**Data Structures/Implementation**

|  |  |  |
| --- | --- | --- |
| **Data Structure/Implementation** | **Alternative** | **Why alternative was rejected** |
| Adjacency List to store graph vertices and indegrees | Adjacency Matrix | Preference. Adjacency list was easier to implement and update indegree of vertices using updateKey and using vector iterators. |
| Priority Queue to sort vertices | Trees | Trees do not have enqueue and dequeue functionality or First In First Out functionality. In a priority queue, you can directly access the front element. You can also pop the front element easily. This functionality is necessary to have to achieve efficient topological sorting. |
| Using a class to implement TopologicalSort | Using a struct | More efficiency, flexibility, and it makes more sense to use a class rather than a struct for a complex problem such as this. |
| Directed Graph | Weighted Graph | Unnecessary to have a weight. We are simply checking for and updating indegree. Directed graph accomplishes this just fine. |

**Time/Space Complexity Analysis**

|  |  |  |
| --- | --- | --- |
| **Function** | **Time/Comment** | **Space/Comment** |
| int main() | O(nn)  Where nn represents the time complexity of the TopologicalSort::sort() function, which is the function with worst time complexity | O(nm)  Where nm represents the space complexity of the TopologicalSort::sort() function, which is the function with worst space complexity |
| TopologicalSort::updateKey(int x, vector<int> inDegree) | O(n)  Where n represents the number of adjacent vertices to vertice x. In EVERY case, the function will loop through all adjacent vertices of vertex x to update its indegree. | O(1)  No new space is allocated |
| TopologicalSort::TopologicalSort(int numVertices) | O(1)  Only 2 instructions are carried out | O(n)  Where n represents the total number of vertices (numVertices). The constructor will allocate space for an adjacency list with n elements |
| TopologicalSort::addEdge(int vertice, int value) | O(1)  Only 1 instruction is carried out | O(1)  The function will at most append 1 adjacent vertice to the appropriate vertex in the adjacency list |
| TopologicalSort::sort() | O(nn)  Where n represents every element in the adjacency matrix. This function will at worst loop through the entire adjacency matrix n times. This is because the indegree of every vertex and its adjacent vertices has to be updated/checked after a vertex has been visited, then rechecked every time a vertice’s indegree is updated to ensure the program sorts lexicographically and topologically. This time complexity can be observed in the while() loop at line 91 and the for loop inside the while loop at line 101 | O(nm)  Where n represents the total number of vertices, and m represents the adjacent vertices to n. The worst space complexity is observed when the priority queue is being populated. |