**Run time:**

**Vector:** File Reading Time Complexity: O(n) – The program reads each line in the file.

Course Object Creation Time Complexity: O(n^2) – Nested loops are employed as each course's prerequisites are sought within the courses vector.

Overall Time Complexity: O(n^2) – Combining both steps yields a quadratic time complexity.

**Hash Table:** File Reading Time Complexity: O(n) – The program reads each line in the file.

Course Object Creation Time Complexity: O(n) – New course objects are created and inserted into the hash table.

Overall Time Complexity: O(n) – The dominant linear time complexity remains consistent throughout.

**Tree:** File Reading Time Complexity: O(n) – The program reads each line in the file.

Course Object Creation Time Complexity:

Average Case: O(log n) – On average, course objects are inserted into the tree with logarithmic time complexity.

Worst Case: O(n) – In the worst-case scenario where the tree is unbalanced, the time complexity becomes linear.

Overall Time Complexity:

Average Case: O(n log n) – The average complexity combines file reading and course object creation in a logarithmic manner.

Worst Case: O(n^2) – In the worst-case scenario, the combination yields a quadratic time complexity.

Top of Form

**Advantages and disadvantages of each data structure:**  
**Vector:** Advantages: Offers swift random access to elements and effortless insertion and deletion at the vector's end.

Disadvantages: Exhibits tardy insertion and deletion within the vector's middle, necessitating element shifting for accommodating or filling gaps.

**Hash Table:** Advantages: Excels in rapid insertion, deletion, and retrieval of elements, maintaining an average O(1) time complexity.

Disadvantages: Encounters potential key collisions, potentially impeding operations, and mandates the use of an effective hash function.

**Tree:** Advantages: Facilitates quick insertion, deletion, and retrieval of elements, typically boasting an average O(log n) time complexity.

Disadvantages: Presents the risk of an imbalanced tree, leading to a worst-case time complexity of O(n), and bears the overhead of memory consumption for storing child node pointers.

**Recommendation for which data structure that will pal to use for code:**

Based on the analysis, hash table, where the worst-case running time of inserting n elements into a hash table is O(n), however in order to sort a hash table easily we have to transfer the all of the data to an array which is not ideal, therefore Tree data structure is recommended, Time Complexity with an average time of O(log n), Balanced Performance using self-balancing tree structures, and Search Flexibility in searching for elements compared to vectors, which require linear search, and hash collision, which may face collisions.