1. Based on the advisor’s requirements, analyze each data structure (vector, hash table, and tree). **Explain the advantages and disadvantages of each structure in your evaluation.**

The milestone has three different data structure implementation options which include a vector, a hash table, and a binary search tree. The data structures will store computer science course information which include course ID, course name, and prerequisite courses. The functionality required from the milestone includes reading, parsing, and storing each course into the respective data structures. Once the data structures contain the courses and its information, the program will require searching, prerequisite checking, printing specific courses, printing courses from lowest to highest ID, and printing all courses.

The vector data structure provides the ability to store each course object in an indexable position. This property provides advantages as accessing the vector can be done in constant time. Therefore, to insert course objects into the end of a vector, the worst-case runtime will be O(N\*M). The insert will depend on the input size, N, as well as the number of prerequisites per course, M. However, when searching for a specific course, constant time accesses do not provide value as each course object’s course ID will have to be compared to the inputted course ID. The search for a course will have the worst-case runtime of O(N). When checking for the number of prerequisite courses, the search function will need to be utilized to find the course and then the number of prerequisites can be determined. This will result in a worst-case runtime of O(N). When printing a specific course, the search function will need to be utilized to find the course and then print the course. This will also result in a worst-case runtime of O(N). In order to print courses from lowest to highest ID, the vector will need to be sorted. QuickSort will be implemented to sort the vector and has a worst-case runtime of O(N2) and an average-case runtime of O(NlogN). The printing function will then start at the beginning of the vector and print each course. As each element in the vector is visited and printed, the worst-case runtime of this function will be O(N). This will give a combined worst-case runtime of O(N2) However, once the vector is sorted, the search function’s runtime can be improved to O(logN) by implementing binary search, which would result in the functions that utilize the search function to also improve to O(logN). Similar to printing courses from lowest to highest ID, printing all courses will have to start at the beginning of the list and print each course. This will also result in a runtime of O(N). The analysis shows that using the vector data structure for this application results in a runtime of O(N) for most functions. The application avoids the vector’s main disadvantage of shifting which results when inserting and removing from the front or middle of the data structure. Overall, the runtime of O(N) is not the most concerning as worst runtimes exist.

The hash table data structure provides a strong average-case runtime for inserts, searches, and deletions of O(1). The hash function will require a runtime of O(N), where N represents the length of the string, as the string will need to be traversed to convert it to an integer. In the process of inserting the courses into a hash table, each course ID will have to be hashed and the resulting key will determine the vector’s index where they will be inserted. Although the average-case for insert is O(1), in the worst-case, there are maximum collisions and the linked-list at the vector’s index will have to be traversed in order to insert the course object. Therefore, the worst-case runtime of insertion into a hash table is O(N). When searching for a specific course, the given course ID will be hashed, the resulting key will be utilize to index into the hash table’s array, and the course will be searched for depending on whether or not more than one element exists in the linked-list. Although the average-case runtime is O(1), the worst-case occurs when there are maximum collisions and the linked-list needs to be traversed to the end to find the course ID. Therefore, the worst-case runtime for insertion is O(N). In order to check for the number of prerequisites and print a specific course, the search function will be utilized. The search in the average-case is O(1) when there is minimal collisions. However, in the worst-case, all of the elements will hash to the same bucket and the search will need to traverse the linked-list with all elements. Therefore, the worst-case runtime of the search function is O(N) and determining the number of prerequisites and printing a specific course will have a worst-case runtime of O(N). To print the courses from lowest to highest ID, the hash table will need to be sorted. However, in order to sort a hash table, it will first have to copied back into a vector data structure. As each element will be traversed, converting to a vector data structure will require a runtime of O(N). Then the QuickSort algorithm is implemented giving a worst-case runtime of O(N2) and average-case runtime of O(NlogN). The vector can then be iterating through as it prints each course, which give the worst-case runtime of O(N). This will give a combined worst-case runtime of O(N2). Printing all the courses from the hash table will require every element to be iterated over. Therefore, the worst-case, where there are maximum collisions, will be O(N). Although many of the worst-case runtimes are similar to the vector data structure, the advantage of a hash table is that it has an average amortized case of O(1) for most functions. However, there is a disadvantage in that sorting a hash table requires copying the elements into a different data structure, which requires additional time and space. Therefore, it should be the preferred data structure when many inserts, searches, and deletions are required without the need for elements being sorted.

The binary search tree data structure provides a good balance in terms of average-runtime complexities for insert, search, and remove functionalities. The binary search tree has an advantage in that it maintains an order from lowest to highest as it inserts new elements. However this property can also serve as a disadvantage as it can result in a maximum tree if the elements are continuously smaller than the previous. The average-case runtime of insert is O(logN) which occurs when the tree is a minimal binary search tree. The worst-case runtime of insert is O(N) which occurs when the tree is a maximum binary search tree. In order to determine the number of prerequisites and print a specific course, a search function is required for the binary search tree. The binary search tree has a average-case runtime of O(logN) when the tree is a minimal binary search tree. However, in the worst-case when the binary search tree is a maximum tree, the runtime is O(N). Therefore, determining the number of prerequisites and printing a specific course will have a worst-case runtime of O(N). In order to print the courses from lowest to highest course ID, the binary search tree has an advantage as it is already sorted. Therefore, to print from the lowest to highest course ID, the binary search tree needs to print using an in-order traversal. In the average-case, the binary search tree will be a minimal tree and the in-order traversal runtime will be O(N). In the worst-case, the binary search tree will have a maximum height and resemble a linked-list. Therefore, the worst-case runtime for printing a binary search tree in sorted order will be O(N). Printing all courses from a binary search tree will have similar runtimes to printing using an in-order traversal. As order does not matter, post-order and pre-order traversals can be implemented. The worst-case runtime, regardless of the type of traversal, will be O(N). In the analysis, a binary search tree has a worst-case runtime for all functions of O(N). However, the average-case runtimes for the functions on average are O(logN).

Worst-Case Runtime Analysis Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Functions | Vector | Hash Table | Binary Search Tree |
| Insert | O(N) | O(N) | O(N) |
| Create data structure with objects | O(N\*M) | O(N2) | O(N2) |
| Search | O(N) | O(N) | O(N) |
| Determine number of prerequisites | O(N) | O(N) | O(N) |
| Print course | O(N) | O(N) | O(N) |
| Print all course | O(N) | O(N) | O(N) |
| Sort and print all courses | O(N2) | O(N2) | O(N) |

Average-Case Runtime Analysis Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Functions | Vector | Hash Table | Binary Search Tree |
| Insert | O(N) | O(1) | O(logN) |
| Create data structure with objects | O(N\*M) | O(N) | O(NlogN) |
| Search | O(N) | O(1) | O(logN) |
| Determine number of prerequisites | O(N) | O(1) | O(logN) |
| Print course | O(N) | O(1) | O(logN) |
| Print all course | O(N) | O(N) | O(N) |
| Sort and print all courses | O(NlogN) | O(NlogN) | O(N) |

Auxiliary Space Complexity Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Functions | Vector | Hash Table | Binary Search Tree |
| Insert | O(1) | O(1) | O(1) |
| Create data structure with objects | O(N) | O(N) | O(N) |
| Search | O(1) | O(1) | O(1) |
| Determine number of prerequisites | O(1) | O(1) | O(1) |
| Print course | O(1) | O(1) | O(1) |
| Print all course | O(1) | O(1) | O(N) |
| Sort and print all courses | O(logN) | O(N) | O(N) |

1. Now that you have analyzed all three data structures, **make a recommendation for which data structure you will plan to use in your code**. Provide justification for your recommendation, based on the Big O analysis results and your analysis of the three data structures.

Based on the analysis, it is hard to determine the best data structure assuming the worst-case runtimes for all of the functions. However, when examining the average-case runtime of the data structures, the hash table provides the best runtimes for all the functionalities. The amortized average-case for a hash table provides constant time for the functions that require insertion and searches. When printing all courses, all data structures provide the same runtime as each course object will have to be visited and printed. In the case of printing all and sorting with printing, the binary tree has the overall best runtime; however, the space complexity is the worst as it is implemented recursively and each call with add to the call stack. The hash table gives the best balance between all the functionalities required in the program. Therefore, assuming that inserts and searches dominate the program functionality, the hash table will be the best data structure in this scenario and will be used for the code.