**Analysis:**

| **Vector Runtime Analysis** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **LOAD FUNCTION** |  |  |  |
| DEFINE variables inputStream, currentLine string, and curren... | 3 | 1 | 3 |
| OPEN text file from specified path | 1 | 1 | 1 |
| IF text file does not open | 1 | 1 | 1 |
| WHILE the text file does not fail to receive input | 1 | n | n |
| INPUT line from text file and ASSIGN to currentLine | 2 | n | 2n |
| DEFINE a new Course object as tempCourse | 2 | N | 2n |
| **CALL parseInput** **function** with (currentLine, ',', 0, and temp... | 68 | 6n | (68\*8)n |
| IF (tempCourse’s courseNumber) or (tempCourse’s course)... | 2 | N | 2n |
| DEFINE iterationFactor | 1 | N | n |
| For EACH prerequisite in tempCourse | 1 | 4n | 4n |
| ASSIGN iterationFactor with 0 | 1 | 4n | 4n |
| FOR EACH Course in courseVector | 1 | 4n(n) | 4n^2 |
| IF (Course's courseTitle) is equal to (prerequisite) | 2 | 4n(n) | 8n^2 |
| IF iterationFactor equals 0 | 1 | 1 | 1 |
| APPEND tempCourse to courseVector | 5 | N | 5N |
| CLOSE inputStream | 1 | 1 | 1 |
|  |  |  |  |
| **PARSE FUNCTION. It is called above ^** |  |  |  |
| DEFINE currentIndex as 0, outputString as empty string, and… | 6 | 1 | 36 |
| WHILE loop iterates until currentChar equals the delimiter | 1 | 10 | 10 |
| APPEND currentChar to end of outputString | 1 | 10 | 10 |
| IF currentIndex is greater than or equal to (the length of the… | 1 | 10 | 10 |
| IF recursionIndex is 0  ELSE IF  ELSE  EXIT LOOP | 3 | 1 | 3 |
| IF recursionIndex is 0  ELSE IF  ELSE | 3 | 1 | 3 |
| CALL parseInput recursively with the substring of the input… | 1 | 6 | 6 |
|  |  |  |  |
| **Total Cost** | | | 12n^2 +565n + 7 |
| **Runtime** | | | O(n^2) |

| **HashTable Runtime Analysis** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **LOAD FUNCTION** |  |  |  |
| DEFINE variables inputStream, currentLine string, and curren... | 3 | 1 | 3 |
| OPEN text file from specified path | 1 | 1 | 1 |
| IF text file does not open | 1 | 1 | 1 |
| WHILE the text file does not fail to receive input | 1 | n | n |
| INPUT line from text file and ASSIGN to currentLine | 2 | n | 2n |
| DEFINE a new Course object as tempCourse | 2 | N | 2n |
| **CALL parseInput** **function** with (currentLine, ',', 0, and temp... | 68 | 6n | (68\*8)n |
| IF (tempCourse’s courseNumber) or (tempCourse’s course)... | 2 | N | 2n |
| DEFINE iterationFactor | 1 | N | n |
| For EACH prerequisite in tempCourse | 1 | 4n | 4n |
| ASSIGN iterationFactor with 0 | 1 | 4n | 4n |
| FOR EACH Course in courseHashTable | 1 | 4n(n) | 4n^2 |
| IF (Course's courseTitle) is equal to (prerequisite) | 2 | 4n(n) | 8n^2 |
| IF iterationFactor equals 0 | 1 | 1 | 1 |
| **CALL courseHashTable's Insert method** with tempCourse... | 2n + 19 | N | 2N^2 + 19N |
| CLOSE inputStream | 1 | 1 | 1 |
|  |  |  |  |
| **PARSE FUNCTION. It is called above ^** |  |  |  |
| DEFINE currentIndex as 0, outputString as empty string, and… | 6 | 1 | 36 |
| WHILE loop iterates until currentChar equals the delimiter | 1 | 10 | 10 |
| APPEND currentChar to end of outputString | 1 | 10 | 10 |
| IF currentIndex is greater than or equal to (the length of the… | 1 | 10 | 10 |
| IF recursionIndex is 0  ELSE IF  ELSE  EXIT LOOP | 3 | 1 | 3 |
| IF recursionIndex is 0  ELSE IF  ELSE | 3 | 1 | 3 |
| CALL parseInput recursively with the substring of the input… | 1 | 6 | 6 |
|  |  |  |  |
| **HashTable, Insert Method** |  |  |  |
| DEFINE local variables: integers key, bucketIndex| Node pointer… | 4 | 1 | 4 |
| ASSIGN key to course's hashCode, bucketIndex to a hash func... | 3 | 1 | 3 |
| ASSIGN newNode to a new Node and currentNode to nullpointer | 2 | 1 | 2 |
| ASSIGN currentNode to the element inside the nodeVector... | 5 | 1 | 5 |
| IF currentNode is a nullpointer | 2 | 1 | 2 |
| ELSE IF currentNode’s key equals (integer max constant) | 2 | 1 | 2 |
| WHILE currentNode’s next pointer is not a null pointer | 1 | N | n |
| Iterate until end of linkedList | 1 | N | N |
| ASSIGN currentNode’s next pointer to the newNode | 1 | 1 | 1 |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** | | | 14n^2 +580n + 7 |
| **Runtime** | | | O(n^2) |

| **Binary Search Tree Runtime Analysis** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **LOAD FUNCTION** | --- | --- | --- |
| DEFINE variables inputStream, currentLine string, and curren... | 3 | 1 | 3 |
| OPEN text file from specified path | 1 | 1 | 1 |
| IF text file does not open | 1 | 1 | 1 |
| WHILE the text file does not fail to receive input | 1 | n | n |
| INPUT line from text file and ASSIGN to currentLine | 2 | n | 2n |
| DEFINE a new Course object as tempCourse | 2 | N | 2n |
| **CALL parseInput** **function** with (currentLine, ',', 0, and temp... | 68 | 6n | (68\*8)n |
| IF (tempCourse’s courseNumber) or (tempCourse’s course)... | 2 | N | 2n |
| DEFINE iterationFactor | 1 | N | n |
| For EACH prerequisite in tempCourse | 1 | 4n | 4n |
| ASSIGN iterationFactor with 0 | 1 | 4n | 4n |
| FOR EACH Course in courseBST | 1 | 4n(n) | 4n^2 |
| IF (Course's courseTitle) is equal to (prerequisite) | 2 | 4n(n) | 8n^2 |
| IF iterationFactor equals 0 | 1 | 1 | 1 |
| **CALL courseBST's Insert method** with tempCourse... | 5n +22 | N | 5n^2 + 22N |
| CLOSE inputStream | 1 | 1 | 1 |
|  |  |  |  |
| **PARSE FUNCTION. It is called above ^** | --- | ---- | --- |
| DEFINE currentIndex as 0, outputString as empty string, and… | 6 | 1 | 36 |
| WHILE loop iterates until currentChar equals the delimiter | 1 | 10 | 10 |
| APPEND currentChar to end of outputString | 1 | 10 | 10 |
| IF currentIndex is greater than or equal to (the length of the… | 1 | 10 | 10 |
| IF recursionIndex is 0  ELSE IF  ELSE  EXIT LOOP | 3 | 1 | 3 |
| IF recursionIndex is 0  ELSE IF  ELSE | 3 | 1 | 3 |
| CALL parseInput recursively with the substring of the input… | 1 | 6 | 6 |
|  |  |  |  |
| **BinarySearchTree, Insert Method** |  |  |  |
| DEFINE newNode with (a new node and the course argument) | 5 | 1 | 5 |
| IF root is a nullpointer | 2 | 1 | 2 |
| CALL addNode function with newNode and course arguments | 1 | 1 | 1 |
|  |  |  |  |
| **BinarySearchTree, addNode Method** |  |  |  |
| ASSIGN node’s course with course argument | 1 | 1 | 1 |
| DEFINE currentNode with root, parentNode with nullpointer… | 9 | 1 | 9 |
| WHILE currentNode is not a nullpointer | 1 | N | N |
| IF key is less than currentNode’s key | 1 | N | N |
| ASSIGN parentNode with currentNode and currentNode’s left... | 3 | N/2 | 3N/2 |
| ASSIGN parentNode with currentNode, currentNode with currentNode with currentNode’s right, direction with 1 | 3 | N/2 | 3N/2 |
| IF direction equals 0 | 2 | 1 | 2 |
| ELSE IF direction equals 1 | 2 | 1 | 2 |
| **Total Cost** | | | 17n^2 +583n + 7 |
| **Runtime** | | | O(n^2) |

**Explain the advantages and disadvantages of each structure in your evaluation**

The analysis that is done in this document describes the worst case scenario for each of the three data structures. A more complete analysis of these algorithms is needed to draw a better conclusion. The average runtime for each of these algorithms would be more helpful in determining which data structure to use for the listed requirements. The reason I say this is because the average runtimes of a binary search tree, hashtable, and vector in searching for a specific course differ drastically. The worst case scenarios of each data structure are similar in runtime requirements.

A vector needs to compare every element of the vector to each prerequisite to ensure that all prerequisites are present. The worst case scenario for a binary search tree is all of the entries are given in order. The binary search tree essentially becomes a linkedlist since each course needs to be compared individually in a line of nodes that all direct towards the right. The worst case scenario for a hash table is the hashing function directs all of the courses into the same linked list. Searching through a linked list is similar to a search through a vector for runtime speeds.

Inserting a course into each respective data structure does have differing effects in the worst case scenario. The vector simply needs to add the course at the end of the vector using only a few constant time operations. The hash table needs to iterate through the entire linkedlist that was created due to the large number of collisions until it finds the end. This operation can take N times to complete. The situation is similar for a binary search tree, where a sorted input leads to a list of N length to iterate through.

Considering average run times of the three data structures: a vector will still typically require o(n) time to find a particular course; a binary search tree will require o(log(n)) time to find a course; and a hash table will require close to constant time operations searching for a particular course. The vector still has constant time operations for inserting a new course object. The binary search tree will have O(logN) time for inserting a course. The hash table will have constant time for inserting a course with a decent hashing function.

Based purely on the worst case scenario, the best choice would be the vector. Based on known averages for each data structure however, the best choice would be the hash table. This analysis is purely based on the runtime analysis for inputting data into the Course data structure.