Program Requirements:

1. Print a list of all the Computer Science courses in alphanumeric order.
2. For a given course, print out its title and prerequisites.

**Vector Time Analysis**

| Code | Line Cost | # Times Executed | Total Cost |
| --- | --- | --- | --- |
| Create file reader | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| Reading file line by line | 1 | n | n |
| Creating Course Object from line | 1 | 1 | 1 |
| Adding Course Object to List | 1 | 1 | 1 |
| Printing out Course List | 1 | n | n |
| Look up desired course | 1 | n | n |
| Print out Course Title | 1 | 1 | 1 |
| Print out prerequisites | 1 | n | n |
| **Total Cost** |  |  | **4n + 5** |
| **Run Time** |  |  | **O(n)** |

**Hash Table Time Analysis**

| Code | Line Cost | # Times Executed | Total Cost |
| --- | --- | --- | --- |
| Create file reader | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| Reading file line by line | 1 | n | n |
| Creating Course Object from line | 1 | 1 | 1 |
| Adding Course Object to List | 1 | 1 | 1 |
| Printing out Course List | 1 | n | n |
| Look up desired course | 1 | 1 | 1 |
| Print out Course Title | 1 | 1 | 1 |
| Print out prerequisites | 1 | n | n |
| **Total Cost** |  |  | **3n + 6** |
| **Run Time** |  |  | **O(n)** |

**Binary Search Tree Time Analysis**

| Code | Line Cost | # Times Executed | Total Cost |
| --- | --- | --- | --- |
| Create file reader | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| Reading file line by line | 1 | n | n |
| Creating Course Object from line | 1 | 1 | 1 |
| Adding Course Object to List | 1 | 1 | log(n) |
| Printing out Course List | 1 | n | n |
| Look up desired course | 1 | log(n) | log(n) |
| Print out Course Title | 1 | 1 | 1 |
| Print out prerequisites | 1 | n | n |
| **Total Cost** |  |  | **2log(n) + 2n + 4** |
| **Run Time** |  |  | **Log(N)** |

Pseudocode Analysis

1. Print a list of all the Computer Science courses in alphanumeric order.
2. For a given course, print out its title and prerequisites.

Regardless of which data structure is utilized, printing out a list of courses will take O(n) time and will be based on the number of elements in the list. There isn’t a way to make this faster in any of the data elements. When it comes to printing out a course and its prerequisites, it first requires that we find the course in question. From a worst case analysis (assuming the course doesn’t exist or is at the very end of the data structure), vector’s and hash tables each have a time complexity of O(n) yet a binary search tree since it is balanced has a time complexity of O(log n) which would be faster to access. However, on average, a binary search tree has a time complexity of O(log n) and a vector has a time complexity of O(n), yet a hash table has an average access time of O(1). Unless you knew ahead of time what prerequisites a course contained, you would need to loop through the vector storing the prerequisites and again all three would have a time complexity of O(n). Sorting the courses into alphanumeric order right now isn’t too bad but if the course list were to expand it could get tricky. Despite the run times listed, given the circumstances, I believe a hash table would be the best data structure to satisfy the requirements. It allows for very fast insertions when sorting and very fast lookup times if a user needed to look up a specified course. If I were to perform the same operations with a vector, the insertions would be the same O(1) but I would need to iterate through the whole vector O(n) to find the desired course. Binary trees for both operations have an average time complexity of O(log n) since you are only traversing through half of the elements in the tree. Binary would have the worst insertion times but the second best access times.

**Vector**

**Advantages:**

1. Can resize itself efficiently to adjust to an increase/decrease in data elements.
2. Fast insertion (if element is stored at end of vector)
3. Easier to set up and manage memory compared to hash tables and binary search trees

**Disadvantages:**

1. Slower access time compared to hash tables and binary search trees since the vector needs to be traversed sequentially.
2. Not able to randomly access a specified element
3. Can take up extra memory due to the usage of one (or more) pointers.

**Hash Table**

**Advantages:**

1. Fast insertion, deletion, searching and access operations
2. Good way to organize large sets of data since each value needs to be stored with a unique key
3. When properly optimized they have a high performance compared to other data structures

**Disadvantages:**

1. If the hash table isn’t properly optimized and/or the data set is large, this can lead to a high number of collisions and greatly reduce its efficiency.
2. Can be difficult to optimize properly
3. Setting up the initial hash table can become quite complex

**Binary Search Tree**

**Advantages:**

1. Fast insertion and deletion when the tree is balanced (child nodes on both the left and right of the parent node) compared to vectors.
2. Faster traversal time compared to vector (O log(n) vs O(n))
3. Takes up less storage space compared to a hash table since hash tables need extra space to take into account potential new insertions while a BST only uses what memory it needs.

**Disadvantages:**

1. If the tree isn’t balanced, operations can become inefficient and take more time to implement.
2. Can be harder to manage structure when it comes to deleting elements since you need to take into account pointers
3. Slower operations (insertion, deletion, search) compared to hash tables