**Evaluation**

**Vector Advantages:**

Vectors are one of the simplest data structures to implement and use. Operations such as adding elements to the end are straightforward.Accessing any element by index is O (1), which means fast lookups when you need to access a specific course using its index.In most implementations, vectors can dynamically resize and use memory efficiently, which can minimize overhead.

**Disadvantages:**

Inserting a new course into the middle of the vector is O(n) because all elements after the insertion point need to be shifted. However, appending elements is O (1) on average. Searching for a specific course by course ID requires linear time, O(n), unless the list is sorted. While vectors manage memory well in terms of growth, if the vector must frequently resize, it might occasionally result in wasted memory or performance slowdowns due to reallocation.

**Hash Table** **Advantages:**

The major advantage of a hash table is its constant-time lookup, O (1), for both inserting and searching for a course using the course ID. Inserting new courses can be done in constant time, O (1), assuming no hash collisions. This is ideal for dynamically adding data. Hash tables allocate memory efficiently, as they adjust the table size as needed, reducing wasted space during insertions.

**Disadvantages:**

The primary disadvantage of using a hash table is that it doesn't maintain any order of insertion. Hash collisions can reduce the efficiency of hash tables. In the worst case (if the hash function produces many collisions), it could degrade the performance to O(n) for lookups and inserts. Deleting or updating courses in a hash table can be slightly more complicated than with other structures because it requires proper handling of collisions and keys.

**Binary Search Tree Advantages:**

A binary search tree (BST) inherently maintains elements in a sorted order. In-order traversal will always give the elements sorted, which is advantageous when you need to print the courses in alphanumeric order. Searching for a course by its course ID in a balanced BST is much faster than a linear search. This is a significant improvement over the vector and hash table, especially when there are many courses. In a balanced BST, insertions and deletions can be done in O (log n). This is better than vector insertions which are O(n) in the worst case, especially for large datasets.

**Disadvantages:**

Implementing a balanced BST is more complex than using a vector or hash table. You need to ensure that the tree remains balanced after each insertion and deletion. The performance of BST operations (search, insert, delete) is O(log n) while hash tables provide O(1) time complexity for the same operations. Each node in the tree requires additional pointers, which increases the memory overhead compared to a simple array or hash table.

**Recommendation**

After evaluating all three data structures, I would recommend using a Hash Table for the following reasons:

* **Fast Lookup and Insertion**: The hash table provides constant-time O (1) operations for insertion and search, which makes it ideal for efficiently managing the courses data. As the requirements involve adding, searching, and potentially modifying course data frequently, hash tables excel in this area.
* **Memory Efficiency**: Hash tables are memory-efficient and flexible, resizing automatically to handle growth. For our needs, they would provide better performance than the vector in terms of insertion and deletion.
* **Handling Duplicate Entries**: Hash tables inherently avoid duplicate entries, which makes them a good fit for ensuring that each course ID is unique.
* **Scalability**: While a binary search tree (BST) is efficient for sorted data, the O (log n) operations are slower than hash tables' constant time for typical operations. If the application scales to many courses, the hash table will perform better in terms of both speed and memory usage.
* **Sorting Issues**: The downside of a hash table is that it doesn't maintain order, which means that if alphanumeric ordering of courses is needed, I would either: Sort the courses manually, when necessary, which would take O (log n) time. Alternatively, I could maintain a separate sorted structure (like a vector or list) for ordering when needed, which would help achieve both fast insertion/search and sorted output.

For loading and inserting courses, the hash table provides O (1) for each operation, leading to O(n) for **n** courses. This is more efficient than the O (n log n) required by a BST.Both the hash table and BST have O (n \* k) space complexity, but the hash table avoids the need for additional memory overhead used by tree nodes.In the worst case, with a poor hash function or many collisions, the hash table may degrade to O(n), but this is typically avoided with a good hash function and rehashing. On the other hand, BST operations are O (log n), which is better than vector operations but not as fast as hash tables in typical cases.