Jonathan Boeglin

CS-300 Project 1

Evaluation

| **Vector Code: vector fileLoad()** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **initialize ifstream file** | 1 | 1 | 1 |
| **open actual CSV file as file** | 1 | 1 | 1 |
| **initialize string line** | 1 | 1 | 1 |
| **TRY {** | 1 | 1 | 1 |
| **WHILE getline(file, line) {** | 2 | N | 2N |
| **initialize Course vector as courses** | 1 | N | N |
| **initialize integer i to 0** | 1 | N | N |
| **initialize a stringstream as inStr(line)** | 1 | N | N |
| **initialize cNum, cName, and preReqs as strings** | 1 | N | N |
| **getline(inStr, cNum = “ “, ‘,’)** | 1 | N | N |
| **getline(inStr, cName = “ “, ‘,’)** | 1 | N | N |
| **getline(inStr, preReqs = “ “)** | 1 | N | N |
| **IF cNum OR cName are: “ “ {** | 1 | 1 | 1 |
| **THROW (“Please use correct course format”)** | 1 | 1 | 1 |
| **initialize new structure Course as c with cNum, cName, and preReqs** | 1 | N | N |
| **add c to the end of vector courses** | 1 | N | N |
| **set line to “” as to ensure empty for next loop** | 1 | N | N |
| **close file** | 1 | 1 | 1 |
| **preReqCheck(courses)** | N | 1 | N |
| **return courses** | 1 | 1 | 1 |
| **CATCH (initialized string error) {** | 1 | 1 | 1 |
| **Print error** | 1 | 1 | 1 |
| **Total Cost** | | | 13N + 19 |
| **Runtime** | | | O(N) |

| **Hash Table Code: void fileLoad()** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **initialize ifstream file** | 1 | 1 | 1 |
| **open actual CSV file as file** | 1 | 1 | 1 |
| **initialize string line** | 1 | 1 | 1 |
| **initialize integers i and num as 0** | 1 | 1 | 1 |
| **initialize Course hashtable as courses** | 1 | 1 | 1 |
| **TRY {** | 1 | 1 | 1 |
| **WHILE getline(file, line) {** | 2 | N | 2N |
| **Set i equal to 0** | 1 | N | N |
| **Set num equal to 0** | 1 | N | N |
| **initialize a stringstream as inStr(line)** | 1 | N | N |
| **initialize cNum, cName, and preReqs as strings** | 1 | N | N |
| **getline(inStr, cNum = “ “, ‘,’)** | 1 | N | N |
| **getline(inStr, cName = “ “, ‘,’)** | 1 | N | N |
| **getline(inStr, preReqs = “ “)** | 1 | N | N |
| **IF cNum OR cName are: “ “ {** | 1 | 1 | 1 |
| **THROW (“Please use correct course format”)** | 1 | 1 | 1 |
| **FOR (i; i is less than cNum.length(); i++) {** | 2 | N | 2N |
| **Initialize char charval equal to cNum[i];** | 1 | N | N |
| **Set integer newnum equal to integer sum of (charval + 0)** | 1 | N | N |
| **Set num equal to num plus newnum** | 1 | N | N |
| **initialize new structure Course as c with cNum, cName, and preReqs** | 1 | N | N |
| **equate c’s key to num % 10** | 1 | N | N |
| **HashInsert(courses, c)** | N | N | N^2 |
| **preReqCheck(courses, c.preReqs)** | N | N | N^2 |
| **set line to “” as to ensure empty for next loop** | 1 | N | N |
| **close file** | 1 | 1 | 1 |
| **return** | 1 | 1 | 1 |
| **CATCH (initialized string error) {** | 1 | 1 | 1 |
| **Print error** | 1 | 1 | 1 |
| **Total Cost** | | | 2N^2 + 17N + 12 |
| **Runtime** | | | O(N^2) |

| **BST Code: void fileLoad()** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| initialize ifstream file | 1 | 1 | 1 |
| open actual CSV file as file | 1 | 1 | 1 |
| initialize string line | 1 | 1 | 1 |
| initialize integers i and num as 0 | 1 | 1 | 1 |
| initialize Course BinarySearchTable\* as bst | 1 | 1 | 1 |
| TRY { | 1 | 1 | 1 |
| WHILE getline(file, line) { | 2 | N | 2N |
| Set i equal to 0 | 1 | N | N |
| Set num equal to 0 | 1 | N | N |
| initialize a stringstream as inStr(line) | 1 | N | N |
| initialize cNum, cName, and preReqs as strings | 1 | N | N |
| getline(inStr, cNum = “ “, ‘,’) | 1 | N | N |
| getline(inStr, cName = “ “, ‘,’) | 1 | N | N |
| getline(inStr, preReqs = “ “) | 1 | N | N |
| IF cNum OR cName are: “ “ { | 1 | N | N |
| THROW (“Please use correct course format”) | 1 | 1 | 1 |
| FOR (i; i is less than cNum.length(); i++) { | 2N | N | 2N^2 |
| Initialize char charval equal to cNum[i]; | 1 | N | N |
| Set integer newnum equal to integer sum of (charval + 0); | 1 | N | N |
| Set num equal to num plus newnum; | 1 | N | N |
| initialize new structure Course as c with cNum, cName, and preReqs | 1 | N | N |
| preReqCheck (preReqs) | N | N | N^2 |
| bst->BSTInsert(c) | 1 | N | N |
| set line to “” as to ensure empty for next loop | 1 | N | N |
| close file | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| CATCH (initialized string error) { | 1 | 1 | 1 |
| Print error | 1 | 1 | 1 |
| **Total Cost** | | | 3N^2 + 16N + 11 |
| **Runtime** | | | O(N^2) |

**Analysis:**

The advantage of a simple vector is the ease of loading file data and creating objects from that data. It’s disadvantages show when actually searching for specific data, or looking to print the data in a specific way. For Hash Tables, increased speed of searching and a set system for finding data with a key makes searching a bit easier, however loading data can take longer. Also, Hash Table keys can cause different objects to be stored in the same slot, making it a longer process of finding an individual object if its key matches another. Finally, Binary Search Trees are the quickest at searching themselves for data, and each node is unique and does not share like with Hash Tables. However, they also take longer to transfer data to, and are harder to remove data from in certain instances.

**Recommendation:**

With these advantages and disadvantages in mind, and the results of the runtime breakdown, I am recommending that a Binary Search Tree be implemented for the code. This is because while according to the runtime evaluation it is slower to load a file onto, this file upload should happen only once, compared to the rest of the program, which will be utilized multiple times. For example, while I may upload my course list once, I will be accessing it from the program multiple times in order to add, remove, print in order, and print individual courses. These actions, compared to uploading, are much faster with a Binary Search Tree, and the data is stored in such a way that it will not overlap, unlike a Hash Table.