Run Time Analysis

Matthew Courts

8/6/2023

With the current known usage of the software the recommendation is to use a simple vector to store the list of courses. With the current state of the software the advisors are using it to view a list of courses in the different orders. The list is not being added to with the program or removed. With this a flat list provided by a vector will be less resource intensive for altering to the needs of the user. The benefit to this there are many built in functions available for vectors that will reduce overhead of additional functionality leaving an average time complexity of O(N). Other data types will provide a better time complexity but will require extra overhead to create a sort-able list.

| **Function** | **Total Lines** | **Complexity** | **Total Cost** |
| --- | --- | --- | --- |
| Append | 1 | 1 | O(1) |
| Prepend | 14 | LogN | O(14LogN+2) |
| Search | 6 | LogN | O(6LogN+1) |
| Remove | 14 | LogN | O(14LogN+3) |
| Deconstruct | 1 | 1 | O(1) |

With the use of a hash table the time complexity of each search will be O(1). This will be extremely fast but also require the addition of O(N^2) in the case of creating a flat list in order for printing the table. The noted use for the software in the current state is viewing the class list. Classes would be added via the CSV and outside the scope of the current use case. Adding and removing courses would give reason to use hash table as this will give fast access to the record of the course. The management through the use of hashes as keys to a bucket allows for direct access to the bucket containing the data. The downside to Hash table are the size of the allocated resources for the table may be taken up by the software even though they are not used. A Hash Table should be scaled in size relative to the data stored, as a filled hash table may no longer be able to store. Too many similar keyed courses may also excessively fill a bucket leading to longer search times.

| **Function** | **Total Lines** | **Complexity** | **Total Cost** |
| --- | --- | --- | --- |
| Constructor | 1 | 1 | O(1) |
| insert | 14 | LogN | O(14LogN+2) |
| Remove | 15 | LogN | O(15LogN+3) |
| Search | 6 | LogN | O(6LogN+1) |
| Deconstruct | 1 | 1 | O(1) |

A binary search tree can a solution to having too many keyed courses in a single bucket by creating a left-right fork path for finding data. This leads to an average O(LogN) complexity searching for data. However a Binary tree may also become unbalances leading to the need of changing the root node to a better middle ground. This can be intensive depending on the size of the tree.

| **Function** | **Total Lines** | **Complexity** | **Total Cost** |
| --- | --- | --- | --- |
| Constructor | 1 | 1 | O(1) |
| AddNode | 7 | LogN | O(LogN+3) |
| RemoveNode | 25 | LogN | O(25LogN+6) |
| Search | 9 | LogN | O(9LogN+2) |
| Deconstruct | 1 | 1 | O(1) |