To evaluate the suitability of vector, hash table, and binary search tree (BST) data structures for the ABCU Computer Science department’s advising program, I analyzed the worst case runtime of the loadCoursesFromFile function, which handles file reading, parsing, error checking, and course object creation, assuming n courses, each with up to p bounded prerequisites. For the vector implementation, the function opens the file, reads n lines, splits each line (costing k per line for line length), trims t tokens per line, creates a course object, adds p prerequisites, and validates prerequisites using a hash set. Each operation, such as file opening, reading, splitting, trimming, and inserting into the vector or set, has a constant cost per line, except for splitting and trimming, which depend on k and t, respectively. The total runtime sums to O(n + n·k + n·t + n·p), simplifying to O(n) since k, t, and p are constants. The hash table implementation follows a similar process, with identical operations except for inserting courses into a hash table instead of a vector, maintaining O(1) average case insertion. Its runtime is also O(n). For the BST, the process mirrors the vector, but inserting each course into the tree costs O(log n) for a balanced BST, and validation uses an O(n) in order traversal, yielding a total runtime of O(n·log n + n), dominated by O(n·log n).

Considering the advisor’s requirements, loading data, printing an alphanumerically ordered course list, and retrieving individual course information, the vector offers simplicity and efficient cache usage due to sequential memory allocation but suffers from O(n·log n) sorting for ordered lists and O(n) linear search for course lookups, making it less efficient for frequent searches. The hash table excels in loading and lookups with O(n) and O(1) average case runtimes, respectively, but requires O(n·log n) to sort course numbers for ordered printing, as it lacks inherent order. Memory wise, vectors use O(n) for storage, with minimal overhead, while hash tables may incur additional space due to collisions or resizing. The BST, when balanced, provides O(n·log n) loading, O(n) for ordered printing via in order traversal, and O(log n) for lookups, offering a balance of efficiency. However, it risks O(n) performance if unbalanced and is complex to implement with balancing mechanisms like AVL or Red Black trees. Memory usage for BSTs is O(n), with pointers adding slight overhead compared to vectors.

I recommend the BST for the program. Despite its O(n·log n) loading time compared to O(n) for vectors and hash tables, its O(n) ordered printing is significantly faster than the O(n·log n) sorting required by the others, which is critical given the frequent need for sorted course lists. Its O(log n) lookup time for individual courses is also competitive, surpassing the vector’s O(n) and approaching the hash table’s O(1). Assuming a balanced BST, it avoids worst case scenarios, and its memory efficiency is comparable to vectors. The BST’s ability to maintain sorted order inherently aligns with the advisors’ primary requirement, making it the optimal choice for this application.