Pseudocode and Analysis of the Greedy Algorithm for the Minimum Dominating Set problem
CS:3330, Spring 2017, Sriram Pemmaraju

(a) The greedy algorithm in Problem 3 with input adjacency list can be implemented in the following way:

\begin{algorithm}
\caption{Dominate(L)}
\begin{algorithmic}[1]
\State Set nonblack be an empty object to host non-black vertices
\State Let \textbf{ds} be an empty set for hosting the dominating set
\State Let \textbf{color} be a length-\(n\) array, all of whose slots are initialized to white
\For {each vertex \(i\) in the graph}
\State nonblack.insert\((i, L[i].\text{length}+1)\)
\EndFor
\State \((v, \text{whiteDeg}) \leftarrow \text{nonblack}.\text{getmax}()\)
\While {\text{whiteDeg} > 0}
\State Save \(v\) to \textbf{ds}
\If {\text{color}[[v]] == \text{white}}
\For {each neighbor \(j\) of vertex \(v\)}
\State nonblack.decreaseValue\((j, 1)\)
\EndFor
\EndIf
\For {each neighbor \(j\) of vertex \(v\)}
\If {\text{color}[[j]] == \text{white}}
\For {each neighbor \(k\) of vertex \(j\)}
\State nonblack.decreaseValue\((k, 1)\)
\EndFor
\State color[[j]] ← \text{gray}
\EndIf
\EndFor
\State \text{color}[[v]] ← \text{black}
\State \((v, \text{whiteDeg}) \leftarrow \text{nonblack}.\text{getmax}()\)
\EndWhile
\State \textbf{return} ds
\end{algorithmic}
\end{algorithm}

(b) Given the running time of the 3 methods, \texttt{getMax}, \texttt{insert}, and \texttt{decreaseValue}, we can analyze the algorithm’s running time complexity as follows. The for-loop (Lines 4-6) executes \texttt{insert}(\(k, v\)) \(n\) times, taking \(O(\log n)\) time for each insertion. Thus, this for-loop will run in \(O(n \log n)\) time. The while-loop is executed \(n\) times because with each execution, one vertex is deleted from \texttt{nonblack}. Each execution of \texttt{getmax}, takes \(O(\log n)\) time and therefore extracting vertices with largest white neighborhood from \texttt{nonblack} take \(O(n \log n)\) time. After a vertex \(v\) is extracted from \texttt{nonblack} and added to \textbf{ds}, we have to update white neighborhood sizes associated with vertices in nonblack. Now note that for each vertex that changes from white to gray or black, we update its neighbors’ values in \texttt{nonblack}. A vertex changes from white to some other color only once and therefore for each edge we perform this update at most twice. Updates of these values (via \texttt{decreaseValue}) take \(O(\log n)\) time. Thus the total time to update sizes of white neighborhood sizes is \(O(m \log n)\). Thus the total running time of this algorithm is \(O((m + n) \log n)\).

(c) The data structure that can fulfill the runtime specifications is a \textit{max-heap} implementation of a priority queue.