more parameters present) | 1 | n | n |
| If (three or more parameters elsewhere) | 1 | n | n |
| continue reading lines | 1 | n | n |
| Else { | 1 | n | n |
| Output error | 1 | 1 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Total Cost** | 8n + 3 |
|  |  | **Runtime** | O(n) |

**Vector Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Try | 1 | n | n |
| Create vector | 1 | 1 | 1 |
| For every line in sampleFile | 1 | n | n |
| Create vector for courseNum | 1 | n | n |
| Create vector for courseName | 1 | n | n |
| While (end of sampleFile not reached) | 1 | n | n |
| Create vector for Prereqs | 1 | n | n |
| Pushback course to end of vector | 1 | n | n |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Total Cost** | 7n + 1 |
|  |  | **Runtime** | O(n) |

**HashTable Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Hash create key for course | 1 | n | n |
| Retrieve node using key | 1 | 1 | 1 |
| If current equal to nullptr | 1 | n | n |
| Create new node | 1 | 1 | 1 |
| Insert new node at keys position | 1 | 1 | 1 |
| Else { | 1 | n | n |
| If (current key equal to UINT\_MAX) | 1 | n | n |
| Current key equals key | 1 | 1 | 1 |
| Current course equals aCourses | 1 | 1 | 1 |
| Next node equals nullptr | 1 | 1 | 1 |
| Else { | 1 | n | n |
| While (next node not equal to nullptr) | 1 | n | n |
| Current node equals next node | 1 | 1 | 1 |
| Insert new node to end | 1 | 1 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Total Cost** | 6n + 8 |
|  |  | **Runtime** | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Try | 1 | n | n |
| Create vector | 1 | 1 | 1 |
| For every line in sampleFile | 1 | n | n |
| Create vector for courseNum | 1 | n | n |
| Create vector for courseName | 1 | n | n |
| While (end of sampleFile not reached) | 1 | n | n |
| Create vector for Prereqs | 1 | n | n |
| hashTable->Insert course to end of vector | 1 | n | n |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Total Cost** | 7n + 1 |
|  |  | **Runtime** | O(n) |

**Tree Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| If (node is larger) | 1 | n | n |
| If (left node is nullptr) | 1 | n | n |
| New node becomes left | 1 | 1 | 1 |
| Else { | 1 | n | n |
| Recurse down left node | 1 | n | n |
| Else { | 1 | n | n |
| If (right node is nullptr) | 1 | n | n |
| New node becomes right | 1 | 1 | 1 |
| Else | 1 | n | n |
| Recuse down right node | 1 | n | n |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Total Cost** | 10n + 2 |
|  |  | **Runtime** | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Try | 1 | n | n |
| Create vector | 1 | 1 | 1 |
| For every line in sampleFile | 1 | n | n |
| Create vector for courseNum | 1 | n | n |
| Create vector for courseName | 1 | n | n |
| While (end of sampleFile not reached) | 1 | n | n |
| Create vector for Prereqs | 1 | n | n |
| bst->Insert course to end of vector | 1 | n | n |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Total Cost** | 7n + 1 |
|  |  | **Runtime** | O(n) |

**Pros and Cons**

**Vector Structure:**

* + **Pros:** It is relatively easy to add or remove the last element in the list. Elements within the list can be accessed by index to accomplish this. Each element is simply added to the end of the vector when parsing the file.
  + **Cons:** In order to search for a specific course each vector must be checked until a match is found. In order to remove elements from the head or middle of the list it requires the rest of the elements to be shifted to accommodate. Searching through larger vectors takes a considerably longer time as well6.

**Hash Table Structure:**

* + **Pros:** A hash table allows for direct access to its contents by quickly being searched by a hash key. The resizing of a table also allows for the rearrangement of elements to which helps decrease the search time.
  + **Cons:** Tends to take up a lot of unnecessary space when developing code. Data collisions could also occur which creates problems that would require removal techniques such as chaining.

**Binary Search Tree Structure:**

* + **Pros**: Data is stored in sorted order with the in-order traversal which allows for quick insertion and deletion of elements. In terms of range searches, it is relatively fast due to it not having to traverse a subtree that could not possibly have a certain element as well.
  + **Cons:** Searching for a specific element though takes longer though due to having to traverse an entire subtree. Trees must be balanced in order to perform efficient searches. The shape of the tree depends on the very first element that becomes the root.

**Data Structure Recommendation**

For this situation, I would advise the ABCU CS department to implement the binary search tree data structure to store, alter, and retrieve course data. The main justification is that this structure inherently has in-order sorting by default, which the application would benefit from being able to do so. This could be accomplished with hash tables and vectors, however sorting those data tables requires additional methods and functions that need to be called to. Trees also make it simple to insert a course if one must be added. I believe the longer runtimes of the subtrees are a downside that is easily manageable especially in a school environment.