Nick Apodaca

CS 300

Runtime Analysis Summary from Project One

**Runtime Analysis Vector:**

Open File:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print statement confirming opening of file** | 1 | 1 | 1 |
| **Initialize parser** | 1 | 1 | 1 |
| **If statement when file cannot be opened** | 1 | 1 | 1 |
| **Print error message for when file can’t be opened** | 1 | N | n |
| **Return statement** | 1 | 1 | 1 |
| **While loop** | 1 | N | n |
| **Read each line of code** | 1 | N | N |
| **Call processLine** | 1 | N | 9 |
| **Close file when it reaches the end** | 1 | 1 | 1 |
| **Print file close message** | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 14 |
| **Runtime** | | | O(n) |

Process Line:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define course data** | 1 | 1 | 1 |
| **IF catch for incomplete data** | 1 | 1 | 1 |
| **Error message for if statement** | 1 | 1 | 1 |
| **Define courseNumber** | 1 | 1 | 1 |
| **Define courseName** | 1 | 1 | 1 |
| **Define prereqs** | 1 | 1 | 1 |
| **Call createCourse** | 1 | 1 | 2 |
| **Total Cost** | | | 8 |
| **Runtime** | | | O(n) |

Create Course:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define course as new course** | 1 | 1 | 1 |
| **Add course to vector** | 1 | 1 | 1 |
| **Total Cost** | | | 2 |
| **Runtime** | | | O(n) |

**Runtime Analysis HashTable:**

Open File:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print statement confirming opening of file** | 1 | 1 | 1 |
| **Initialize parser** | 1 | 1 | 1 |
| **If statement when file cannot be opened** | 1 | 1 | 1 |
| **Print error message for when file can’t be opened** | 1 | N | n |
| **Return statement** | 1 | 1 | 1 |
| **While loop** | 1 | N | n |
| **Read each line of code** | 1 | N | N |
| **Call processLine** | 1 | N | 16 |
| **Close file when it reaches the end** | 1 | 1 | 1 |
| **Print file close message** | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 22 |
| **Runtime** | | | O(n) |

Process Line:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define course data** | 1 | 1 | 1 |
| **IF catch for incomplete data** | 1 | 1 | 1 |
| **Error message for if statement** | 1 | 1 | 1 |
| **Define courseNumber** | 1 | 1 | 1 |
| **Define courseName** | 1 | 1 | 1 |
| **Define prereqs** | 1 | 1 | 1 |
| **Call createCourse** | 1 | 1 | 10 |
| **Total Cost** | | | 16 |
| **Runtime** | | | O(n) |

Create Course:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define course as new course** | 1 | 1 | 1 |
| **Insert course into hashtable** | 1 | 1 | 9 |
| **Total Cost** | | | 10 |
| **Runtime** | | | O(n) |

**Runtime Analysis BinaryTree:**

Open File:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print statement confirming opening of file** | 1 | 1 | 1 |
| **Initialize parser** | 1 | 1 | 1 |
| **If statement when file cannot be opened** | 1 | 1 | 1 |
| **Print error message for when file can’t be opened** | 1 | N | n |
| **Return statement** | 1 | 1 | 1 |
| **While loop** | 1 | N | n |
| **Read each line of code** | 1 | N | N |
| **Call processLine** | 1 | N | 24 |
| **Close file when it reaches the end** | 1 | 1 | 1 |
| **Print file close message** | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 30 |
| **Runtime** | | | O(n) |

Process Line:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define course data** | 1 | 1 | 1 |
| **IF catch for incomplete data** | 1 | 1 | 1 |
| **Error message for if statement** | 1 | 1 | 1 |
| **Define courseNumber** | 1 | 1 | 1 |
| **Define courseName** | 1 | 1 | 1 |
| **Define prereqs** | 1 | 1 | 1 |
| **Call createCourse** | 1 | 1 | 18 |
| **Total Cost** | | | 24 |
| **Runtime** | | | O(n) |

Create Course:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define course as new course** | 1 | 1 | 1 |
| **Insert course into tree** | 1 | 1 | 17 |
| **Total Cost** | | | 18 |
| **Runtime** | | | O(log n) |

**Evaluation:**

For this program, two things are important. The programs ability to quickly sort and present the list of course and its ability to search through the list of courses for the one who’s details are needed. For this program we considered three possible types of data structures, vectors, hash tables, and binary search trees.

Vectors are the most straightforward. It is useful when the size of the structure is inconsistent. From our runtime, we see that it has the shortest runtime cost. That being said, adding additional elements can come at a cost since it would require shifting elements. Also it is not the most effective when it comes to a sorted search.

A hash table stores unordered items to an array with a hash key. This method of storing with keys means that it is an ideal data structure for searches, enabling a low cost for searches. It is susceptible to hash collisions though when multiple keys map to the same index. In addition, the elements are unordered which means extra lengths must be taken if the data needs to be structured in ordered fashion like this program’s requirements have.

The last structure is the binary search tree. The structure excels are maintain elements in a sorted order. It also is proficient at searches because it uses a logarithmic complexity. It has it’s disadvantages though in that as our runtime shows, it has the highest cost of operation. It has more memory usages than the other structures. It also is more complex requiring more effort in creating.

Even though in our runtime analysis it had the largest cost, because it sorts data in an ordered fashion and because it is efficient in searching through data, it is our recommendation that a binary search tree be used to handle the course data. We believe that it will provide best experience for the user.