1. **Data structures used for Q and visited. Your rationale behind the choice of data structures.**
   a) Use a LinkedList for Q because it supports push and pop operation in O(1) time.
   b) Use a HashSet for visited because it supports add and search operation in O(log(n)) time where n is the number of vertices.

2. **Number of edges and vertices in the graph WikiCS.txt.**
   a) Number of edges: 23869
   b) Number of vertices: 500

3. **Number of strongly connected components in WikiCS.txt**
   9

4. **Size of the largest component in WikiCS.txt.**
   492

5. **The data structures that you built/used in GraphProcessor.**
   LinkedHashMap<String, LinkedList<String>> G;
   For every entry (u, [v1,v2,..,vn]) in G, <u, vi> is an edge in the graph.

6. **Analyze and report the asymptotic run time for each of the class GraphProcessor.**

Let N be the number of vertices in the graph, and
E be the number of edges in the graph, and
C be the number of SCCs in the graph.

SCCAlgo:

```
T = O (
      E*log(N) + N*log(N)     // computeOrderAlgo
    + N*log(N)                // sort V by finish time
    + N*log(N)                // sccDFS
) = O( (N+E)*log(N) )
```

outDegree:

T = O(log(N)) // find key in a map.

sameComponent:

T = O (

(N + E)*log(N) // SCC Algo

+ C // loop for components

* log(N) // find v in the component set

) = O((N+E+C)*log(N))

componentVertices:

T = O (

(N + E)*log(N) // SCC Algo

+ C // loop for components

* (log(N) // find v in the component set

+ N) // create list from a set

) = O((N+E)*log(N) + CN))

largestComponent:

T = O (

(N + E)*log(N) // SCC Algo

+ C // loop for components

) = O((N+E)*log(N)+C)

numComponents:

T = O((N+E)*log(N)+C) // same as SCC Algo

bfsPath:

T = O (

N // traverse through n vertices

* log(N) // find node in the visited set

+ N // build the path

) = O(N*log(N))