# PA3 Report

Throughout programming assignment 3 the goal was to construct a graph data structure and then perform a topological sort on it. The data is assumed to be in the order given by the input files given for testing. Two classes were used throughout this assignment, the vertex class and graph class. The vertex class consists of some constructors, and the objects of type int; label, indegree, and top_num. The graph class consists of 3 objects two are vector of vertices called node_list and top_sorted, and the last is a vector of a list of ints called adj_list. The node_list vector holds all the vertices, adjacency list holds all the edges going out from the vertices, and the top_sorted holds the sorted vertices in topological order. 5 functions are also in the graph class, they are buid_graph, display_graph, topological_sort, compute_indegree, and print_top_sort. The running times for build_graph is O(|E| + |V|), where E is the number of edges and V is the number of vertices. Build_graph uses two while loops, one to grab the vertex, than one to grab the adjacency of that vertex, it stores the edges in a temp list and then it pushes the list into the adjacency list.. Display_graph has the same runtime has build_graph which is O(|E| + |V|), it displays the vertex than the corresponding adjacency list using two for loops. The topological_sort function has a running time complexity of O(|E| + |V|). The topological_sort function follows the pseudocode for the textbook, where first each vertex with an indegree of 0 is added to a queue, then the vertex is popped from the que and all vertexes adjacent to that initial vertex that have indegree – 1 = 0 are added to the queue, and the it selects the next vertex in the queue and repeats until all vertexes and edges have been checked. The compute_indegree function has a runtime of O(|E| + |V|) and follows the pseudocode similar to the textbook, where all vertices are initially assigned an indegree of 0 and then a vertex is selected and then the vertices adjacent to the initially selected vertex's indegrees are incremented by 1, this is repeated for all vertices. The print_top_sort function has a runtime of O(|V|), it runs through a vector of the topological sorted graph and prints it. The algorithm uses a queue because it is a first-in-first-out data structure, a stack could also be used if you enter in the data backwards. For example if you put 1, 2, and 3 into a queue in that order, then 1, 2, and 3 will be removed in that order, but if you put in 1, 2, and 3 into the stack they will be removed in the order 3, 2, and 1. So to use a stack you must enter the data like so 3, 2, and 1 so that it can be removed in the correct order 1, 2, and 3. The algorithm detects cycles by using a counter that increments every time a vertex is popped from the queue, if the counter is not equivalent to the number of vertices than a cycle has been found. The following are what I used to test my code for correctness.

## Case 1:

**Input:**

1 2 4 5 -1

2 3 4 7 -1

3 4 -1

4 6 7 -1

5 -1

6 5 -1

7 6 -1

**Output:**

```
1: 2 4 5
2: 3 4 7
3: 4
4: 6 7
5:
6: 5
7: 6
After topological sort the order is : 1 2 3 4 7 6 5
```

## Case 2:

**Input:**

1 3 -1

2 3 8 -1

3 4 5 6 8 -1

4 7 -1

5 7 -1

6 -1

7 -1

8 -1

**Output:**

```
1: 3
2: 3 8
3: 4 5 6 8
4: 7
5: 7
6:
7:
8:
After topological sort the order is : 1 2 3 4 5 6 8 7
```

## Case 3:

**Input:**

1 2 4 -1

2 5 7 8 -1

3 8 6 -1

4 5 -1

5 9 6 -1

6 -1

7 -1

8 -1

9 -1

**Output:**

```
1: 2 4
2: 5 7 8
3: 8 6
4: 5
5: 9 6
6:
7:
8:
9:
After topological sort the order is : 1 3 2 4 7 8 5 9 6
```

## Case 4:

**Input:**

1 2 4 5 -1

2 3 4 7 -1

3 4 -1

4 6 7 -1

5 4 -1

6 5 -1

7 6 -1

**Output:**

```
1: 2 4 5
2: 3 4 7
3: 4
4: 6 7
5: 4
6: 5
7: 6
A cycle has been found.
```