1. Short answer.
   1. True, remains the MST. Every MST of a graph has -1 edges. The cost of this MST goes down by collectively, and so does every other potential MST.
   2. False:
   3. Recognizable but not decidable. A program decides the halting problem. Write a program whose inputs are a program and its input where it runs on line of but before everything originally on line . will return true if halts because it can get to that line, and will return false if it doesn’t halt, because it can never get to that line.
2. We do this using dynamic programming. The base cases, which we set at the beginning of the code, is

The recurrence schemes are

Because you either do just that one obstacle at step or do two obstacles, and .

We fill those two arrays in from to .

The solution is

This algorithm is because you iterate over each number from to , in each of which you perform several constant time operations. Damn it I overthought it; I had the correct answer before. The recurrence scheme is just . This relies on the assumption that we stopped at obstacle . I think mine is still correct, though.

1. I don’t know. We won’t start using a new truck unless . When we are on the m­th truck, the total weight is . So, we need more than that divided by many trucks (i.e. ). QED.
2. Convert from the SAT problem. Each of the clauses gets converted to a guest. There are variables. Each non-negated literal in clause means guest likes topping . Each negated literal doesn’t like topping . This SAT problem has the constraint that there must be at least one non-negated literal set to true and no more than one negated literal set to true in each clause. Can this SAT problem be solved with or fewer literals set to true? This modified SAT problem is obviously NP-complete because given a solution, you can verify its correctness (so it’s NP) and it’s harder than the original SAT problem, so it’s NP-hard (I really hope I don’t have to do another reduction to show this). Vertex cover is used in the solution. Guests are edges and toppings are vertices.
3. Convert from independent set. Each node is a judge and each edge is a contestant. Whether there is an independent set of size gets converted directly to this problem.
4. Unrecognizable. This is similar to a question on homework 8. But the solutions say undecidable…
5. Dynamic programming. Base case is . Recurrence is , where is the latest interval before that starts before starts. The solution is . Fill in the array from to . This is because you fill in all cells of the array, and for each one, you use binary search for . No. The correct recurrence was , where means students whose shifts cover event , and meaning the index of that student. For questions like this, remember that reduction to shortest path is good, too! Make nodes, one for each event. Make a directed edge for each student, starting and ending at his start and finish shift times; give the cost accordingly. Make backward edges for going from a future to a past event, all of cost 0. Make and before and after the first and last events, respectively. goes to the 1st student, has cost 0; is after the last student, has cost 0. The labeled edges in the cheapest path from to are the students hired.
6. 2 questions
   1. NP-complete. Reduce from the traveling salesman problem. If there’s an edge , then make , if there’s not, make . Set .
   2. This doesn’t have to go through all nodes, so it’s polynomial-time decidable. Just delete and get the cheapest path from to using Dijkstra’s or some shit. See if the cost of that is .
7. Make all the edge weights that go from event to even ( the negative of . Create a node that goes to the first event. Make the cost that edge . Use the Bellman-Ford algorithm to find the cheapest path from to every node possible. The cheapest set of edges to an event/node ever is the set of events to go to. Running time of Bellman-Ford is .
8. Reduce from weighted vertex cover. Each expert is a node. Each topic is an edge. The value of each node is . Each node’s incident edges are the topics the expert covers. If no other expert on the list shares a particular area of expertise (edge) with an expert (node), then set the “fake” node’s cost to 0. That’s wrong Greedy solution: keep on taking an expert that covers a new area of expertise and is the cheapest. For each area we need to cover, the cost of our corresponding expert is less than the cost of the optimal solution’s corresponding expert. The sum of all experts for each area in the optimal solution is then in turn less than two times the cost of all optimal experts. QED.