**Report**

**Summary:**

Topological sorting is a linear ordering of vertices in a directed acyclic graph (DAG). Topological order is not possible if the graph is acyclic c. If the number of vertices with in-degree 0 is less than the number of vertices in the graph, then the graph is not a DAG( Directed Acyclic Graph). It helps solve problems involving dependencies and precedence, such as scheduling tasks or building systems.

**Data structures used:**

1.Array lists

2. Hash maps

3.Priority queue

**Time and space complexity analysis:**

The time complexity of the find the **Topological Order** of a graph in my program is O(V + E), where V is the number of vertices and E is the number of edges in the graph and the method used to fin the topological order is findTopologicalOrder() which is one of the public methods in the Graph class.

1.The first step is initializing the priority queue, I iterated over all vertices (V) to find vertices with in-degrees of 0 and add them to the priority queue. This operation takes O(V) time.

2. Topological Sorting Loop: O(V + E)

- using a while loop I iterated through all vertices in the graph. (In the worst case, it may processes each vertex and its outgoing edges exactly once)

- For each vertex processed, I removed it from the priority queue in constant time

- For each edge (u, v), I updated the in-degree of vertex v and also at the same time I added it to the queue if its in-degree becomes 0.

- The total time spent inside this loop is O(E), where E is the number of edges in the graph.

- Since each vertex and edge is processed once, the overall time complexity of this loop is O(V + E).

3. After the loop, I checked if the size of the `topologicalOrder` list is less than the number of vertices (V). If yes, I printed a message indicating that the graph contains a cycle.

- This checking operation took O(V) time as I iterated over the `topologicalOrder` list.

So, the final time complexity for this algorithm that I implemented was O(V)+ O(V + E)+O(V) which equals to O(V + E) (because it’s the dominant one) .

**Space complexity:**

**verticesList:** This ArrayList stores the vertices of the graph. In the worst case, it can contain all V vertices (where V is the number of vertices in the graph). So, the space complexity for verticesListi O(V).

**AdjacencyList:** This ArrayList of ArrayLists represents the adjacency list. Each vertex has its own list of neighbors, and each edge is represented once in one of these lists. In the worst case, if all vertices have many neighbors and there are E edges in the graph, the space complexity for adjacencyList is O(V + E), where V is the number of vertices, and E is the number of edges in the graph.

**IndegreeMap:** This HashMap stores the indegree of each vertex. In the worst case, it can have entries for all V vertices. So, the space complexity for indegreeMap is O(V).

So, the overall space complexity is O(V+E) since it is the dominant of all.

**Summary of important methods of Graph class:**

* **findTopologicalOrder():**
* The `findTopologicalOrder()` method determines the topological order of vertices in a directed graph represented as an adjacency list. I created a priority queue to identify vertices with in-degrees of 0, indicating starting points in the graph. The method then performs topological sorting, removing vertices and updating in-degrees of their neighbors until the entire order is determined. If the graph contains cycles, a message is displayed, and an empty list is returned. Ultimately, the method returns the topological order as a list, provided that it's a valid directed acyclic graph.
* **Graph():**The graph constructor will initialize  verticesList,adjacencyList,indegreeMap.
* **addVertex(int vertex):** addVertex method add the vertex to the vertx list,adjecency listr and indegree map.
* **addEdge method**: This method will add an edge between two vertices.