|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | #Times Executes | Total Cost |
| Open input file | 1 | 1 | 1 |
| For each line of input file | 1 | N | N |
| Create vector | 1 | 1 | 1 |
| Create vector course item | 1 | N | N |
| For each prereq | 1 | N | N |
| PrintAllCourseInformation()  For all courses | 1 | N | N |
| For each prereq | 1 | N | N |
| Findcourse  For each course in vector | 1 | N | N |
|  |  | Total Cost | 6n+2 |
|  |  | Runtime | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Hashtable | Line Cost | #Times executes | Total Cost |
| Open input file | 1 | 1 | 1 |
| Create hash table | 1 | 1 | 1 |
| For each line input file | 1 | N | N |
| Create key for hashtable | 1 | N | N |
| Add course to hashtable with courseNumber as key | 1 | N | N |
| Validate Prereqs  For each prereq in prereq | 1 | N | N |
| Printcourseinformation() | 1 | 1 | 1 |
| DisplayCourses()  Sort | 1 | N | N |
| For each courseNumber in courseNumbers | 1 | N | N |
| Sort(courseNumbers)  For I from 0 to n-1 | 1 | N | N |
| For j from 0 to n-i-1 | 1 | N | N |
| GetKeysFromHashTable()  For each entry in Hashtable | 1 | N | N |
| Append entry.key to keys | 1 | N | N |
|  |  | Total cost | 10n+3 |
|  |  | Run time | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Tree | Line cost | # times executes | Total cost |
| Open input file | 1 | 1 | 1 |
| Create Binary Tree | 1 | 1 | 1 |
| Read Lines of input file | 1 | N | N |
| Add course to BST using course number as key | 1 | N | N |
| Create a new course object | 1 | 1 | 1 |
| Course injection into bst | 1 | N | N |
| validatePrerequsities  for each prereq in prereq | 1 | N | N |
| searchBST | 1 | N | N |
| printCourseInformation | 1 | N | N |
| InOrder | 1 | N | N |
|  |  | Total cost | 7n+2 |
|  |  | Runtime | O(n) |

Vector:

Vectors are typically easy to implement within a project. It has a low memory usage and is very efficient at a low to medium number of data sets. The vector datatype has contiguous memory allocation to store elements. This leads to better performance when reading files and accessing portions of the vector. The random access of a vector allows a program to be very efficient when trying to access multiple portions of the vector without having to traverse through the entire thing. Vectors also include dynamic resizing which is a very large advantage compared to the other datatypes. The main disadvantage of a vector is the inefficiency of the insertions or deletions. It requires shifting elements to maintain order and impacting the performance for especially large subsets of data.

Hash table:

Hash tables are renowned for their fast retrieval of data. They are highly efficient for lookup operations as they use a hash function to compute an index directly from a hash key. This is particularly useful in a very large dataset. They are particularly well suited for an associative data structures, this in turn allows for very efficient insertion, deletion, and search operations. The disadvantages are that they typically use significant amounts of memory and might be intensive when working with large data sets. When working with hash tables, a good programmer needs to remember proper collision management as well. This is where multiple keys hash to the same index and require additional data structures like a linked list. This has ramifications in the form of significant performance issues if not addressed properly.

Binary Search Tree:

The BST facilitates efficient in order traversal allowing for easy access to elements in a sorted manner. They are dynamically resized with the addition or removal of nodes which provides flexibility in handling various datasets and types without the need for resizing maintenance operations. The tradeoff is that they are very memory intensive where each node in a BST typically requires extra memory for pointers to its child nodes. In a worst case scenario, a BST might become “unbalanced”. To remedy this, balancing operations would need to be implemented for maintenance as a tree grows and changes.

For this particular project, I would recommend going with the vector datatype. This datatype excels where there is a small dataset. Seeing as the course list is only about 150, the overhead memory and performance would be the best using this datatype. It is the simplest to implement and change.