Run Time Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | Time Executes | Total |
| Create Vector | 1 | 1 | 1 |
| Each line | 1 | n | n |
| Create vector | 1 | n | N |
| While prereq | 1 | n | n |
| Append prereq | 1 | N | N |
| Push back | 1 | N | N |
| Total | | | 5n+1 |
| runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Hash Table | Line Cost | # executes | Total |
| Create Tale | 1 | 1 | 1 |
| Insert | 0 | 0 | 0 |
| Create Key | 1 | n | N |
| If no entry found | 1 | n | n |
| Assign node | 1 | n | N |
| Else | 1 | n | N |
| Assign old node | 4 | n | 4n |
| Else | 1 | n | N |
| Find next node | 1 | n | N |
| Add new node | 1 | n | N |
| Each new line | 1 | n | N |
| Create vector | 1 | n | N |
| While prereq | 1 | n | N |
| Append prereq | 1 | n | N |
| Insert item | 1 | n | N |
| Total | | | 16n+1 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Tree | Line cost | # executes | Total |
| Add node | 0 | 0 | 0 |
| If null add | 1 | 1 | 1 |
| If node is less | 1 | n | n |
| No left node | 1 | n | n |
| Node becomes left | 1 | n | n |
| Node is greater | 1 | n | n |
| No right node | 1 | n | n |
| Node becomes right | 1 | n | n |
| Each line | 1 | n | n |
| Create vector | 1 | n | n |
| While prereq | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert item | 1 | n | n |
| Total | | | 11n+2 |
| Runtime | | | O(n) |

Vectors have the advantage of being the quickest method for reading and adding objects. But it has the disadvantage of search the list for a specific object. It had the quickest runtime.

Hashtables can search a list fast but is not the best at creating the initial list or allow the table itself to be sorted.

Binary Trees are a mix between hash tables and vectors, in that its can be quicker than a vector but not as easy to implement as a hash table.