Correctness Proof: 20 points

Prove the following loop invariant for the algorithm on the previous page:

Here is an algorithm that, given a graph with vertices

Analyzing loops-20pts

Order questions- 10 points each For each, answer True or False, and give a short explanation for your answer.

1. \( n^2 \in O(n^5) \).
   Yes. \( n^2 \leq n^5 \) for \( n \geq 1 \), so the claim follows using \( c = 1 \) in the definition of \( O \).

2. \( n^2 \in \Theta(n^5) \).
   False. If \( n^2 \) were \( \Theta(n^5) \), we would have \( n^2 \geq cn^5 \) for some \( c > 0 \) and sufficiently large \( n \). But then \( c \leq 1/n^3 \), which goes to 0, a contradiction.

3. \( 2^{\log n} \in O(2^{\log n}) \).
   False. \( 2^{\log n} = 2\log n^2 = n^2 \), whereas \( 2^{\log n} = n \). As above \( n^2 \) is not in \( O(n) \).

4. If \( f \), \( g \) and \( h \) are functions from positive integers to positive integers, and \( f(n) \in O(g(n)) \), then \( f(n) + h(n) \in O(g(n) + h(n)) \).
   True. We know \( f(n) \leq cg(n) \) for some \( c > 0 \) and sufficiently large \( n \). Then for sufficiently large \( n \), \( f(n) + h(n) \leq cg(n) + h(n) \leq c'(g(n) + h(n)) \) where \( c' = \max(c, 1) \). Thus, by definition of \( O \) \( f(n) + h(n) \in O(g(n) + h(n)) \).

Analyzing loops-20pts Here is an algorithm that, given a graph with vertices \( V = \{1...n\} \) in adjacency list format, computes the in-degree \( D(u) \) for each node \( u \in V \), the number of edges of the form \((v, u)\). As usual, \( N(u) \) represents the list of out-neighbors of the node \( u \).

\[
\text{InDegrees(G)}
\]

1. Initialize an array \( D \) by \( D[u] \leftarrow 0 \) for each \( u \in V \).
2. FOR each \( v \in V \) do:
3. FOR each \( u \in N(v) \) do: \( D[u] ++ \).
4. Return \( D \)

Give a time analysis, up to order, for this algorithm, in terms of the number of nodes and edges, \( n = |V| \) and \( m = |E| \). Be sure to explain your answer.

Let \( d(v) = |N(v)| \) be the outdegree of \( v \). For each \( v \), when we run through the outside FOR loop with \( v \), we take \( O(d(v)) \) time in the inside FOR loop. Thus, the total time for all executions of the inside FOR loop is \( \sum_v d(v) = |E| = m \), the number of edges. Then the time to initialize the array is \( O(n) \), since we initialize one array position for each \( v \). Thus, the total time is \( O(n + m) \).

Correctness Proof: 20 points Prove the following loop invariant for the algorithm on the previous page:

Then after the iteration with \( v = i \), each \( D[u] \) is the number of in-neighbors of \( u \) occurring earlier, i.e., the number of \( 1 \leq j \leq i \) with \((j, u) \in E \).

We’ll prove that \( D[u] \) is the number of in-neighbors of \( u \) \( j \) with \( j \leq i \) after \( i \) iterations. The base case is \( i = 0 \), i.e., before the loop starts. \( D[u] = 0 \) initially, and there are no vertices \( j \leq 0 \), so \( D[u] = 0 \) is the number of such \( j \).

Assume that after \( i \) iterations , \( D[u] \) is the number of in-neighbors \( j \) of \( u \) with \( j \leq i \). Then if \( i + 1 \) is an in-neighbor of \( u \), the number of neighbors with \( j \leq i + 1 \) is the number of neighbors \( j \leq i \) plus one (for \( j = i + 1 \). In this case \( D[u] \) is incremented, so it is equal to one plus its previous value which is one plus the number of neighbors \( j \leq i \) which is the number of in-neighbors \( j \leq i + 1 \).

If \( i + 1 \) is not an in-neighbor of \( u \), the number of in-neighbors \( j \leq i + 1 \) is equal to the number with \( j \leq i \), and \( D[u] \) is unchanged. So by the induction assumption, it is still equal to the number of in-neighbors \( j \leq i \), which equals the number with \( j \leq i + 1 \). So in either case, the conclusion still holds after \( i + 1 \) iterations.

By induction, it thus holds after any number \( i \) iterations.