# Runtime Analysis and Memory

Vector Data Structure

1. Reading and Parsing file:
   1. Cost per line:
      1. Opening the file: O(1)
      2. Reading each line: O(n)
      3. Parsing each line and creating course objects: O(n)
      4. Total cost: O(n) for reading and parsing as each line is processed once
   2. Memory usage:
      1. Space to store n course objects: O(n)
2. Runtime Analysis:
   1. Reading the file creation of course objects include processing each line once, resulting in O(n) time complexity.
   2. Overall Time Complexity: O(n)
   3. Overall Space Complexity: O(n) for storing courses in the vector

Hash Table Data Structure

1. Reading and parsing the file:
   1. Cost per Line:
      1. Opening the file: O(1)
      2. Reading each line: O(n)
      3. Parsing each line and creating course objects: O(n)
      4. Inserting each course object into the hash table: O(1) on average per insertion
      5. Total cost: O(n) for reading, parsing , and inserting into the hash table
   2. Memory Usage:
      1. Space to store n course objects plus overhead for the hash table and potential collision handling: O(n)
2. Runtime Analysis:
   1. File Reading and Object Creation: O(n) for reading and parsing
   2. Hash Table Insertion: O(1) on average
   3. Overall Time Complexity: O(n) for reading and parsing plus O(n) for insertions, so overall O(n)
   4. Overall Space Complexity: O(n) considering additional space for hash table overhead.

Tree Data Structure

1. Reading and parsing the file:
   1. Cost per line:
      1. Opening the file: O(1)
      2. Reading each line: O(n)
      3. Parsing each line and creating course objects: O(n)
      4. Inserting each course object into a balanced tree: O(log n) per insertion for balanced trees
      5. Total Cost: O(n log n) for reading, parsing ,and inserting into the tree
   2. Memory Usage:
      1. Space to store n course objects plus overhead for tree nodes and pointers: O(n)
2. Runtime Analysis:
   1. File Reading and Object Creation: Reading and parsing is O(n)
   2. Tree Insertion: Insertion into a balanced tree is O(log n) per insertion and O(n log n) overall
   3. Overall Time Complexity: O(n) for reading and parsing plus O(n log n) for insertion so total of O(n log n)
   4. Overall Space Complexity: O(n) space for the tree nodes and course objects.

Vectors have a time complexity of O(n) for reading, parsing ,and storing. O(n) space complexity. On the other hand, Hash Tables have a time complexity of O(n) for reading, parsing , and insertion. They have a space complexity of O(n) with some overhead. Lastly, Trees have O(n) time complexity for reading and parsing, but O(n log n) for insertion. They also have a space complexity of O(n).

The vector has an O(n) runtime, predictable performance, and simple memory utilization. Fast average performance is offered by the hash table, but in the worst case, performance may suffer. If balanced, the tree(BST) provides logarithmic performance; nevertheless, more complex handling may be necessary to prevent worst-case behavior.