**Vector (Array):**

* Reading the file and parsing each line: O(n) - where n is the number of lines in the file
* Creating a course object: O(1) - since it involves simple memory allocation and assignment operations
* Adding the course object to the vector: O(1) - constant time for appending to the end of a vector

Advantages:

* Simple and straightforward implementation
* Constant-time access to elements by index
* Memory-efficient when the number of elements is known and fixed

Disadvantages:

* Insertion and deletion of elements can be costly (O(n)) as it may require shifting elements
* Dynamic resizing of the vector may lead to occasional reallocations and copying of elements, impacting performance

**Hash Table (HashMap):**

* Reading the file and parsing each line: O(n) - where n is the number of lines in the file
* Creating a course object: O(1) - constant time for creating and adding to a hash table
* Adding the course object to the hash table: O(1) - constant time for inserting into a hash table

Advantages:

* Fast insertion, deletion, and retrieval of elements on average (O(1))
* Well-suited for cases where quick access and retrieval of data are essential
* Dynamic resizing typically does not affect the overall performance as much as in vectors

Disadvantages:

* Hashing collisions can occur, degrading performance to O(n) in the worst case
* Slightly higher memory usage due to the overhead of maintaining the hash table structure

**Tree (Binary Search Tree):**

* Reading the file and parsing each line: O(n) - where n is the number of lines in the file
* Creating a course object: O(1) - constant time for creating a node in the tree
* Adding the course object to the tree: O(log n) on average, O(n) in the worst case - due to tree balancing issues

Advantages:

* Ordered data structure, which can be useful for certain applications
* Efficient for searching (O(log n)) when the tree is balanced

Disadvantages:

* Potentially unbalanced trees can lead to poor performance (O(n)) for searching, insertion, and deletion

More complex implementation and maintenance compared to vectors and hash tables

Based on the Big O analysis results and the advantages and disadvantages of each data structure, my recommendation would be to use a Hash Table (HashMap). The reason behind this recommendation is that the Hash Table offers fast insertion, deletion, and retrieval of elements on average (O(1)), making it efficient for handling large amounts of data in the worst-case scenario. While hash collisions can occur, they are typically well-handled by modern hash table implementations, and their performance is usually favorable compared to the other options.

Additionally, the Hash Table has a lower memory overhead compared to the Tree, and it does not suffer from the potential resizing and reallocation issues of the Vector. Its constant-time insertion and retrieval make it a suitable choice for storing and managing course objects, especially if quick access and efficient memory usage are crucial considerations in the application. Hash Table seems like the most appropriate choice for this scenario.