**Project One – Everything Else**

**Runtime Analysis:**

**Vector -- readCourseFile**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| For each line in file | 1 | n | n |
| If line not empty | 1 | n | n |
| Add line to courseLines | 1 | n | n |
| Close file | 1 | 1 | 1 |
| For each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| if parts < 2 | 1 | n | n |
| if parts[0] already seen | 1 | n | n |
| add parts[0] to validCourseNumbers | 1 | n | n |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| for each item starting at index 2 | 1 | n | n |
| if item not in validcourseNumbers | 1 | n | n |
| Total Cost | | | 12n + 2 |
| Runtime | | | O(n) |

**Vector -- createCourseObjects**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| create course object | 1 | n | n |
| set course.courseNumber | 1 | n | n |
| set course.courseName | 1 | n | n |
| for each part starting at index 2 | 1 | n | n |
| add part to course.prerequisites | 1 | n | n |
| append course to vector | 1 | n | n |
| Total Cost | | | 8n |
| Runtime | | | O(n) |

**Hash Table – readCourseFile**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| open file | 1 | 1 | 1 |
| for each line in file | 1 | n | n |
| if line not empty | 1 | n | n |
| add line to courseLines | 1 | n | n |
| close file | 1 | 1 | 1 |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| if parts < 2 | 1 | n | n |
| if parts[0] empty OR parts [1] empty | 1 | n | n |
| if parts[0] already seen | 1 | n | n |
| add parts[0] to validCourseNumbers | 1 | n | n |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| for each item starting at index 2 | 1 | n | n |
| if item empty | 1 | n | n |
| if item not in valid courseNumbers | 1 | n | n |
| Total Cost | | | 14n + 2 |
| Runtime | | | O(n) |

**Hash Table –** creatCourseObjects

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| create Course object | 1 | n | n |
| set course.courseNumber | 1 | n | n |
| set course.courseName | 1 | n | n |
| for each part starting at index 2 | 1 | n | n |
| add part to course.prerequisites | 1 | n | n |
| if key exists in table | 1 | n | n |
| insert course into has table (assuming worst possible) | n | n | n² |
| Total Cost | | | 8n + n² |
| Runtime | | | O(n²) |

**BST – loadCourseFile**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| open file | 1 | 1 | 1 |
| for each line in file | 1 | n | n |
| if line not empty | 1 | n | n |
| add line to courseLines | 1 | n | n |
| close file | 1 | 1 | 1 |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| if parts < 2 | 1 | n | n |
| add parts[0] to validCourseNumbers | 1 | n | n |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| for each item starting at index 2 | 1 | n | n |
| if item not in validCourseNumbers | 1 | n | n |
| Total Cost | | | 11n+2 |
| Runtime | | | O(n) |

**BST -- createCourseTree**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| for each line in courseLines | 1 | n | n |
| split line by commas | 1 | n | n |
| create course object | 1 | n | n |
| set course.courseNumber | 1 | n | n |
| set course.courseName | 1 | n | n |
| for each part starting at index 2 | 1 | n | n |
| add part to course.prerequisites | 1 | n | n |
| insert course into BST by key (again, assuming the worst) | n | n | n² |
| Total Cost | | | 7n + n² |
| Runtime | | | O(n²) |

**Evaluation:**

In the scenario, the advisors requested two tasks: print all CS courses in alphanumeric order, and print a given course’s title and prerequisites. After writing out the pseudocode for each and completing the runtime analysis, each data type has its advantages and disadvantages. A vector is easy to build, but has no inherent order, so printing the full list needs a sort, and finding one course is a linear scan. A hash table is great for single lookups, but also isn’t ordered. Also, in the worse cases, inserts can approach O(n²) due to collisions. A BST fits the ordered list requirement naturally. The in-order traversal prints without needing to be sorted ahead of time, and a BST’s single item search is O(log n) on average. It too has a downside, however, as the speed of insertion and lookups of single courses could be bad if the table is unbalanced.

Ultimately, I recommend the BST, as it eliminates the extra sorting, and on average the single course insertions and lookups is still pretty efficient.