Solutions and grading standards

Problem 0 (1 point)

You could have lost the point for this problem for several reasons:

you earned some credit on a problem and did not put your name on the page;
you did not indicate your discussion section and t.a.;
you did not indicate your login on the EECS instructional systems;
you did not identify where you were sitting.

The reason for this apparent harshness is that we need ways to recover misplaced or unstapled. We also need to know where you will expect to get your exam returned, and how to submit your grade to the grading system.

Problem 1 (3 points)

This problem involved comparing the recurrences

$$S(n) = 2S\left(\frac{n}{4}\right) + 2n^2\sqrt{n}$$

$$T(n) = 9T\left(\frac{n}{2}\right) + 3n^2$$

Both recurrences were most straightforwardly solved via the Master method. This would work as follows. For S, a = 2 and b = 4. $\log_b a = \frac{1}{2}$. Thus we want to compare $2n^{2.5}$ with $n^{1/2}$. Case 3 applies, so $S(n) = \Theta(n^{5/2})$.

For T, a = 9 and b = 2. $\log_b a = 3+\varepsilon$. Thus we want to compare $3n^2$ with $n^{3+\varepsilon}$. Case 1 applies, so $T(n) = \Theta(n^{3+\varepsilon})$. T dominates S. Thus $S = O(T)$.

The scoring roughly broke down to 1 point per application of the Master method and 1 point for circling the right containments *consistent with the results obtained from the Master method*. Thus you may have gotten both applications of the Master method incorrect and gotten the wrong containment as well, while still scoring 1 point for consistent claims. Likewise, solutions that specified the correct containment that was inconsistent with their applications of the Master method did not receive that point.

Generally, students did well on this problem. Some students applied the Master method incorrectly, and some more overlooked opportunities to simplify gross expressions involving powers and logs. A few students circled inconsistent sets of statements such as $f = O(g)$, $f = \Theta(g)$, but not $f = \Omega(g)$.

Problem 2 (6 points)

The following questions were to be answered. Is DAG-longest-path in NP? Does DAG-longest-path reduce to Longest-path? If DAG-longest-path were in NP and it were reducible to Longest-path, would it then be NP-complete? The answers are

a. Yes. A solution to DAG-longest-path can produce a certificate—the path itself—that can be easily tested in the same way that a solution to Longest-path can. This is by checking that there are at least k vertices in the path and that each edge in the path is also in G. (With Longest-path, one would also need to test that all the vertices in the path are different; in a DAG, however, there are no directed cycles, so the path won't loop back on itself.)

   An alternative approach is to cite the polynomial-time algorithm for finding a longest path in a DAG (CLR, page 538; Readings, section 20) to show that DAG-longest-path is in P, and to note that every problem in P is also in NP.

b. Yes. Longest-path, given the DAG (no processing is needed to transform it), can be used as a subroutine to determine the existence of a longest path in the DAG. Alternatively, since Longest-path is NP-complete, all problems in NP reduce to it; since DAG-longest-path is in NP by part a, it also reduces to Longest-path.

c. No. Longest-path (a known NP-complete question) has to be reducible to DAG-longest-path to show the NP-completeness of DAG-longest-path. (You might also have answered "maybe", but you had to say something about reduction in the wrong direction.

All three questions could earn 2 points, 1 for the answer and 1 for the explanation. In part a, if you listed the features to be tested, you needed both of them (path length plus inclusion in G) to get the explanation point. In part c, you got the explanation point for anything approximating "not the other way around". You could also get the explanation point if you stated the reduction criteria for determining NP-completeness and said how DAG-longest-path and Longest-path fit into those criteria; with only the definition, you didn't get this point.

The most frequent error in part a was to forget to check that edges in the certificate path are also in G. Some people seemed not to understand that $P \subseteq NP$, claiming that the existence of a polynomial-time algorithm for DAG-longest-path meant that it was not in NP. Some of you also tried to invent a polynomial-time algorithm by using Dijkstra's algorithm on a graph with negative edges, or modifying it to invert the direction of comparisons. This doesn't work because Dijkstra's algorithm can't handle negative-weight edges; DAG-shortest-paths or the Bellman-Ford algorithm must be used in this situation.

In part b, a number of students got the special-case logic backward, saying for example that a DAG can't be converted to a graph with cycles or vice-versa. Another instance of this confusion was the attempt to remove some edges to make a graph with cycles into a DAG (e.g. by doing DFS and removing back edges). Others said this would help but that it can't be done in polynomial time. Finally, some students claimed that since Longest-Path worked on undirected graphs, the two algorithms would be incompatible; however, the exam specifically noted that we were referring to the directed graph version as LongestPath

Some of you got the direction of reduction wrong, causing you to get no for part b and yes for part c. This mistake, it seems, is not quite the same as that of the misguided 170 student of the problem statement. That student apparently understood the definition of reduction, and so was able to correctly determine that DAG-Longest-Path reduces to Longest-Path, but was mistaken about what direction of reduction is required for NP-completeness. If you made the mistake just described, it would seem that you misunderstood the definition of reduction.

Problem 3 (8 points)

This problem involved first finding an example, then characterizing all graphs that contain vertices u and v that satisfy the following property.

> Depth-first search results in u's finishing value being greater than v's, no matter how the vertices of G are ordered.

Graphs that satisfy the given property are exactly those that contain vertices x and y where there is a path from x to y but not from y to x. (Note that x and y are not necessarily the u and v mentioned in the property.) Equivalently, this is the set of graphs that contain at least two strongly connected components A and B and a path from some vertex x in A to some vertex y in B. Here's the proof.

a. All such graphs have the property. Consider a graph with a path $x = x_0, x_1, \ldots, x_m = y$ from x to y. Let v be y. Let $u = x_k$ be the last vertex along the path for which a path exists from u to x. If there is no such u, let u be x. Note that there is no path from v to u since there is no path from v to x, and following the path from v to u, then from u to x produces a path from v to x. Note also that u is in a different strongly connected component from v.

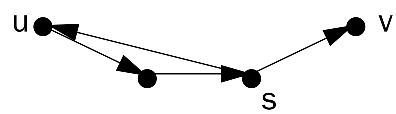
   Now, consider the first vertex on the path from u to v to be encountered in a depth-first search.
   If that vertex is u, the rest of the path from u to v consists solely of white vertices, so v is a descendant of u in the depth-first tree by CLR Theorem 23.8. By CLR Theorem 23.6, f[v] < f[u].
   Otherwise, let s be that vertex. By Theorem 23.6, it must be the case that [d[u],[f[u]] and [d[s],f[s]] are entirely disjoint; u is not a descendant of s in the depth-first tree since there's no path from s to u (otherwise there would be a path from s to x), and s is not a descendant of u in the depth-first tree since s is visited before u. Since d[s] < d[u], it must be the case that f[s] < f[u]. Moreover, Theorem 23.8 says that v is a descendant of s in the depth-first tree, so either f[v] = f[s] or f[v] < f[s]. In either case, d[v] < d[v].

b. Now suppose G has the property, that is, there are vertices u and v in G such that f[u] > f[v] in all depth-first searches.
   We first claim that there must be a path from u to v. If not, we can start a depth-first search at u and get f[u] < f[v] by Theorems 23.8 and 23.6.
   Now we claim that there's no path from v to u. If there were, we can start a depth-first search at v and get f[u] < f[v], again by Theorems 23.8 and 23.6.
   Thus there are a pair of vertices x and y such that there's a path from x to y and no path from y to x, namely x = u and y = v.

A few students were able to simplify the proofs somewhat by focusing on the edge that crosses from one strongly connected component to another, for example by letting v be x_{k+1} where x_k was chosen as in part a. The desired u and v are the endpoints of this edge.

Part a was worth 2 points; almost everyone earned both points. Part b's points were split three ways: 2 for the characterization, 2 for showing that every graph you described had the desired property, and 2 for showing that every graph with the desired property was included in your description. Many of you neglected to argue containment in both directions and thereby lost at least 2 points.

A frequent incorrect answer was the following characterization: all graphs where there is a path from the u mentioned in the property to the v mentioned in the property but not from v to u. The graph below provides a counterexample, since if depth-first search starts at s, u might finish before v.



This lost a point for the characterization and at least 1 other point in the "if graph then property" category. Other common characterization errors were the following:

- saying that there had to be vertices in two separate strongly connected components, but forgetting to say that there had to be a path from one to the other;
- saying that DAGs were the relevant class of graphs.

The first of those errors got a 1-point characterization deduction and at least a 1-point deduction in the "if property then graph" category. For DAGs, you need the additional condition that vertices are connected. With this condition, you possibly earned both the "if graph then property" points; without it, deductions were made in that category as well as in the "if property then graph" category. (Note that the graph above is a non-DAG that satisfies the property if you take u to be s.)

Your characterization also had to say something about accessibility in both directions to get full credit. In particular, solutions that talked about a path from u to v without mentioning that a path from v to u was not allowed lost 2 points for the characterization.

Many students lost points for inadequate detail in their explanation. Jumping directly from the existence of a path from u to v to the claim that $f[u] > f[v]$ was an example. Another common error, usually earning a 5- or 6-point deduction in part b, was to define the class of graphs in terms applied to vertices or edges in a *particular* depth-first search, such as "ancestor", "tree edge", and "back edge". Different depth-first searches would classify vertices and edges along these lines.

Problem 4 (8 points)

You needed to represent the Jolt Cola problem—finding how to measure 4 liters of Jolt Cola with three jugs of size 8, 5, and 3—as a graph.

Vertices are triples of nonnegative numbers that sum to 8. The first element of the triple represents the amount of cola in the 8-liter jug; the second and third represent the contents of the 5- and 3-liter jugs. (Actually, only two values per vertex are needed. The assumption that the contents of the three jugs totals 8 liters allows the contents of the third jug to be computed from the other two.) Two vertices (a,b,c) and (d,e,f) are connected by an edge if either a = d, b = e, or c = f, the other two are unequal, and one of the other two is either 0 or equal to the corresponding jug capacity. Another characterization of edges is that the vertex (a,b,c) is connected to as many as six other vertices:

(a – min(a, 5–b), max(5, b+a), c)	pour 8-liter jug into 5-liter jug
(a – min(a, 3–c), b, max(3, c+a))	pour 8-liter jug into 3-liter jug
(max(8, a+b), b – min(b, 8–a), c)	pour 5-liter jug into 8-liter jug
(a, b – min(b, 3–c), max (3, c+b))	pour 5-liter jug into 3-liter jug
(max(8, a+c), b, c – min(c, 8–a))	pour 3-liter jug into 8-liter jug
(a, max(5, b+c), c – min(c, 5-b))	pour 3-liter jug into 5-liter jug

The desired pouring sequence is represented by the shortest path from the vertex (8,0,0) to any vertex that contains a 4.

Points for this problem were divided as follows: describing the vertices are was worth 3 points, describing the edges was worth another 3 points for edges, and describing the source and destination of the breadth-first search was worth 1 point each. With the ordered-triple (or ordered-pair) explanation of vertices, the edge points were awarded as follows. You received 1 edge point for saying that edges had something to do with pouring. You received 1 more edge point if you said what the endpoints of an edge are, namely that they are the amounts in the jugs before and after the pours. For the third point, your description had to rule out partial (illegal) pouring and had to mention that a jug might be filled or a jug's contents might be exhausted in a pour.

The start vertex and end vertex points were awarded straightforwardly. We gave credit if you merely mentioned getting 4 liters in a jug; you didn't need to take the extra step or two to get (4, 4, 0). Also, some students mentioned that any of the vertices (4, 4, 0), (4, 0, 4), and (0, 4, 4) were reasonable stopping points. Since the third jug only holds three liters, two of those vertices aren't legal. We didn't deduct points for this error, however.

A common wrong answer was to have vertices representing only the contents of the 8-liter jug. This solution has no way to represent pours between the other two jugs. It could earn at most 3 points: 1 for the vertex description, 1 for the edge description, and 1 for the start and end vertex descriptions combined. To get the edge point with this approach, you needed essentially to give the 3-point edge description given above. Many students with this solution said that they could always change the contents of the 8-liter jug by +5, –5, +3, or –3. This could take the place of saying "always fill" or "always empty" a jug, since we assumed you had in mind either that the 5- or 3-liter jug was filled from or emptied into the 8-liter jug. You could earn 1 more point (not 2) if you stated the correct start and end vertices given your vertex description, namely that you should start at 8 and end at 4.

Another wrong answer was a three-vertex solution (one for 8, one for 5, and the other for 3). This solution at most 1. It received 0 vertex points, and could earn 1 point for mentioning any of the following:

> an edge description mentioning pouring *and* an indication to pour until empty or full;
> a sensible start vertex for the solution;
> a sensible end vertex for the solution.

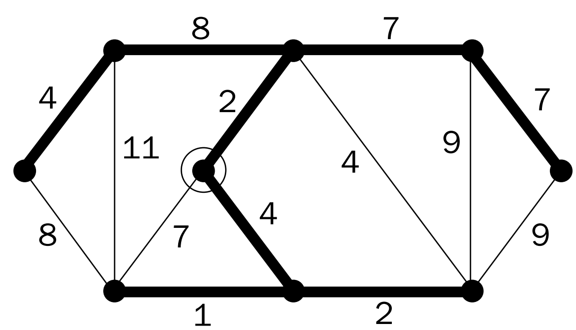
Explicit use of forbidden operations—e.g. "eyeballing" the pouring of half a jug's contents—lost you all the points on this problem.

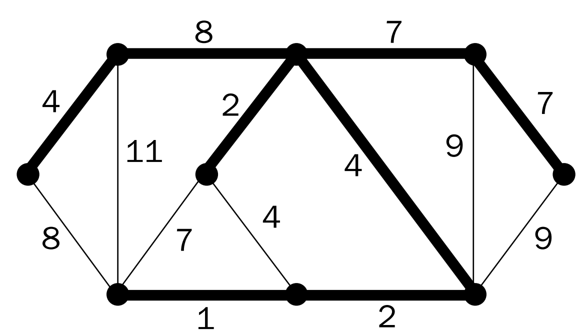
Incidentally, here's a solution pouring sequence.

> fill beaker 2 from beaker 1, giving (3, 5, 0);
> fill beaker 3 from beaker 2, giving (3, 2, 3);
> empty beaker 3 into beaker 1, giving (6, 2, 0);
> empty beaker 2 into beaker 3, giving (6, 0, 2);
> fill beaker 2 from beaker 1, giving (1, 5, 2);
> fill beaker 3 from beaker 2, giving (1, 4, 3).

Problem 5 (8 points)

This problem involved determining the last edge added to a minimum spanning tree by the Prim and Kruskal algorithms. Here were the graphs used.





Prim's algorithm's last choice is the left edge weighted 4. One way to show this is to trace through the algorithm. A faster way is to note that the edge weighted 8 has to be the next-to-last edge added to the tree, and the 4 edge is the only edge connected to it.

The last edge chosen by Kruskal's algorithm is the heaviest edge, namely the edge weighted 8.

Each part was worth 2, 1 point for the answer and 1 for the explanation, except that an incorrect answer almost always received a 0. Almost everyone got this correct.