| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **ReadLines(); -vector-** | 1 | n | n |
| **ParseFile();** | 1 | 1 | 1 |
| **ReadFile();** | 1 | 1 | 1 |
| **PrintMenu();** | 1 | 1 | 1 |
| **PrintSpecifiedCourse();** | 1 | n | n |
| **OrderList(); -vector-** | 1 | n log(n) | n log(n) |
| **Total Cost** | | | 3n \* log(n) +3 |
| **Runtime** | | | n log(n) |

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **ReadLines();-HashTable-** | 1 | 1 | 1 |
| **ParseFile();** | 1 | 1 | 1 |
| **ReadFile();** | 1 | 1 | 1 |
| **PrintMenu();** | 1 | 1 | 1 |
| **PrintSpecifiedCourse();** | 1 | 1 | 1 |
| **OrderList(); -HashTable-** | 1 | n log(n) | n log(n) |
| **Total Cost** | | | n \* log(n) +6 |
| **Runtime** | | | n log(n) |

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **ReadLines(); -BST-** | 1 | log(n) | log(n) |
| **ParseFile();** | 1 | 1 | 1 |
| **ReadFile();** | 1 | 1 | 1 |
| **PrintMenu();** | 1 | 1 | 1 |
| **PrintSpecifiedCourse();-BST-** | 1 | log(n) | log(n) |
| **OrderList(); -BST-** | 1 | n log(n) | n log(n) |
| **Total Cost** | | | n \* 3log(n) +3 |
| **Runtime** | | | n log(n) |

Evaluation

Vector:

The main advantage with this data structure is its ease of implementation. Vector sorting would follow the quick sort algorithm and would be n log(n) time at an average which would leave the sorting at a decent complexity. For the search, the time would follow n time at an average as it would have to iterate through the vector to find the desired course. It’s memory usage will be average but it trades this for n complexity for that n search time.

HashTable:

The main advantage of a hash table is its search time. With a good hashing function and minimal collisions, search can be extremely fast. However, the cost of this is memory usage is the trade off for the fast access and search. Since a hash table would be unordered, we would need to dump it into a vector, but we can still utilize a quicksort algorithm to order it which would leave the complexity the same as the vector here.

Binary Search Tree:

The main advantage of a BST is its logarithmic creation and search. Its ordering time complexity would be equivalent to a quick sorting algorithm, n log(n), thus losing no complexity but not gaining any either. With this in mind, the main trade off is complexity of implementation. This however, is a worthy trade off as its blazing search and insert speed is very desirable.

Conclusion:

If I were to make a general recommendation, I would recommend the BST as it ticks off speed, size, and complexity requirements for a good general use data structure. With this said, if memory usage is not a concern and a very good hashing function is available, a hash table can beat out the BST in speed of insertion and search. If we look at their time complexities only, we would have logarithmic functions for search and insertion in addition to an n log(n) function for sorting with the BST and with the hash table, we have only the n log(n) sort as a major impact for time complexity. The main issue with a hash table is its space, so if that were not a factor in the program, hash table all the way!