Project One

|  |  |  |  |
| --- | --- | --- | --- |
| **Code for opening and reading file** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Open fileName with fileStream | 1 | 1 | 1 |
| If file is open | 1 | 1 | 1 |
| Set fileRead equal to fileName that is open | 1 | 1 | 1 |
| If fileRead is null | 1 | 1 | 1 |
| Print “Error: “ + error message | 1 | 1 | 1 |
| While not at end of file | 1 | n | n |
| Read file by parsing each line | 1 | n | n |
| Create vector courseInfo | 1 | 1 | 1 |
| Create string courseData | 1 | 1 | 1 |
| Getline from file into courseData | 1 | n | n |
| While courseData length is greater than 0 | 1 | n | n |
| If courseData includes at least two parameters and preReqs exist | 1 | n | n |
| Add courseData into vector courseInfo | 1 | n | n |
| Else print “Error: insufficient course data.” | 1 | 1 | 1 |
| Else print “Error opening file.” | 1 | 1 | 1 |
| Total Cost | | | 6n + 9 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector to create course objects** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Create vector | 1 | 1 | 1 |
| Create variables courseNum, name, prereqN | 1 | 1 | 1 |
| Open file to read | 1 | 1 | 1 |
| While not at end of file | 1 | n | n |
| Set variables from data in file | 1 | n | n |
| Store courseData in vector | 1 | n | n |
| Close file | 1 | 1 | 1 |
| Total Cost | | | 3n + 4 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash table to create course object** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Create variables courseNum, Name, prereqN | 1 | 1 | 1 |
| Create key by hashing course num | 1 | n | n |
| Retrieve node by using key | 1 | 1 | 1 |
| Set retrieved node to new node | 1 | 1 | 1 |
| Open file to read | 1 | 1 | 1 |
| While not at end of file | 1 | n | n |
| Set variables from data in file | 1 | n | n |
| Store data in vector | 1 | n | n |
| Close file | 1 | 1 | 1 |
| Total Cost | | | 4n + 5 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tree to create course object** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Create node to hold course info | 1 | 1 | 1 |
| IF tree is empty | 1 | 1 | 1 |
| Create new rooth with course info | 1 | 1 | 1 |
| Set left child to null | 1 | 1 | 1 |
| Set right child to null | 1 | 1 | 1 |
| Else if tree is not empty | 1 | 1 | 1 |
| Set current node to root | 1 | 1 | 1 |
| While current node is not null | 1 | n | n |
| If node is less than current node | 1 | n | n |
| If left child of current node is null | 1 | n | n |
| Set current nodes left child to node | 1 | 1 | 1 |
| Set current node to null | 1 | 1 | 1 |
| Else |  |  |  |
| Set current node to current nodes left child | 1 | 1 | 1 |
| Else |  |  |  |
| If right child of current node is empty | 1 | n | n |
| Set current nodes right child to node | 1 | 1 | 1 |
| Set current node to null | 1 | 1 | 1 |
| Else |  |  |  |
| Set current node to current nodes right child | 1 | 1 | 1 |
| Set nodes left and right child to null | 1 | 1 | 1 |
| Total Cost | | | 4n + 14 |
| Runtime | | | O(N) |

Advantages/Disadvantages:

Vector:

**Advantages:** Simple and easy for inexperienced programmers. It is very easy to add or remove the last element in the list. Elements can be accessed by index.

**Disadvantages:** List must be sorted properly in order to search effectively. Removing or inserting elements from the front or middle of the list is difficult because each following element must be shifted. Searching through large vectors takes a lot of time.

Hash Table:

**Advantages:** Direct access to items. Can quickly be searched by hash key. Resizing allows for rearranging of elements to decrease search time.

**Disadvantages:** Takes up a lot of unnecessary space. Collisions may occur which creates problems when searching for an element that shares a bucket with other elements.

Binary Search Tree:

**Advantages:** Allows for quick insertion and deletion of elements. Retrieving elements in increasing or decreasing order is very efficient.

**Disadvantages:** Searching takes longer than a hash table. Tree must be balanced in order to perform efficient searches. The shape of the tree depends on the very first element that becomes the root.

Recommendation

For this project, I would recommend using a binary search tree to store, modify, and retrieve course data. The main reason is because our application must be able to print the courses in alphabetical order, which a BST does automatically. Hash tables and vectors could do this, but extra code has to be written to sort those data tables. Trees also allow for easy insertion if a course needs to be added. Although is does take longer to search a BST rather than search through a hash table, I think it is worth it because a BST avoids any collisions and will have elements already sorted in order.