When designing a system to manage and print courses, we must consider the efficiency of different data structures. The three primary options, vector, hash table, and binary search tree. Each has strengths and weaknesses in terms of insertion, search, sorting, and memory usage. A vector allows easy iteration but is inefficient for searching and sorting, requiring O(n log n) time for sorting and O(n) for searching. A hash table provides O(1) average time insertion and search but does not maintain order, meaning it requires additional O(n log n) time when sorting for display. A BST, when balanced, allows for efficient O(log n) insertions and searches while maintaining sorted order naturally, making it useful when frequent sorting is required. However, an unbalanced BST degrades to O(n) for these operations, making it inefficient in the worst case.

The process of loading course data begins with reading a file containing course information. Each line of the file represents a single course and must be parsed into structured data. The operations involved include reading the file line by line, splitting each line into components, checking for formatting errors, creating a course object, and inserting it into the chosen data structure. Each of these steps operates in O(1) time, but the insertion step varies in complexity depending on the data structure used. If we store the data in a vector, appending to the end is O(1), but inserting in sorted order would be O(n) per insertion, leading to a worst-case total of O(n²). A hash table allows O(1) insertion per course, making the total complexity O(n). A BST requires O(log n) per insertion in the balanced case, resulting in an overall complexity of O(n log n).

Combining the file reading and data insertion operations, we find that a hash table provides the fastest performance with an O(n) overall complexity since it allows constant-time insertion and retrieval. A BST offers a balanced trade-off at O(n log n) due to logarithmic insertion times while maintaining a sorted structure, making it efficient for frequent ordering operations. A vector, however, performs poorly when maintaining order, as sorting requires O(n log n) time, and inserting in order could degrade to O(n²) in the worst case. Given that the advising system needs to display courses in sorted order, a BST or a hash table with sorting when needed would be the most practical choices. All three data structures require O(n) memory to store n course objects. However, the hash table has additional overhead for hash storage, while the BST requires extra space for pointers to child nodes. A vector has the smallest memory overhead but lacks efficient search and sorting operations. Therefore, if memory is a major concern and sorting is infrequent, a hash table is the best choice. If frequent sorted retrieval is needed, a BST is preferable, as it avoids the need for repeated sorting.

When evaluating the vector, hash table, and binary search tree data structures for ABCU’s advising program, each has its advantages and disadvantages. A vector is simple and efficient for sequential access, with low memory overhead, but it struggles with searching and inserting elements in sorted order, requiring O(n log n) time for sorting. This makes it less ideal for dynamic data and frequent sorted output. A hash table offers fast insertion and retrieval with O(1) average time complexity, making it ideal for quick course lookups. However, it does not maintain order, requiring additional sorting for displaying courses, and has higher memory overhead. On the other hand, a BST naturally maintains sorted order with an O(log n) time complexity for insertion and search in a balanced tree, making it the best choice when ordered output is frequently required. However, it can degrade to O(n) if unbalanced, and managing tree balancing adds complexity. Ultimately, the hash table is best for fast lookups when sorting is infrequent, while the BST is superior for maintaining sorted order with frequent updates and display requirements. A vector is best avoided for this application unless the dataset is small or static.

Based on the Big O analysis and the overall evaluation of the three data structures, I recommend using the Binary Search Tree for the advising program. The BST strikes a good balance between efficient insertion, retrieval, and maintaining sorted order, which is a key requirement for the program. The O(log n) complexity for both searching and inserting in a balanced BST ensures that the system can efficiently handle a growing number of courses. This is important because the advising program will likely need to frequently add new courses and display them in sorted order. The in order traversal of a BST naturally provides a sorted list of courses, eliminating the need for post processing sorting, which is required by the hash table.