Please enter your name and uID below.

Name: Jason Crandall

uID: u0726408

Submission notes

- **(PS 1 specific) This is a warm-up assignment. Full points will be given for answers that have some correct elements and that clearly been given a legitimate effort.**

- *Solutions must be typeset* using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the semester, so please retain the original files.
- Please remember that for problem sets, collaboration with other students must be limited to a high-level discussion of solution strategies. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1.

To start off with a new algorithm for topological sort filling from right to left begins with the first node of the algorithm. A simple way of determining a valid start point comes from the in-degree of the first node which will be 0, as no other nodes will be going into it. Thus, if a node has no other nodes going into it, it can be listed as a valid starting point. In order to find a node like this, during a graph construction, it would be best to not only have a structure containing a node and the nodes it is directed to, but also a parallel structure containing a node and all the nodes directed to it. If the list of nodes directed to it are empty, it is a valid start position.

Knowing what the first node in the array will be, the algorithm can proceed as follows: For the overlapping method, it will first mark all nodes as “New”. Next it will start with the initial position being zero in the array building left to right. Next it will start the TopSortDFS function which will take in the node (starting with the initial one that we found) followed by the position.

For the recursive TopSortDFS function, the overall logic of the algorithm is to traverse, starting from the first node down the graph. Once it reaches a node, it checks to see if any of its parents have been visited, and if not, then it will go up that path. When a node’s parents have all been entered into the sorted array, then the existing node will be added, marked, and traverse down the graph to the next node. Essentially this will prioritize adding parent nodes before adding child nodes.

2.

For the algorithm described on the previous page, there could arise the situation where a node and edge is visited twice, once to see if it has unmarked parents to visit, and again once those parents have been added. Thus, a reasonable upper bound complexity of this algorithm would be O(2(|V| +|E|)). The total number of vertices plus the number of edges times 2.

3.

One way of solving this problem is to use a variant of topological sort. Rather than using DFS to order the graph, one can use BFS and a queue to begin ordering the graph. By using breadth first search and a queue, as we begin ordering the graph we add all of the child nodes of the level of the graph into the queue. Knowing this, if after adding the necessary components to the sorted array, there exists only 1 node left in the queue, that node must be a cut node. The algorithm would then proceed as usual. Doing so with this technique would solve the algorithm with an upper bound of $O(|V|+|E|)$ as such is the case of topological sort, only needing to visit each edge and node once.