# Run Time Tables

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create Vector (DEFINE empty list courses)** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create vector course item (CREATE new Course)** | 1 | n | 1 |
| **While prereq exists (IF more than 2 tokens)** | 1 | n | n |
| **Append prereq (ADD remaining tokens)** | 1 | n | n |
| **Pushback course item (ADD Course to courses list)** |  |  |  |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create hash table (implied initialization)** | 1 | 1 | 1 |
| **Insert method (grouping for hash table ops)** | 0 | 0 | 0 |
| **Create key for course (courseNumber = parts[0])** | 1 | n | n |
| **If no entry found for key (containsKey check)** | 1 | n | n |
| **Assign node to key (coursesTable.put)** | 1 | n | n |
| **Else (duplicate check branch)** | 1 | n | n |
| **Assign old node key to UNIT\_MAX, etc. (simplified as print error)** | 1 | n | n |
| **For each new line in file (While loop)** | 1 | n | n |
| **Create vector course item (course = new Course)** | 1 | n | n |
| **While prereq exists (if parts.length > 2)** | 1 | n | n |
| **Append prereq (prerequisites.append)** | 1 | n | n |
| **Insert course item (createCourse call)** | 1 | n | n |
| **Total Cost** | | | 11n + 1 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary Seach Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Add node method (grouping for BST ops)** | 0 | 0 | 0 |
| **If root is null, add root (root = new Node)** | 1 | 1 | 1 |
| **If node is less than root then add to left (courseNumber < current)** | 1 | n | 1 |
| **If no left node (current.left is empty)** | 1 | n | n |
| **This node becomes left (current.left = new Node)** | 1 | n | n |
| **If node is greater than root add to right (courseNumber > current)** | 1 | n | n |
| **If no right node (current.right is empty)** | 1 | n | n |
| **This node becomes right (current.right = new Node)** | 1 | n | n |
| **For each line in file (while loop)** | 1 | n | n |
| **Create vector course item (course = new Course)** | 1 | n | n |
| **While prereq exists (for i = 2 to parts.length)** | 1 | n | n |
| **Append prereq (prerequisites.append)** | 1 | n | n |
| **Insert course item (coursesTree.insert)** | 1 | n | n |
| **Total Cost** | | | 11n + 1 |
| **Runtime** | | | O(n) |

# Evaluation

**Based on the advisor’s requirements, analyze each of the vector, hash table, and tree data structures. Explain the advantages and disadvantages of each structure in your evaluation.**

Each data structure has its own strengths and weaknesses. When it comes to the Vector data structure, it can be fairly easy to use and especially fast if you know exactly where a course is, however if you don't it can then become slow and it can also become more difficult to add or remove courses. For Hash Tables, their strengths lie in their ability to find a course by its number, this can make it quicker to add or remove certain courses. Its weakness is that the courses are not in any order, so it takes more time to sort them out. Lastly the Binary Search Tree method has its strengths in keeping the courses in order and can quickly find, add, or remove any certain course. However, the system can be a lot slower if it is not balanced properly and it can also be more difficult to set up than a vector or hash table.

**Now that you have analyzed all three data structures, make a recommendation for which data structure you plan to use in your code. Provide justification for your recommendation based on the Big O analysis results and your analysis of the three data structures.**

Based upon the analysis of all three different data structures, I would lean more towards recommending a Binary Search Tree method. I chose this method for a few reasons. The first reason is the ease of being able to quickly find, add, or remove specific courses. This is a benefit that the vector method does not have. While the system is slower than a vector, it is more efficient and is able to perform this in the same runtime as the hash table with the added benefit of being ordered. The only downside would be the complexity of the system.