**Runtime Analysis: Vector – Milestone 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times**  **Executes** | **Total**  **Cost** |
| **Open file in reading mode with name = filename** | **1** | **1** | **1** |
| **First if statement: If file cannot be opened, print error and exit** | **1** | **1** | **1** |
| **Create empty list called courseList** | **1** | **1** | **1** |
| **Create empty set called allCourseIds** | **1** | **1** | **1** |
| **First for loop: For each line in file** | **1** | **n** | **n** |
| **Split the line into parts using delimiter** | **1** | **n** | **n** |
| **if number of parts < 2: print error** | **1** | **<=n** |  |
| **courseID = parts[0]** | **1** | **n** | **n** |
| **title = parts[1]** | **1** | **n** | **n** |
| **Add courseID to allCourseIds** | **1** | **n** | **n** |
| **End for loop** | **1** | **1** | **1** |
| **Create empty list called referenceList** | **1** | **1** | **1** |
| **Create a variable called prereqId** | **1** | **1** | **1** |
| **For each line in file:** | **1** | **n** | **n** |
| **Split line into parts using delimiter** | **1** | **n** | **n** |
| **courseID = parts[0]** | **1** | **n** | **n** |
| **title = parts[1]** | **1** | **n** | **n** |
| **For each part from index [2] to end:** | **1** | **n** | **n** |
| **prereqId = parts[2]** | **1** | **n** | **n** |
| **If prereqId is not in allCourseIds:** | **1** | **<=n** |  |
| **Print "Invalid Prerequisite.”** | **1** | **<=n** |  |
| **Else Add prereqId to the referenceList** | **1** | **<=n** |  |
| **End if statement** | **1** | **1** | **1** |
| **End nested for loop** | **1** | **1** | **1** |
| **Create course Object** | **1** | **1** | **1** |
| **Add courseID, title, referenceList to course Object** | **1** | **n** | **n** |
| **Add course Object to courseList** | **1** | **n** | **n** |
| **End for loop** | **1** | **1** | **1** |
| **Close the file** | **1** | **1** | **1** |
| **Return courseList** | **1** | **1** | **1** |
| **Total Cost** | | | **13** |
| **Runtime** | | | **13** |

**Runtime Analysis: Hash Table – Milestone 2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Open file in reading mode with name = filename** | **1** | **1** | **1** |
| **First if statement: If file can’t be opened, print message and exit** | **1** | **1** | **1** |
| **Create an empty set called allCourseIds** | **1** | **1** | **1** |
| **Construct a hash table** | **1** | **1** | **1** |
| **HashTable(size\_t size : table\_size(size), table(size)** | **1** | **1** | **1** |
| **Function hashFunction(int key): Return key** | **n** | **n** | **n** |
| **Call the hashFunction to compute indexes for keys** | **1** | **1** | **1** |
| **int index = hashFunction(key)** | **1** | **1** | **1** |
| **Insert a key-text pair into the hash table at an indexed location** | **1** | **1** | **1** |
| **table[index].push\_back(make\_pair(key, text))** | **1** | **1** | **1** |
| **For each line in file:** | **1** | **n** | **n** |
| **Split the line into parts using delimiter** | **1** | **n** | **n** |
| **if number of parts < 2: print error** | **1** | **<=n** |  |
| **courseID = parts[0]** | **1** | **n** | **n** |
| **title = parts[1]** | **1** | **n** | **n** |
| **Add courseID to allCourseIds** | **1** | **n** | **n** |
| **Call the Insert function with courseID and title as the argument** | **1** | **n** | **n** |
| **Insert(courseID, title)** | **1** | **n** | **n** |
| **End for loop** | **1** | **1** | **1** |
| **For each line in file: Split using delimiter, set courseID = parts[0] and title = parts[1]** | **1** | **n** | **n** |
| **Create a variable called prereqId** | **1** | **1** | **1** |
| **For each part from index [2] to end:** | **1** | **n** | **n** |
| **If line has a third element determined by the delimiter, set prereqId = parts[2]** | **1** | **n** | **n** |
| **Else if line has third and fourth element determined by the delimeter PrereqId = parts[2] prereqId = parts[3]** | **1** | **n** | **n** |
| **Else prereqId = “ “** | **1** | **n** | **n** |
| **End if elif statement** | **1** | **1** | **1** |
| **If prereqId is not in allCourseIds: Print "Invalid Prerequisite.”** | **1** | **<=n** |  |
| **prereqId = “ “** | **1** | **<=n** |  |
| **End if statement** | **1** | **1** | **1** |
| **End nested for loop** | **1** | **1** | **1** |
| **Call the Insert function; Insert(courseID, title.append(prereqId))** | **1** | **n** | **n** |
| **End for loop** | **1** | **1** | **1** |
| **Close the file and Return HashTable** | **1** | **1** | **1** |
| **Total Cost** | | | **16** |
| **Runtime** | | | **16** |

**Runtime Analysis: Binary Search Tree – Milestone 3**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Open file in reading mode** | **1** | **1** | **1** |
| **If file can’t open: print error message and exit program** | **1** | **1** | **1** |
| **Create a node called Binary\_Search\_Tree** | **1** | **1** | **1** |
| **Create an empty set called allCourseIds** | **1** | **1** | **1** |
| **If the file has >= 2 parameters: Print “File is formatted correctly”** | **1** | **1** | **1** |
| **Else Print “File is incorrect”** | **1** | **1** | **1** |
| **End if statement** | **1** | **1** | **1** |
| **For each line in file:** | **1** | **n** | **n** |
| **Split the line into parts with a delimiter** | **1** | **n** | **n** |
| **The element before the first delimiter is index[0]** | **1** | **n** | **n** |
| **The element before the second delimiter is index[1]** | **1** | **n** | **n** |
| **Each element has an index** | **1** | **n** | **n** |
| **courseID = index[0]; Add courseID into allCourseIds** | **1** | **n** | **n** |
| **courseName = index[1]** | **1** | **n** | **n** |
| **If index[2] is not within allCourseIds: print “Invalid Prerequisite”** | **1** | **n** | **n** |
| **Else preReqCourse = index[2]** | **1** | **n** | **n** |
| **End if statement** | **1** | **n** | **n** |
| **Create a new object call Obj** | **1** | **n** | **n** |
| **Add courseID, courseName, and preReqCourse (if any) into Obj** | **1** | **n** | **n** |
| **Add Obj into the next node Binary\_Seach\_Tree** | **1** | **n** | **n** |
| **End for loop** | **1** | **1** | **1** |
| **Close the file** | **1** | **1** | **1** |
| **Return nodes** | **1** | **1** | **1** |
| **Total Cost** | | | **10** |
| **Runtime** | | | **10** |

There are several advantages and disadvantages of each data structure. The advantages of using a vector is it is dynamic, meaning their size can be changed, whereas arrays have a fixed size and it can occupy reserved space in the memory, whereas arrays cannot. One disadvantage of using a vector is unlike arrays, vectors do not provide direct access to elements using an index. The advantages of using a hash table is that it provides quick access to data and it is dynamic and can grow or shrink as needed. One disadvantage of using a hash table is that a poorly designed hash function can lead to many collisions. The advantages of using a Binary Search Tree is that elements are stored in a sorted order making them easy to find and elements can be added efficiently. The disadvantage of using a Binary Search Tree is that it requires additional memory to store pointers to child nodes.

My recommendation is to use Binary Search Tree as the data structure for my code because the runtime analysis for milestone 3 has the minimum number for runtime and total cost, which means that it is the best case. “Best case” is defined as the scenario where there is a minimum number of steps an algorithm takes to complete its task.