**Runtime Analysis**

**Vector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Define course object structure:  (integer) Course number  (string) Course Title  (array) Course Prerequisites | 4 | 1 | 4 |
| Initialize vector of data type Course | 1 | 1 | 1 |
| Open input file | 1 | 1 | 1 |
| If input file could not be opened:  Display error message “Error, input file could not be read”  Close program | 4 | 1 | 4 |
| For each line in the document | 1 | n | n |
| Iterate through the line storing data, divided by a space character | 1 | n | n |
| (integer) Course number = the first value of the iterator | 1 | n | n |
| (string) Course title = the second value of the iterator | 1 | n | n |
| If there is no third value:  (array) Course Prerequisites = null | 3 | n | n |
| Else:  (Boolean) matchingNumber = false  For each line in the document  If the first value in the line equals a number in the original third value  matchingNumber = true  if matchingNumber = true  (array) Course Prerequisites = the third value of the iterator  Else, (array) Course Prerequisites = null | 9 | N2 | N2 |
| Create a new Course object, passing Course number, Course title, and Course Prerequisites as parameters | 1 | 1 | 1 |
| Add the new course object to the end of the Course vector | 1 | n | n |
| For the courses in the course vector | 1 | n | n |
| (integer) j = current index of the vector | 1 | n | n |
| While j > 0 and the string comparison of the course name of the Course object at index j is less than the string comparison of the course name of the Course object at index (j – 1): | 1 | n | n |
| (Course) temp = course object at index j | 1 | N2 | N2 |
| Course object at index j = course object at index (j -1) | 1 | N2 | N2 |
| Course object at index (j-1) = temp | 1 | N2 | N2 |
| Decrease j by one | 1 | N2 | N2 |
| For all courses in the Courses vector, given a course number  If the course’s course number matches the given course number  Print out course information  For each course prerequisite  Print prerequisite course information | 1 | N2 | N2 |
| **Total Cost** | | | 6n2 + 9n + 11 |
| **Runtime** | | | O(n2) |

**Hashtable**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Define course object structure:  (integer) Course number  (string) Course Title  (array) Course Prerequisites | 4 | 1 | 4 |
| Initialize vector of data type Course | 1 | 1 | 1 |
| Open input file | 1 | 1 | 1 |
| If input file could not be opened:  Display error message “Error, input file could not be read”  Close program | 4 | 1 | 4 |
| For each line in the document | 1 | n | n |
| Iterate through the line storing data, divided by a space character | 1 | n | n |
| (integer) Course number = the first value of the iterator | 1 | n | n |
| (string) Course title = the second value of the iterator | 1 | n | n |
| If there is no third value:  (array) Course Prerequisites = null | 3 | n | n |
| Else:  (Boolean) matchingNumber = false  For each line in the document  If the first value in the line equals a number in the original third value  matchingNumber = true  if matchingNumber = true  (array) Course Prerequisites = the third value of the iterator  Else, (array) Course Prerequisites = null | 9 | N2 | N2 |
| Create a new Course object, passing Course number, Course title, and Course Prerequisites as parameters | 1 | 1 | 1 |
| Add the new course object to the end of the Course vector | 1 | n | n |
| Define Node structure:  (course) class  (Node pointer) next = null | 3 | 1 | 3 |
| Initialize vector of data type Node | 1 | 1 | 1 |
| Resize Node vector to size of Course vector | 1 | 1 | 1 |
| Define hash function given a (string) course title:  (integer) Hash key = (ascii to integer conversion of c string conversion of course object’s title) MOD (size of Node vector)  Return key | 4 | 1 | 4 |
| For each course item in the Course vector: | 1 | N | N |
| Calculate hash key | 1 | N | N |
| Create Node object using current Course object | 1 | N | N |
| Check index of Node vector at hash key  If the bucket is empty  Insert Node item at this location | 3 | N | 3n |
| Else  Find the last value entered at this location (Node with next pointer = null) | 3 | N | 3n |
| Assign the next pointer value of that Node to the Node currently being worked with | 1 | N | N |
| For the node at the beginning of the Node vector to the node at the end of the Node vector: | 1 | N | N |
| If the current node is not empty | 1 | N | N |
| While the current node’s next pointer does not point to null: | 1 | N2 | N2 |
| Print course object information | 1 | N2 | N2 |
| For each course prerequisite | 1 | N3 | N3 |
| Print prerequisite course information | 1 | N3 | N3 |
| Current node = current node’s next pointer | 1 | N2 | N2 |
| **Total Cost** | | | 2n3 + 4n2 + 18n + 20 |
| **Runtime** | | | O(N3) |

**Binary Tree**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Define course object structure:  (integer) Course number  (string) Course Title  (array) Course Prerequisites | 4 | 1 | 4 |
| Initialize vector of data type Course | 1 | 1 | 1 |
| Open input file | 1 | 1 | 1 |
| If input file could not be opened:  Display error message “Error, input file could not be read”  Close program | 4 | 1 | 4 |
| For each line in the document | 1 | n | n |
| Iterate through the line storing data, divided by a space character | 1 | n | n |
| (integer) Course number = the first value of the iterator | 1 | n | n |
| (string) Course title = the second value of the iterator | 1 | n | n |
| If there is no third value:  (array) Course Prerequisites = null | 3 | n | n |
| Else:  (Boolean) matchingNumber = false  For each line in the document  If the first value in the line equals a number in the original third value  matchingNumber = true  if matchingNumber = true  (array) Course Prerequisites = the third value of the iterator  Else, (array) Course Prerequisites = null | 9 | N2 | N2 |
| Create a new Course object, passing Course number, Course title, and Course Prerequisites as parameters | 1 | 1 | 1 |
| Add the new course object to the end of the Course vector | 1 | n | n |
| Define Node structure:  (Course) class  (Node pointer) left = null  (Node pointer) right = null | 4 | 1 | 4 |
| Initialize root Node, setting to null | 1 | 1 | 1 |
| Initialize current node, set equal to root node | 1 | 1 | 1 |
| For each object in the course vector | 1 | N | N |
| If the root node is null, set the root’s course equal to the course of the current vector item | 1 | N | N |
| Else | 1 | N | N |
| If the ascii to integer conversion of the c string conversion of the course title is higher than the ascii to integer conversion of the c string conversion of the course title in the root node | 1 | N | N |
| If the current node’s right pointer is equal to null pointer | 1 | N | N |
| Create node object and assign to current node’s right pointer | 1 | N | N |
| Else | 1 | N | N |
| Make recursive call to this function, passing current node and class number as parameters | 1 | N Log n | N Log n |
| Else | 1 | N | N |
| If the current node’s left pointer is equal to null pointer | 1 | N | N |
| Create node object and assign to current node’s left pointer | 1 | N | N |
| Else | 1 | N | N |
| Make recursive call to this function, passing current node and class number as parameters | 1 | N Log n | N Log n |
| Starting with the root node: | 1 | 1 | 1 |
| If the given node is not null | 1 | 1 | 1 |
| Call this again, using the given node’s left pointer as the parameter | 1 | Log n | Log n |
| Print course information | 1 | 1 | 1 |
| Print course prerequisites | 1 | 1 | 1 |
| Call this again, using the given node’s right pointer as the parameter | 1 | Log n | Log n |
| **Total Cost** | | | N2 + 2n(Log n) + 17n + 21 |
| **Runtime** | | | O(N2) |

Upon analyzing the various data structure options for the program, it was found that the binary tree and the vector structure were tied for the fastest with O(n2), while the hashtable was slowest with O(n3). That said, a great deal of the slower than expected outcome is due to the algorithm that reads the given files and builds the data into Course objects. Further, the hashtable’s result of O(n3) is due to the printing portion, which would assume all contents are chained in the same bucket with lengthy prerequisites, a worst case scenario that is highly unlikely to happen in a live environment.

Omitting the performance of the reading to object algorithm that is constant between the three, we would find the binary search tree would win out with a runtime more on the order of O(n log n), though depending on the layout of the data in the file, it could still easily remain O(N2), if the course titles in the file are already sorted, for instance, such that the tree resembles a list. Conversely, if such data were mostly organized as such, the hashtable may prove more ideal, as it could sort closer to linear time. That said, it would be more ideal to sort the hashtable by course number rather than title, as it would result in more evenly spread buckets (for instance, if the hash key function resulted in all courses with a title beginning “Intro to…” going into the same bucket, it would inevitably result in a massive bucket, while several others surrounding it may go completely unused).

While the tree had the same runtime as the vector, one must consider that the total number of courses wont be very high (likely on the range of a few hundred, versus many thousands or more). As such, the total run cost of the programs should be considered as well, as the constant has less data to “hide” itself into. In the analysis, we find that the tree structure had one instance of n2 time, while the vector had six. As such, it is very likely that the final program would feel that multiplier in a live environment.

In all, despite the worst case scenario being that it would run the same as the vector structure, the tree structure would provide the most potential for the best performance. Further, removing common denominators, the tree’s print function was the only one that would work faster than O(n2). As such, I would recommend that a tree data structure be used to contain the course objects in the proposed program.