PROFESSOR: How about I propose this? We go through knapsack because it's on the Pset, not knapsack but a variation of it. If we have time, we do a couple variations. If not, maybe not. And if we have time, we do at a distance. Let's see what happens after knapsack.

AUDIENCE: [INAUDIBLE]?

PROFESSOR: How many people got the difference between polynomial and pseudo polynomial? OK, so we should definitely do knapsack.

AUDIENCE: Constant factors, they don't matter. Oh wait. [INAUDIBLE]. Yeah, they do.

PROFESSOR: So the thing is it's not constant factors. That s there is not a constant.

AUDIENCE: [INAUDIBLE].

PROFESSOR: So the knapsack problem. So the way I look at it. You're a thief. You somehow made your way into a vault. That vault has n items.

AUDIENCE: Way better than camping.

PROFESSOR: Each item has a weight of si pounds, and after you get out of the vault, assuming you make it alive and everything, you can sell it for vi dollars on the market, so this is how much money you get out of it, vi. Well, now the problem is the only thing you have with you as you enter that vault is one knapsack that can carry at most s pounds.

If you try to put more stuff in it, so if you try to load it up with more than s pounds, it's going to break as you try to escape from the vault and stuff is going to fall on the floor, and then many laser beams will shred you to pieces, so that's undesirable. So

we can only load the knapsack with s pounds. Now, given this restriction, we want to make as much money as possible out of the whole thing, so we want to load up the knapsack optimally.

Does this make sense? Everyone remembers? Good. So two ways to solve it, graphs and dynamic programming. We're going to do both. Who wants to go for graphs first? Who wants-- OK, never mind. Majority has been achieved.

How do we represent this? First off, let's represent the solution. Our solution is which items we chose, right? So we can phrase this as n decisions. Each decision is a true/false decision, and it indicates if you take that item with you or not. So di says, do I take item i?

So now this looks more like a game. You are on a TV, there's a game show, and you get asked n questions. Do you want to take this item with you, yes or no? If you somehow manage to get items that are more than s pounds heavy, you get shot, you don't make it to the end of the game. Otherwise, when you leave the game, you make some money and the money that you make depends on what items you took with you. So now this looks like the game problems that we used to solve with graphs in that you have decisions, those decisions are moves, and they get you through a graph of states.

Now, let's see what's in the state. What do we need to keep track of? First, we need to keep track of what item we're thinking about, right? And then there's something else that we need to keep track of.

AUDIENCE: How much capacity we have.

PROFESSOR: Yep, very good. So the reason I need that is when I'm deciding whether I'm taking an item or not, I want to know if I'm going to get shot or not. If I take too much stuff, I'm not going to make it, so I need to know if this item fits in my backpack or not. That's equivalent to knowing how much weight I have left.

So s pounds. Let's say this is capacity, how much weight left in my backpack's

capacity. This is equivalent to how much weight I have accumulated, the total weight of my items. Weight of the items I have taken so far. So the sum of the weights of the taken items.

AUDIENCE: [INAUDIBLE] left. Isn't it the total minus?

PROFESSOR: Well, I'm saying they're equivalent, so if you know one, you can compute the other. They're not equal. Knowing one lets you know the other one, but this is more useful in terms of putting the graph together, I claim. We'll see if that is true or not as we try to put the graph together. So what's a node? What's an edge?

AUDIENCE: An edge is putting in one more item, and a node is a state.

PROFESSOR: So what the weight going to be on an edge? If the edge means I'm taking an item, then the weight is going to be what?

AUDIENCE: The weight of the item you're adding.

AUDIENCE: The earning associated with that.

PROFESSOR: Yeah. I like this better because in the end, my goal is to maximize earnings, right? And I'm going to feed my graph to a shortest path algorithm. Maximize earnings, minimize path weight. They're the same issue, flip sign. So I'm going to say this is the value of the item almost. Does this work or not?

Negative value of the item. I have to flip the sign to turn it from a maximization problem to a minimization problem. So shortest path will give me the shortest path. That's going to correspond to making the least amount of money if I don't add this minus sign. So what's a node?

AUDIENCE: Is it a unique set of items?

PROFESSOR: Sorry?

AUDIENCE: A unique set of items.

PROFESSOR: OK. So the state in a node is going to have i, which is the item that I'm looking at,

because this way, I'm going to have one node for each item. And what else did I say I need to keep track of?

AUDIENCE: The weight of the taken items.

PROFESSOR: Yep. Weight of taken. So this is sort of like the gas problem where you have to keep track of how much gas you have as well as where you are on the map. Sorry if it brings bad memories. Let's talk about an example so that we can draw a graph for it and see what things look like. So say we have three items, and say our backpack has five pounds.

And my three items are a golden statue, value $10, weight, four pounds. These are 1800 dollars when money actually used to be worth something. Crystal ball, you can sell this for $4, and it weighs two pounds, and someone in the previous section wanted a fountain pen, so we're going to use a fountain pen that's worth $7. Culture is worth a lot of money, right? And weighs three pounds. Really ancient fountain pen. I don't know how people wrote with it. So how do we draw the graph for this?

AUDIENCE: We'd make a tree where each level the tree is take item i, or don't take item i. Make a big binary tree. Does that make sense?

PROFESSOR: So if each level of the tree says whether I'm taking an item or not, and then I have two descendants, then that's going to be 2 to the number of items. So that's going to be exponential in the number of items, whereas we're going to end up in a solution where the running time is proportional to the number of items.

AUDIENCE: [INAUDIBLE].

PROFESSOR: I mean, it makes sense in some cases, if you have fractional costs or something. By the way, all these weights are integers. Sorry I didn't mention that. My bad. But I like the idea of having some sort of levels based on items, so let's say I'm going to have a starting node, and then I'm going to have some sort of layer for item one, some sort of layer for item two, and some sort of layer for item three, some sort of vertical layer.

How many nodes do I have in a layer? One node for each possible weight because I promised that in a node, I keep track of the item that I'm considering and the weight of the items I've taken so far. How many possible weights do I have here?

AUDIENCE: Three weights.

PROFESSOR: So my backpack holds five pounds, so what are the possible sums I can get? How many?

AUDIENCE: 4, 2, 3, 5.

PROFESSOR: So far, I heard 2 3, 4, 5, 6. Who wants to bid more?

AUDIENCE: Not more than 6.

PROFESSOR: So the answer is 6, right? The possible weights are from 0, which is an empty knapsack, until weight 5, which is a full knapsack. So weight 0, 1, 2, 3, 4, 5, 6, and I'm going to start drawing the nodes.

AUDIENCE: You could have weight 10.

PROFESSOR: We're never going to fill up a knapsack with more than 5. Otherwise, we're going to die.

AUDIENCE: Why do we have 6, then?

PROFESSOR: Because I can't thank. Thank you. So this node, the first node. Weight in the backpack, 0. We're looking at item one. It's connected to the starting node. What outgoing edges do I have? What do I do with item one? I can take it or not take it.

AUDIENCE: You connect the edges 4, 2, 3. Am I answering a different question?

AUDIENCE: If it's item one, shouldn't it have a weight?

PROFESSOR: So those have two numbers in them, i and j. Sorry. They're in the wrong order here. 1, 0. So I'm looking at item one. So far, I have zero pounds in my backpack. Looking at item two with zero pounds, looking at item three with zero pounds. Looking at

item one with one pound in my backpack, item two with one pound, item three with one pound, so on and so forth.

So if I'm looking at item one, and I have zero pounds in my backpack so far, what happens if I don't take item one? Where do I end up?

AUDIENCE: 2, 0.

PROFESSOR: Yeah. So I'm looking at item two after I'm done deciding, what do I do with item one, and if I don't take item one, then I'm not going to have anything in my backpack. So this edge corresponds to we don't just take one. What if I want to take item one? Where do I land?

AUDIENCE: 2, 4.

PROFESSOR: All right. 2, 3. 2, 4. So if I did take item one, then I had zero pounds in my backpack before. Now I have four pounds, and I'm still considering item two. What about edge weights? What's the weight of the edge that says I don't take item one? What's the weight of the edge that says I do take item one?

AUDIENCE: [INAUDIBLE].

AUDIENCE: Minus 10.

PROFESSOR: All right. 0 and minus 10. That minus lets us get the shortest path. Now, suppose I'm in 2, 0. What are the outgoing edges?

AUDIENCE: It's the same as making a giant binary tree, right?

PROFESSOR: It's not going to be a tree because I can have multiple paths from a root to some node.

AUDIENCE: But this looks like it's taking the same complexity as building a tree if you were to carefully build the tree so that you don't have unfeasible solutions on the tree.

PROFESSOR: Then you're actually doing dynamic programming. You're not building the tree if you're doing that. You're still converging to this. You're probably thinking this and

you said binary decision tree, or at least that's what I understood. You're probably thinking of the right thing. We can see if we're thinking the same thing when we end up looking at the running time. Yes?

AUDIENCE: The goal here is to have the most valuable items possible in the bag in terms of money?

PROFESSOR: Yeah.

PROFESSOR: So the negative 10 weight that you have there was because that item was $10.

PROFESSOR: Yep. And in the end, we're going to give the graph to the shortest path algorithm. So yes, speaking of the goal, what's the answer here, by the way?

AUDIENCE: 11.

PROFESSOR: 11. How do I get 11?

AUDIENCE: By getting the last two items.

PROFESSOR: So I take the ball and the pen and I get 11, right? Everyone on the same page? So I'm at item two. I have zero pounds in my backpack. What are my outgoing edges?

AUDIENCE: To 3, 0 and to 3, 2. Wait, sorry. That's the third item.

PROFESSOR: No, second item. You're good.

AUDIENCE: Shouldn't it be 3, 3 because the third item has three pounds?

PROFESSOR: I said I'm looking at the second item and then I'm deciding where I'm going. So this graph has a problem in that when I get to the third item, what do I do? So I need one more layer here, which means that I'm done.

AUDIENCE: I was saying that it's the outgoing edge from 2, 0, it's the weight of the taken item, so if you decide to take the third item, shouldn't the outgoing edge go to 3, 3, not 3, 2?

PROFESSOR: But when I'm at layer two, I'm only looking at item two. So I'm looking at the items in

sequence. First I have to decide, am I taking item one? Then I decide, am I taking item two, and then I'm deciding, am I taking item three? While I'm here, I don't see item three. I only see item two.

AUDIENCE: Aren't those weights on the left side, though?

PROFESSOR: Yeah. These are backpack weights. So down here, these are pounds and these are items. What are the weights on the edges?

0 and negative 1.

PROFESSOR: OK. How about this other node, 2, 4? What are the edges coming out of it.

AUDIENCE: 2, 0.

AUDIENCE: 3, 4.

PROFESSOR: So if I decide I'm not going to take item two, but I already have four pounds in my backpack, I'm still going to end up with four pounds in my backpack, and I don't get anything out of it. And?

AUDIENCE: That's it.

PROFESSOR: And that's it, because if I try to take the second item, that would put me overweight, so the edge would go out of the graph. Therefore, it doesn't exist.

AUDIENCE: You can remove it, and then you go back to 3, 0, right? Is that not allowed?

PROFESSOR: No, because when I'm here, I don't know how I got here. I don't know if I had item one or not. The benefits of doing it this way is here, I'm just looking at item two. I do not know how I got there. I do not care what I'm going to do later. I'm just looking at one item and making one decision based on that. Does this make sense? Where do I get my answer from?

AUDIENCE: Can you put a final node on the right side?

PROFESSOR: OK, so one way of doing it is that I'm going to have a final note on the right side and

connecting everything to it.

AUDIENCE: Do you need the done nodes right there, or could you have just skipped done and connected 3 to that?

PROFESSOR: So I can do that, but then it's going to be hard to reason about edges. This makes it easier to reason about edges because when I'm looking at this layer, I'm still going to have edges deciding whether I take item three or not. So it would be confusing to have all the edges pointing to the same place. I can, it's just that the graph would look more confusing. By the way, if I'm at 3, 2, what are the outgoing edges from 3, 2?

AUDIENCE: It's just all horizontal. You can't take anymore.

PROFESSOR: So I can be done with two pounds, and that means I get 0 or I can take item three, and then where do I land?

AUDIENCE: Negative 7.

PROFESSOR: Does this make sense now somewhat?

AUDIENCE: So what about this dynamic programming style?

PROFESSOR: We'll get there in a bit. Before, let's see what's the running time for this. So one way of finishing this up is we connect everything to one destination node. Another way of doing it is that we connect the source nodes to everything here with edges of cost 0 and say, well, this is how much weight we're going to waste. So if my solution path goes here and then goes somewhere through the graph, it means that I'm going to waste one pound of capacity, so my backpack is going to have four pounds when I'm done.

AUDIENCE: Would it be [INAUDIBLE] or infinity?

PROFESSOR: What should the weight be? Good question.

AUDIENCE: This is not possible, right? To get from s to 1, 1.

PROFESSOR: So I'm saying that if I get from s to 1, 1, this means that I'm wasting a pound. My backpack will have four pounds of stuff and then one pound of capacity will go to waste.

AUDIENCE: Isn't there another way to waste it on the right side if you end at the top? Or not at the bottom, basically.

PROFESSOR: Yeah.

AUDIENCE: So you're not double counting in terms of losing stuff? You're representing losing weight on the left side and the right side.

PROFESSOR: Well, if I represent losing weight on the left side, then the advantage is that I have a single destination node. What is it?

AUDIENCE: That dot.

PROFESSOR: So this is the source. The destination is this guy. I can waste capacity here, so that means I can say that on this side, I don't need to waste stuff. On this side, I need to arrive at done 5. That's the advantage of doing it this way, aside from the fact that it's going to map to dynamic programming.

There are many ways of doing it. I can choose to connect the source to 1, 0, and then look at all the paths here and choose the best one. I can connect the source to everything here with paths of some weight, which you guys still need to tell me what it is, and then I can look at the answer here.

AUDIENCE: Cost 0.

PROFESSOR: Thank you. So this is another way. And yet the third way would be to only connect the source to the first node, and then connect all these done nodes to another node with cost 0, and this is equivalent to this. The reason we're doing it this way is because this maps to dynamic programming.

AUDIENCE: Easier.

PROFESSOR: Yeah. It maps closely to how the DP runs.

AUDIENCE: I mean is it possible to do the DP such that you do it the other way as well?

PROFESSOR: Yeah. You can flip the DP around, too. How are we doing with this? So then I hope the running time analysis will go really fast. How many vertices?

AUDIENCE: v or three.

PROFESSOR: OK, so v is?

AUDIENCE: ns.

PROFESSOR: n layers, right? n plus 1 layers, actually.

AUDIENCE: n times weight mass.

PROFESSOR: n times capacity, right? So it's actually n plus 1 times capacity plus 1, but I don't want to deal with that, so I'm going to use order of so that I don't have to deal with that. How many edges going out of a vertex?

AUDIENCE: Two.

PROFESSOR: Yep. So at most, two edges per vertex. So how many edges total?

AUDIENCE: ns.

PROFESSOR: What shortest path algorithm am I going to use?

AUDIENCE: Bellman-Ford.

AUDIENCE: Topological sort BFS.

PROFESSOR: I'm going to choose the second one because it runs faster. So Bellman-Ford's running time is V times E. If I use the DAG algorithm, that means topological sort and then a DFS. That's V plus E. So please, whenever you have a dynamic programming graph, this is the answer. It's never anything else. It's never Dijkstra. It's never Bellman-Ford. It's always this. And this means?

AUDIENCE: ns.

PROFESSOR: So this is the running time of the graph solution.

AUDIENCE: If you use BFS, do you need to put in the dummy nodes for the edge weights?

PROFESSOR: If you use BFS. Yes. So then the running time would be bigger than V plus E.

AUDIENCE: Right. So then the running time depends on [INAUDIBLE] multiply by s again?

PROFESSOR: This isn't doing BFS. This is doing the shortest path algorithm for direct acyclic graphs. It's the DAG shortest path.

AUDIENCE: Right.

PROFESSOR: Were you here before Thanksgiving?

AUDIENCE: No

PROFESSOR: Right before? You missed the algorithm. It's in the Dijkstra lecture notes.

AUDIENCE: But basically, you don't have to put in all the dummy nodes?

PROFESSOR: No. So you do a topological sort, and then you only consider a node once, and you look at all the edges coming through it and make a decision based on that. So this is the running time, this is the algorithm, this is the solution.

What if we do dynamic programming instead? Instead of nodes, we have sub-problems. A sub-problem maps to a node. The same things that we decided about this data that we store in a node, the same things are going to apply to the state that we have in a sub-problem. So what are the sub-problems? First up, how many sub-problems am I going to have?

AUDIENCE: n.

AUDIENCE: ns.

PROFESSOR: I just said that one sub-problem will map to a vertex in the graph. ns vertices, ns

sub-problems, right? Come on, guys. Bear with me. Get more cookies. Are the cookies still there? Please pass them around and eat them. Yes?

AUDIENCE: One sub-problem would be getting the most amount of money for that little weight. At one pound [INAUDIBLE], how much [INAUDIBLE].

PROFESSOR: I like that, so let's start writing. What is the maximum value that I can get using a certain weight? If I label the nodes indices i, j, I'm also going to label the sub-problems using i, j. So I'm going to say that sub-problem i, j is, what's the maximum amount of money I can get by using a weight of at most j, and you said something about items, so we have to figure out which items. Which items? Anyone else can help her. You already did the hard work, so you have most of the stuff filled in. Feel free to jump in.

AUDIENCE: Greater than or equal to i.

PROFESSOR: OK. Sounds good. We're going to number the items starting from 0 this time because we're going to have a dynamic programming table, and that means we like zero base indexing. So the items are going to be 0, 1, and 2. 0 is the statue, 1 is the ball, and 2 is the pen. This means we're going to look at suffixes of items. First, the empty set, no item, then only the pen, then the ball and the pen, and then the statue, the ball, and the pen. And we're also going to consider how much weight we have in our backpack.

So let me write this again. 0, 1, 2 here, and weights 0, 1, 2, 3, 4, 5, and this is going to be our DP table. Now, how are we going to compute this? DP of i, j is?

AUDIENCE: Maximum.

PROFESSOR: All right, a maximum of something. Good. Perfect way to start. How many decisions do we have?

AUDIENCE: Two.

PROFESSOR: Two. It's the same as in the graph problem, right? Once we solve that, this should be pretty easy because it's really very similar. So the two decisions are, do I take

item i, do I not take item i? If I don't take item i, what situation do I land in? So suppose I don't take item i.

AUDIENCE: i plus 1.

PROFESSOR: OK. i plus 1. So I have to make up my answer using items i plus 1 all the way through n minus 1, and how much weight do they have to have?

AUDIENCE: j.

PROFESSOR: Yep. Now suppose I do take item i. What happens?

AUDIENCE: i plus-- no, no, no.

PROFESSOR: You're thinking of the right thing. We're making some money, right?

AUDIENCE: Value i.

PROFESSOR: Value i. So this is how much money we're making. Good.

AUDIENCE: Plus dp i plus 1 j minus weight i.

PROFESSOR: Make sense? OK Now I only have to do one more thing to make sure this code doesn't crash.

AUDIENCE: I'm sorry. What's the third term for?

PROFESSOR: This is I'm making some money, right? This means I'm going to have to look at the items starting from i plus 1 and i minus 1. And now I took item i, so that means I have si pounds in my backpack. So the remaining items must weigh j minus si pounds because I'm going to add the si pounds for this item, and in total, I'm going to have at most j pounds. Make some money, lose some capacity.

AUDIENCE: Did we [INAUDIBLE]?

PROFESSOR: Yep. This is what happens when you move from the graph to dynamic programming, or you can draw the graph the other way around.

AUDIENCE: So could you have also done plus si, and then done something else as well?

PROFESSOR: Sorry. Plus si--

AUDIENCE: If you did plus si and then you did some other check case that was different.

PROFESSOR: Well, I do need to make a check.

AUDIENCE: Yes. So that's greater than or equal to 0.

PROFESSOR: This thing, right? Because otherwise, the code is going to crash. So j minus si is greater or equal to 0. I think this means j is greater than or equal to si, right?

AUDIENCE: Yes.

PROFESSOR: OK. At least for me, the natural order when I have the graph is to consider the moves that by making in the game. I'm answering the questions one by one. Do I take this item? Do I not take this item?

When I'm in the DP, when I make a decision, I want to have the full information on that decision. Whether I take the item or not depends on what would happen with the items following it. Representing it this way allows me to make the decision right here.

AUDIENCE: If we'd done prefixes rather than suffixes, we could have done it the same.

PROFESSOR: It would have been exactly the same as there. So we taught dynamic programming over many years, and it turns out that people understand suffixes better than prefixes, and the graph format makes more sense that way, so that's why we're doing it this way. It's for your own sake. Trust me. Or at least we think so.

AUDIENCE: Where does suffix come in? I know what it means.

PROFESSOR: These guys. These are the sub-problems. Nothing, item two, items one, two, zero, one, two, so it's a suffix of the set of items.

AUDIENCE: How come you're going backwards rather than forwards? It doesn't actually matter if

you go backwards or forwards?

PROFESSOR: Yep. In the end, we're going to look at everything. So we need two things. We need to know where is the answer going to be, and we need to know what are the initial conditions. Actually, I lied. We also need a topological sort. So where do we want to start?

AUDIENCE: First one.

PROFESSOR: Where is the answer? Yes?

AUDIENCE: The lower right hand corner. I mean, if you have all the [INAUDIBLE] nodes. Oh, we don't have a Done column, do we?

PROFESSOR: No. Well, so the answer will have considered all the items, right?

AUDIENCE: Top left.

PROFESSOR: So which i? Almost. So the top left corner means I looked at all the items and I have an empty knapsack.

AUDIENCE: Oh, so the max of all of the first column.

PROFESSOR: So the max of all columns would be the answer if the weight here would be equal, but I said the weight has to be smaller or equal, so the answer is easier. I don't have to do a max.

AUDIENCE: Bottom left corner.

PROFESSOR: I can only look here. So this means that I will use a weight of at most s, because this is s, and I will have considered items 0 and larger, so all the items. This is where the answer is. And in general terms, that means DP of what and what? So what's the bottom left corner?

AUDIENCE: 0, s.

PROFESSOR: Yep.

AUDIENCE: Can you explain again why it needs to be DP 0, s.

PROFESSOR: I think it's easier to look at 0, s and see how it maps to the sub-problem, and then convince yourself that this is the answer, so let's do that. DP of 0, s means i is 0, j is s. So this is going to tell me what is the maximum amount of dollars I can get by using a weight of at most s-- agrees with the initial problem-- and using items i or greater than i, or 0 or more, so items 0 through n minus 1. These are all the items. This is the maximum capacity, so this maps to the initial problem. And that's why I did this. That's why I don't have an equal sign here.

AUDIENCE: And if you did have an equal sign, then you would be changing your sub-problems?

PROFESSOR: The recursion stays the same, but I have to look at all these to get the maximum. Someone suggested that.

AUDIENCE: If you put an equals there, you're going to have to do something to the DP such that the equals actually is captivated, because you can't just arbitrarily--

PROFESSOR: What do you think of this?

AUDIENCE: Something there is going to change, isn't it?

PROFESSOR: No. I think this works. This stays the same no matter what the sign is. The only thing that changes is where is the answer and what are the initial conditions.

AUDIENCE: Oh. So there's something else that changes.

PROFESSOR: Yep. Initial conditions. Good. I like that you're thinking about that because the next problem changes pretty much that. The sign changes and then these change. What's a good topological sort? This should be pretty easy. If I want to generate a topological sort, what variables do I iterate over, and in what order?

AUDIENCE: i.

PROFESSOR: OK. And you're pointing at n, right?

AUDIENCE: n down to 0.

PROFESSOR: Yep. And then the only one left is j, so i in, and then j goes from where to where?

AUDIENCE: j goes to s.

AUDIENCE: It can go either direction.

PROFESSOR: Can it? What do the dependencies look like? When I'm computing i, j, I want i plus 1 and j minus-- so I'm looking at lower j's, higher i's and lower j's.

AUDIENCE: But it's always higher i's. Doesn't that mean in this for loop that you're already going to have calculated the next right column, so it doesn't matter where the column-- I may be wrong.

PROFESSOR: You're right. Fine. OK, both orders work. I just looked at this because I wanted to have a nice, simple-- but fine, both work.

AUDIENCE: Wait. You mean reversing j works?

PROFESSOR: Yes, because we're iterating over i before iterating over j. Of course, this makes more sense. Just because they're both good answers doesn't mean you should choose the one that requires more thinking. I always like the answer that requires the least amount of thinking to prove that it's correct.

AUDIENCE: Does that mean you're starting with a full--

PROFESSOR: I start here.

AUDIENCE: But if you're going from s down to 0, then you're starting down at the corner, which means you have a full knapsack, so that doesn't make sense.

PROFESSOR: So the argument was when I compute this, I'm looking one right, i plus 1j, and then I'm looking somewhere here. And I've already computed this column, so it doesn't matter if I go up or down. And that argument is correct because first I'm going to compute this column, then this column, then this column, then this column.

By the way, what's this column? This column is the initial condition. Now that we

figured out the topological sort, let's try to compute values here and see what the initial condition should be. So if I want to compute DP of 2, 0, that is the maximum of either DP 3, 0 or something else. This is going to evaluate to false, so it's going to be this. What should the answer be here?

AUDIENCE: 0.

PROFESSOR: 0. So we want DP of 3, 0 to be 0 for this to work. So what's a reasonable initial condition?

AUDIENCE: Column 3 is of 0 height.

PROFESSOR: DP of nj is 0 for all j, pretty much what you said in math mode. So I'm going to have an extra column. This is like the Done problem over there, and here I'm going to say that whenever I'm looking at the empty subset-- these are all suffixes. All three items, two items, one item, empty suffix. Whenever I look at the empty suffix, I can fill up a backpack with any weight by using no items, and I'm going to make 0 money.

So what are the values here? You guys nodded, so you understood, so then you should be able to dictate the values in the table.

AUDIENCE: So is that one also 0 because you can't go up to anything, and then the next one is 4 because you can go--

PROFESSOR: Sorry. Next one is-- so it's for the fountain pen, right? Last item.

AUDIENCE: Crystal ball.

PROFESSOR: So we're going statute, ball, pen.

AUDIENCE: And this is weight on the left side, right?

PROFESSOR: So we're filling them like this in the topological sort order that we chose here. We said this is the order, so this is what we're going to use to calculate the table.

AUDIENCE: Oh. So it's still 0, then.

PROFESSOR: OK.

AUDIENCE: 7.

PROFESSOR: So here, this condition is going to evaluate to True finally, so this is going to be the maximum of either DP i plus 1j, so DP of 3, 3, which is 0, or 7 plus DP of 3, 0. So 7 plus 0, which is 7. That's why it's 7.

AUDIENCE: 7, 7.

PROFESSOR: 7, 7. How about this?

AUDIENCE: 0. 0. 4.

PROFESSOR: The maximum of 0 and 4 plus 0, right?

AUDIENCE: 4.

AUDIENCE: No. 7.

PROFESSOR: So it's the maximum of 7 or 4 plus 0, so this is 7.

AUDIENCE: 7. 11.

PROFESSOR: 7 or 4 plus 7. Does this make sense for everyone? Last column. Let's do it.

AUDIENCE: 0. Feels like 0. 0 again. Why not? 4. How much does this thing weigh?

AUDIENCE: 4.

AUDIENCE: 7. 10.

PROFESSOR: Sorry for my handwriting.

AUDIENCE: And it needs to be 11 because you said that'd be the answer.

PROFESSOR: All right. So it's 11 because it's either this guy or 10 plus DP of 1, 1. The first item weighs 4, so I need to look one right and four up when adding. Does this make

sense? Please say yes.

AUDIENCE: What's the [INAUDIBLE] again? I get how if you're looking horizontally, the top thing is the max. It's saying DP of i plus 1 j.

PROFESSOR: So this is looking horizontally, and this means I did not take item i.

AUDIENCE: You didn't take item i, but if you are going to take item i, you add the value of i, and then you look back for DP of i plus 1, which is the previous column, and the row number is whatever capacity you have left if you did take it?

PROFESSOR: Yep.

AUDIENCE: And so in the case of cell 1, 3, if you took item one, then capacity you'd have left is 3.

PROFESSOR: Right. So you said total capacity three, right? So if I take item one, how many pounds does it have?

AUDIENCE: It has four.

PROFESSOR: Statue is 0, ball is 1, pen is 2.

AUDIENCE: Oh. You're not using--

PROFESSOR: So I'm using those items, but I'm starting using zero based indexing.

AUDIENCE: Yeah, but we're going from here to here. So I'm saying if you pick the middle cell that's 1, 3, that one--

PROFESSOR: So this has item one, so the ball, weight three. If I take the ball, how much weight do I have left?

AUDIENCE: The ball weighs two pounds, so you'd have three pounds left, and that's why you're in the three column.

PROFESSOR: I have a backpack of three pounds, and then I put a ball of two pounds in it, and I have three pounds left?

AUDIENCE: No, no, no. You have three pounds left if you put a ball and two pounds in, right?

PROFESSOR: I have a backpack of three pounds. I put a ball of two pounds. How many pounds do I have left of capacity? So backpack, three pounds total. I put a ball in that has two pounds. How many pounds?

AUDIENCE: We have none left. Oh, it can hold three total pounds?

PROFESSOR: Yeah.

AUDIENCE: Then you have one pound left. But we have a five pound backpack.

PROFESSOR: Well, you said we were looking at 3, 1. You said we were looking here.

AUDIENCE: Yeah, we're looking there.

PROFESSOR: So this means three pound backpack, because the sub-problem says your weight is at most j.

AUDIENCE: Got it, OK. Oh, I see. So j minus si is saying--

PROFESSOR: It means you used up, so you put up something there, so you ate some capacity.

AUDIENCE: Well, you ate only two pounds, though, so why is it in row three?

PROFESSOR: You told me to start here.

AUDIENCE: I know. I'm trying to remember how we got there.

PROFESSOR: You said, let's start here. That's why this arrow points here to one. You have two arrows, 2, 3, and 2, 1.

AUDIENCE: Because it's j minus si.

AUDIENCE: Oh, I see. Minus 3.

AUDIENCE: So j is the amount of weight you have left, right?

PROFESSOR: So j is the amount of weight I can have for this sub-problem.

AUDIENCE: Yeah, the amount of capacity I have still in my backpack.

PROFESSOR: Let's talk about pseudo polynomial time very quickly and what does it mean, OK? And you guys will promise to look at the recitation notes and look at the other problems, right? So incentive for that. Problem two is easier than knapsack, so if you get that, that should be a good confirmation that you got knapsack.

Problem three is a bit harder than problem two, but it shows up on interviews, so you want to understand problem three. I got problem two twice in four years, so there's a decent chance that you'll get it. So you want to get to problem three, so you should go through problem two.

AUDIENCE: How many times have you got problem three?

PROFESSOR: Twice in four years, so that's the problem that you want to get to. Problem two is a stepping stone. So running time for dynamic programming. How many sub-problems?

AUDIENCE: Same, ns.

PROFESSOR: Yep. Sub-problems. How many sub-problems do I look at when I compute the answer of a sub-problem? Two, right? So order one. So this is how much time it takes to compute the answer to a sub-problem, because I have the max of two elements, so it's constant time. So the total running time is?

AUDIENCE: Order ns.

PROFESSOR: All right. Order of ns. This polynomial, is this the same kind of algorithm as Dijkstra?

AUDIENCE: Pseudo polynomial.

PROFESSOR: Pseudo polynomial.

AUDIENCE: Don't know why.

PROFESSOR: So intuitively, the problem is s shows up in your running here, and s is the property of your input numbers. It's not how many elements you have in the input. It's what's inside one of those elements. So let's see why that matters. Let's look at the practical example.

AUDIENCE: What about measuring in cubic inches or cubic centimeters? Is that a problem? It would take a lot longer if we do it in cubic centimeters.

PROFESSOR: Yeah, but then you could argue that if it's an integer amount of cubic inches, you should divide everything by that. Where this really matters is suppose you have a 100 item input, so 100 items. What's the worst case input on a 32-bit machine?

An input looks like this, by the way. How many elements you have, so 3, and then for each item, what's the weight and what's the value? So then we're going to have three weights, which I believe are 4, 2, 3. You guys have to take my word for it. And then I have three values, which are 10, 4, 7.

Let's not worry about the values. I claim that they're irrelevant. You can convince yourselves afterwards that that's the case. Let's only look at this. The weights have to be between 0 and what for the problem to make sense?

AUDIENCE: 5.

PROFESSOR: If I have a weight bigger than this, I know I'm not going to take that item. How many bits do I need to represent these weights?

AUDIENCE: Log s.

PROFESSOR: It's log s plus 1 technically, but if I write order of log s, I'm good.

AUDIENCE: Times the number of items.

PROFESSOR: Yes. So if I want to represent all of them, I have order of n times log s, plus the number of bits required to represent this. This is log n. So log n is smaller than n. I'm not going to add it here.

So this is my input size. This is how many bits I need for the input. So the worst case input on a 32-bit machine is going to have roughly 3,200 bits. What's the worst case s that I can represent on a 32-bit machine? These are all 32-bit numbers.

AUDIENCE: You mean each of the weights are 32-bit numbers?

PROFESSOR: Yeah. So what's the worst case s I can represent?

AUDIENCE: 2 to 31 minus 1.

PROFESSOR: Which is roughly 4 times 10 to the ninth. So the worst case running time is? Roughly ns, right? So n times s, which means roughly 100 times 10 to the ninth, which means 4 times 10 to the 11 operations.

Now, suppose we're looking at the worst case input on a 64-bit machine. Still 100 items. What's the input size?

AUDIENCE: To the 21 total at the end.

PROFESSOR: That's how many operations. Let's go through it step by step. How many bits of input?

AUDIENCE: 6,400.

AUDIENCE: Where does that come from?

AUDIENCE: 100 items times 32 bits.

PROFESSOR: 100 items. Each item weight has 64 bits. What's the worst case s?

AUDIENCE: 2 to the 64.

PROFESSOR: 2 to the 64 minus 1. Doesn't matter too much. It's 1 times 10 to the 19th, 1.6 times 10 to the 19th. So the worst case running time is?

AUDIENCE: 10 to the 21.

PROFESSOR: 10 to the 21. So what happened? I increased my word size. The running time

increased quadratically. This is not a polynomial increase. What's the problem here? Why is this the case? If I write log s as b, so log s becomes b, what's the input size?

AUDIENCE: nb.

PROFESSOR: Yep. n times log s equals n times b, so this is the input size. Now, what is the running time?

AUDIENCE: The number of operations?

PROFESSOR: Yep.

AUDIENCE: n times 2 to the b.

PROFESSOR: Yep. n times s, and s is 2 to the b. So this is the problem. This is a polynomial in n and b, but here, if I write the input this way, I have b as an exponent. So if I double the number of bits by doubling the field size, my running time increases quadratically, so this is not polynomial. However, if I write it as order of n times s, this looks like a polynomial. So it looks like a polynomial, but it's not truly a polynomial, so that's why it's pseudo, as in fake.

AUDIENCE: So it's actually exponential.

PROFESSOR: It's not exponential. So an exponential algorithm for this means try all the possible subsets, and that's order of 2 to the n times n to compute the sum. This is exponential, right? You have n here. It's clear.

AUDIENCE: I know, but if you have 2 to the b, that seems pretty clear as well.

PROFESSOR: Well, you have to look inside this s and see what it means. That's the difference. So by increasing the number of items, if you double the number of items but you don't do anything to the word size, then your running time is going to increase polynomially, but if you change the field size, it's going to increase exponentially.

So pseudo polynomial means watch out, there's a trap. It's not really polynomial. If

you increase the input size by increasing the number width, bad stuff is going to happen. So think of Dijkstra. What's the running time of Dijkstra? You count the number of vertices, the number of edges, and you have a polynomial in that, right? You don't have to look at the number size. You don't have to look at any weird stuff like that. Here, you do. That's the difference.

Does it make more sense? So whenever you have an input number that shows up in your running time, instead of how many numbers you have, that means there's a trap. That's pseudo polynomial.

AUDIENCE: So it's just like having some constant that depends on something?

PROFESSOR: It's not a constant.

AUDIENCE: I mean coefficient.

AUDIENCE: Once you set it, it's a constant, right?

PROFESSOR: I mean, that's true for everything, right? You can say that you have 10 to the 80 atoms in the universe, so all the numbers that you work with are at most 10 to the 80. Therefore, your running time is order one no matter what you do, and then running times are no longer useful. You have to draw a line somewhere.

AUDIENCE: I think it's just like s, if it depends on your field size and you can scale it, it's kind of like--

PROFESSOR: So asymptotic running time. What's the point of that? How do our algorithms scale? As our data becomes bigger and bigger, what happens to the running time? This pseudo polynomial thing tells you that if you're shifting to a larger number size, to a larger word size, then your running time is going to explode. It's not going to scale linearly. Still don't buy it?

AUDIENCE: I trust you, I just--

AUDIENCE: So it's only dependent on the field in this case.

PROFESSOR: Do you guys want to look over Dijkstra and see what the input to Dijkstra looks like and why that's different? Do you think that's worth your time, given that this is what's standing between you and the weekend?

AUDIENCE: What time is it?

PROFESSOR: If you guys want to, I'm willing to do it.

AUDIENCE: It's 4:07. I'll stay.

PROFESSOR: Well, if you guys want to go, you can go. I will draw this for the people who still want to know what it looks like. Dijkstra. What's the input to Dijkstra? It's a graph, right?

AUDIENCE: Yeah, nodes and edges.

PROFESSOR: What does the graph look like? It's some number of nodes, so it's a single number-- that's the number of nodes-- that has log v bits in it. And then for each edge, we have three numbers-- first vertex, second vertex, and the weight. What are the sizes of these numbers? Log v, log v, and the last one?

AUDIENCE: Log w?

PROFESSOR: OK. And what's w?

AUDIENCE: Maximum weight.

PROFESSOR: Yeah. So this is a property of the field size, right? Let's look at v, actually. So we have E edges here, right? So the input size is going to be log v bits plus E times log v plus log w.

AUDIENCE: How did you get that?

PROFESSOR: Because I have the edges, so I have E of these. I only have one vertex count, but then I have E edges, and each edge has three numbers, and these are the widths of the numbers.

AUDIENCE: Is the adjacent edges to v? Is that right? You say you have E edges.

PROFESSOR: I'm assuming it's a list, so the most compact representation of edges, I think-- or it might be a reasonably compact representation is that you have the list of edges. So you have a graph that has five nodes, and then you have an edge that goes from 1 to 2, and then weight 3, an edge that goes from 1 to 5, weight 4.

AUDIENCE: If you use a number, it's not a--

PROFESSOR: Well, do I need anything else for vertices? Not really, right?

AUDIENCE: So v1 and v2 are neighbors of capital V?

PROFESSOR: So v1, v2, w, these are the first edge. Each edge has these fields.

AUDIENCE: This is the entire graph in one thing.

PROFESSOR: This is the entire graph as a list of numbers, and this is how many bits it takes to represent the graph in a reasonably compact representation. Now let's say little v is log v, little w is log w. So then this is order of how many bits do I have here? E times v plus w plus v. I just replaced the logs with these variables. This is how many bits. Now, how many operations does Dijkstra take? What's the running time?

AUDIENCE: Well, it depends on--

AUDIENCE: E log v.

AUDIENCE: Wouldn't it be E plus v? That's the fastest one, right? But I think practically, it's only going to be--

PROFESSOR: So this is E plus v log v, the fastest theoretical limit. This is still smaller than E log v. This is going to make my life easier. So this is smaller than this. If this thing is polynomial, this is polynomial for sure. E log v is E times little v. So how many bits in the input? E times v plus w. How many operations? E times v. Any exponential anywhere here?

AUDIENCE: Wouldn't you change the size of v? That looks fine.

PROFESSOR: Well, so there is a trick that v is 2 to the v, right? So you can say that this is order of-

- so this is definitely bigger than 2 to the little v times v, but then you have the same thing in the input. So the input also has at least 2 to the little v times v bits.

But don't worry about that. That's too much. The point is if you're worrying about this, don't worry. The math still works out. So whatever you have here as an input, the running time is going to be a polynomial in the size of the input. What happens if you double the word size? What happens if you have bigger weights?

AUDIENCE: Everything like v is multiplied by 2, and w is multiplied by 2 and everything in this problem, right?

PROFESSOR: So if you're doubling the word size, then this is going to double, this is going to double. Everything's fine. What if you double the size of the weights?

AUDIENCE: That only adds an extra bit.

PROFESSOR: Sorry. So if you double the size of the weight numbers? So if you move from 32-bit weights to 64-bit weights?

AUDIENCE: That's still a constant factor, right?

PROFESSOR: What happens to the running time?

AUDIENCE: Nothing.

PROFESSOR: Nothing. w does not show up in Dijkstra. If it would, then we'd be trouble. It wouldn't be polynomial anymore.

AUDIENCE: But practically, you will be accessing that number, right?

PROFESSOR: Yeah, but that shows up in the cost of one operation. That's why I'm saying this is how many operations you do, how many arithmetic operations. Then the model of computation that we use is RAM, and that says that you can do any math operation in order one.

AUDIENCE: [INAUDIBLE].

PROFESSOR: So here, my number in the input, s, which is the size of a weight, showed up in the running time.

AUDIENCE: Oh, the size of the input showed up in the running time.

PROFESSOR: So the size of the input is OK, but one number in the input showed up, whereas here, that's not the case. The weights do not show up in the running time.

AUDIENCE: So why don't we just always run Dijkstra, then?

PROFESSOR: Actually for this problem, for the knapsack problem, you can't find an algorithm that is polynomial. If you do, there's a $1 million prize for it because you just proved that p equals mp. This is the best we can do for knapsack.

AUDIENCE: Is counting sort pseudo linear, because you need to have a maximum, a range? Does that make sense?

PROFESSOR: Counting sort.

AUDIENCE: Depends on your range.

PROFESSOR: Yeah, but the range shows up under a log. You're allowed to have logs. You're allowed to have log w. You're not allowed to have w. It's dependent on the size of the input. If you double the number size, then you're going to have twice as many rounds, but you don't have an exponential number of rounds.

Sorry. You're thinking of counting sort. I thought radix sort. Radix sort doesn't matter because you assume we can do everything in-- never mind. You're right for counting sort. Sorry. You're right for counting sort. Sorry. I was confusing counting sort with radix sort. So for counting sort, yeah, it's not linear. It's linear in your range size, which is not linear in the input size, which is why we don't do counting sort. We do radix sort.

AUDIENCE: We do counting sort in radix sort?

PROFESSOR: Yeah. But radix sort limits the size of the range, right? That's the point of radix sort.

AUDIENCE: Oh. I see what you're thinking. So doing counting sort with each number?

AUDIENCE: Right, with really big numbers, taking a really long time.

AUDIENCE: Yeah, that would take a long time.

PROFESSOR: Yep. That is very true. If you try to do pure counting sort on 64-bit numbers, you're going to run out of RAM. Does this make some sense? OK. Promise to look over the other problems, and in return, given that I didn't have time to cover them here, I promise to answer any emails you guys might ask me over the weekend.

AUDIENCE: Oh. Awesome.