Luke Barton

Prof. Jamalian

CS-300

12 October 2025

Run Time Evaluation

The above system, regardless of storage design, has two functions which are processed similarly in all three scenarios: reading in the file, and creating course objects. In the worst case scenario for vector storage, the read function will create an fstream object, set the separation character as comma, create a vector, and then begin a for loop. Since these functions are only accomplished once, they are constant. Then, a FOR loop iterates over each line of the code, checking if the line contains a valid course description, initializing the first and second elements to the course ID and title variables and then checking if there are prerequisites. In the worst case scenario, there will be two prerequisites, which will require the system to check two additional elements on the line to ensure they are valid courses, and then append those elements to the current course vector. A course object would then be created, and the course information would then initialize a new course object, calling the constructor. Once the course object is created, the temporary vector is cleared and the course is appended to the vector of available courses. Each line in the FOR loop is executed once per line in the file being read, and the check to ensure each course is valid iterates over all previous courses in the vector, giving the vector storage file reader and course creator a Big O value of O(NlogN).

The hash table storage system functions very similarly, except when searching for valid prerequisites, the complexity is constant, since hash values map directly to indices within a hash table. When checking if a prerequisite is in the list of available courses, the system simply recomputes the hash value of the course in question, checks the element at that index, and returns whether or not the hash table is empty at that index. Since the hash table prevents the system from having to iterate through all existing courses when checking if a prerequisite is valid, the Big O value of the hash table storage system is O(N).

The binary search tree system is similar again, however, its difference in complexity also comes from its insertion of courses into a BST. When checking for valid prerequisites, the complexity is O(logN), however reading the file in line by line gives the read function a complexity of O(N), so the prerequisite validation will not affect the Big O value. However, inserting courses is more complex in a BST, due to the nature of the structure. A binary search tree sorts data as it is inserted, and therefore, does not require a separate function for sorting the data. This means that an accurate comparison between the binary search tree and the other aforementioned data structures cannot be made, at least when only attempting to compare the complexity of reading in and creating course objects. That said, the complexity of the system using a BST to store and sort data has a Big O value of O(NlogN).