**Runtime & memory**

Vector Big O = O(n\*m)

HashTable Big O = O(n\*m)

BST Big O = O(n \* m + n log n)

A white board with writing on it

Description automatically generated

Image above was how I calculated Big O for each DS that reads a file and creates course objects.

| **Code**  **loadCourses (vector DS)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Vector<Course> courses** | 1 | 1 | 1 |
| **File = Parser(csvPath)** | 1 | n \* m | n \* m |
| **try** | 1 | 1 | 1 |
| **For loop to read all csv rows** | 1 | n | n |
| **Course course** | 1 | 1 | n |
| **Course.courseNum** | 1 | 1 | n |
| **Course.courseName** | 1 | 1 | n |
| **For all courses >= file[i][2]** | 1 | 1 | m |
| **Prereqs.push\_back(file[i][n])** | 1 | 1 | m |
| **Courses.push\_back(course)** | 1 | 1 | n |
| **catch** | 1 | 1 | 1 |
| **Throw error** | 1 | 1 | 1 |
| **Return courses** | 1 | 1 | 1 |
| **Total Cost** | | | 3+n\*m+(4+m)n |
| **Runtime** | | | O(n\*m) |

| **Code**  **loadCourses (Hash Table DS)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **File = Parser(csvPath)** | 1 | n \* m | n \* m |
| **try** | 1 | 1 | 1 |
| **For loop to read all csv rows** | 1 | n | n |
| **Course course** | 1 | 1 | n |
| **Course.courseNum** | 1 | 1 | n |
| **Course.courseName** | 1 | 1 | n |
| **For all courses >= file[i][2]** | 1 | 1 | m |
| **Prereqs.push\_back(file[i][n])** | 1 | 1 | m |
| **HashTable->Insert(course)** | 1 | 1 | n |
| **catch** | 1 | 1 | 1 |
| **Throw error** | 1 | 1 | 1 |
| **Total Cost** | | | n\*m+1+(4+m)n |
| **Runtime** | | | O(n\*m) |

| **Code**  **loadCourses (BST DS)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **File = Parser(csvPath)** | 1 | n \* m | n \* m |
| **try** | 1 | 1 | 1 |
| **For loop to read all csv rows** | 1 | n | n |
| **Course course** | 1 | 1 | n |
| **Course.courseNum** | 1 | 1 | n |
| **Course.courseName** | 1 | 1 | n |
| **For all courses >= file[i][2]** | 1 | 1 | m |
| **Prereqs.push\_back(file[i][n])** | 1 | 1 | m |
| **Bst->Insert(course)** | 1 | 1 | log n |
| **catch** | 1 | 1 | 1 |
| **Throw error** | 1 | 1 | 1 |
| **Total Cost** | | | n\*m+(3+m+log n)n |
| **Runtime** | | | O(n\*m + log n) |

As for the memory usage of each DS, the vector will have to create n new courses and create a vector container to hold all courses; therefore, the vector will have a memory allocation of O(n). The hash table will create n new courses and n new nodes to, one node to hold each course and then specify the next nodes for chaining purposes; therefore, the hash table will be O(n) for its memory allocation. Finally, the BST will have to create n new courses and n new nodes. The nodes are important to keep track of parents and right & left children which means that the memory allocation of the hash table is also O(n). Additionally, each DS will also have to create a vector within each course object to hold a list of prerequisites.