CS300 Module Six Project One Evaluation

**VECTOR**

n = number of courses, k = number of prerequisites

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Check if file is opened | 1 | 1 | 1 |
| Loop over each line | 1 | n | n |
| Split lines into tokens | 1 | n | n |
| Create COURSE object | 1 | n | n |
| Assign course number and name to object | 1 | n | n |
| Loop over each prerequisite | k | n | k\*n |
| Append to vector | 1 | n | n |
| Validating outer loop | 1 | n | n |
| Validating inner loop | k | n | k\*n |
| Search for each prerequisite | n | n\*k |  |
| Total Cost | | | 𝑘+2nk+6𝑛+2 |
| Runtime | | |  |

**HASH TABLE**

n = number of courses, k = number of prerequisites

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Check if file is opened | 1 | 1 | 1 |
| Loop over each line | 1 | n | n |
| Split lines into tokens | 1 | n | n |
| Create COURSE object | 1 | n | n |
| Assign course number and name to object | 1 | n | n |
| Loop over each prerequisite | k | n | n\*k |
| Append prerequisites to list | 1 | n\*k | n\*k |
| Insert object into hash table | 1 | n | n |
| Validating outer loop | 1 | n | n |
| Validating inner loop | k | n | n\*k |
| Search for each prerequisite | 1 | n\*k | n\*k |
| Total Cost | | | 4nk+6𝑛+2 |
| Runtime | | |  |

**Binary Search Tree**

n = number of courses, k = number of prerequisites

| Code | Line Cost | | # Times Executes | Total Cost |
| --- | --- | --- | --- | --- |
| Open file | | 1 | 1 | 1 |
| Check if file is opened | | 1 | 1 | 1 |
| Loop over each line | | 1 | n | n |
| Split lines into tokens | | 1 | n | n |
| Create COURSE object | | 1 | n | n |
| Assign course number and name to object | | 1 | n | n |
| Loop over each prerequisite | | k | n | n\*k |
| Add prerequisites to object | | 1 | n\*k | n\*k |
| Insert object into tree | | logn | n | n(logn) |
| Call loadPrereqs | | k(logn) | n | n(klogn) |
| Total Cost | | | | n(klogn) + n(logn) + 2nk + 4n + 2 |
| Runtime | | | | O(n(klogn)) |

**Comparison and Recommendation:**

After creating a runtime analysis of all three data structures and comparing line by line for loading course data from the file, reading each line, transferring each course into an object, and ensuring that all course data is in the object each data structure has its advantages in different scenarios. The vector is much simpler and does not require much in cost compared to a hash table and binary search tree. This data structure is good if the course data file is small and does not require a quick and complex program to do the job. However, the time to search for a course is linear O(n) and that can become inefficient if the course data file were to grow. The hash table is like the vector; however, it can handle a larger course data file and is much quicker in searching courses as it would be constant time O(1) because the program would search by course number .The binary search tree is slower than the hash table and the vector due to the time to load and insert objects into the tree. Searching for courses in a tree costs the most O(logn) and will result in a slower course retrieval time. The advantage of using this data structure though is it is already sorted as new course objects are inserted into the tree. The binary search tree is good if the courses need to be always sorted, but not for efficiency.

My recommendation is the hash table as it can handle a more complex course data file if needed, it is much more efficient and quicker compared to the other two data structures. The cost of searching for a course is constant, making it quick to retrieve the course. These two factors are the most important as a hash table can allow easy access to any course and add more courses into the system. Although the vector and binary search tree had their own advantages in different areas, I think that the hash table would be the best fit.