SRINI DEVADAS: Morning, everyone. Welcome to 6.S095. Thanks for registering for this class. I hope you like it.

My name is Srini Devadas. I'm a professor of computer science at MIT, obviously, and I've been teaching at MIT for 30 years. So I've been at it for a while.

The reason I'm teaching this IAP class is because I want to try out this way of teaching programming that starts with the recreational world of mathematical puzzles, algorithmic puzzles, and I want to connect it up to writing programs. And I have taught this material before in 6.009 that I think some of you are registered for, but I wanted to teach it in a more interactive, informal setting.

And I should mention that I wrote a book on this, which is a book that's recommended, though not required, for 6.009. And so you might see this in the MIT Press Bookstore and other places. And most of the material I'm going to cover here is from the book.

But I don't want you to read the book because I want you to listen to my explanations for the puzzles and try and think of the solutions yourself and then obviously do some amount of coding not during lecture but offline. And I'll show you some solutions to the puzzles, and there will be some exercises.

So I don't want to spend a whole lot of time on logistics. This is an IAP class, as I mentioned. I'd like this to be informal and relaxed. So we meet every day for two weeks. We have the holiday coming up on Monday, so we'll have nine total lectures roughly for an hour.

It might spill over, but hopefully, you'll be OK with it because you don't have another class to go to at noon. And if you do, well, let me know, and I'll give you that five minutes of overflow personally at a time of your choosing. But hopefully, I'll keep on time.

And in terms of work, this is a pass/fail. It's a for-credit class. I know some of you may be listeners, but some of you are registered for it. So I'll have to assign you a grade, so there will be some exercises. And I'm going to make it very simple.

There'll be exercises up on the website. They'll be due two days from when I put them up. And you'll have options in terms of whether to do a really simple exercise that's probably going to take about five minutes or do something harder. And for each of the puzzles, there are two or three exercises, and you get to do all of them if you want. You can choose to do one or the

other.

I'm going to have office hours every day from 1:00 to 2:00. I don't plan on putting solutions up to the exercises. The solutions to the puzzles themselves, they'll be shown mostly in lecture and put up on the Stellar website, but the solutions to the exercises, I'm not going to put up. But if you ever get stuck, well, that's what office hours are for. And if you can't solve the exercises, I'll help to solve them.

So with that, without further ado, let's dive in and talk about puzzles and solving puzzles using algorithms and then programs. I'd really like this to be interactive, so please ask questions and suggest solutions that I haven't thought of. That'd be great. In my second edition, whenever that is, you'll get credit for that.

So the first puzzle, I tried to have, I don't know, funny titles for all of these puzzles-- intriguing. And so the first puzzle is called "You Will All Conform." This is really a warm-up puzzle. As you'll see-- this is true for any class-- things are going to get more complicated as we go along. So we'll probably do 10 or 12 puzzles in the next two weeks.

And by the end, you're going to see things like memoization and dynamic programming and things like that towards the end of next week. And if you don't know what those mean, well, that's great because hopefully, you will know by the end of the two weeks. And a lot of these things are algorithmic insights, are insights that will help you in Course 6 classes, for sure, and perhaps even if you follow a career outside of Course 6.

So this particular puzzle is, as I said, a fairly simple puzzle, at least to describe. And the setting is as follows. So you're a gatekeeper at a baseball game. We'll just call it a baseball game because baseball players wear caps, and most people are watching games with caps on.

And so you're a gatekeeper, and there are a bunch of people standing in line. Let's just say each of them knows their number. And this is Python, so the number starts at 0 as opposed to 1. And there are a whole bunch of people standing in line waiting to get in, and they all have caps.

And some of them are wearing their caps forward, so with the-- I guess, what do you call it? What's the front of the cap called? The shade-- the thing that gives you shade up front. And so we just say-- let me try and draw this out.

So this person is wearing his cap like that, and this lady here is wearing her cap like this. So that's forwards, and that's backwards. So your job here is straightforward. You can only let all of these people into line-- sorry-- into the stadium who are in line if they all conform.

So they all have to wear their caps in a particular orientation, and you don't particularly care which one. And so they all have to be forwards, all, let's say, 13 of them. Or they all have to be backwards.

Now, all of them know their position on the line. So this person knows that his position is 0, this lady here knows that her position is 1, et cetera, et cetera. And so obviously, you'd like to minimize the amount of work that you do, and you could certainly choose-- let's say you want everyone to be forwards. You could say, hey, lady in position 1, flip your cap.

Oh, I should mention something that is important. So people have a different definition of forwards and backwards. It's sort of what's natural and what's unnatural to them. So we think that this person has his cap on forwards.

But for all you know, he thinks he's got his cap on backwards or vice versa. So you can't assume that the definition of "forwards" and "backwards" is identical for all of the people in line and really is the same as your definition. It seems kind of unreasonable. Hey, this is a puzzle, so let's not argue about that. We can argue about other things, like how to write Python programs properly.

So what you have to do now is you have to call out commands that correspond to saying, person in position 0, flip your cap. That's all you can say. You can just say, flip your cap. You can obviously see how these caps are oriented, and you can ask-- let's say that you had something like that. I'm just making this up as I go along, in terms of I'm not going to draw all the caps here.

But this is what it looks like. And so according to you, you see F, B, B, in terms of Forwards and Backwards. And let's say you decide that you want everyone to have their cap on forwards. Then you could say, person in position 1, flip your cap. Person in position 2, flip your cap. And then person in position 4, 7, et cetera.

And you want to make it easier on yourself because people do know their indices, their positions in line. And so to save yourself some trouble, you could say, people in positions 1 through 2-- and the implication is that it's inclusive-- flip your caps. And so that takes care of

these two. And perhaps 8 here and 9 had their caps on backwards, and you could say, people in positions 7 through 9, flip your caps.

So in this particular example, assuming that things ended with 10 people in line, 0 through 9, you could get away with three commands. You could say, 1 through 2-- and the implication, as I said, is this is inclusive-- flip. And then you could say, 4, flip. And then you could say, 7 through 9, flip.

And if you did it the other way, you could also say, person at 0, flip, and so this would go backwards. Person at 3, flip. Obviously, you can't say, 0 through 3 because that would be wrong. And then you could say 5 through 6 here. So you would get three commands in that case, as well.

But if in fact, for argument's sake, let's just say that I'm going to just turn this into 10 and put an F down here. Then these three commands would still work if you wanted to focus on the B's, on the Backwards people, and make them make them go forwards. But if you wanted to make all of the forward people go backwards, you would need 1, 2, 3, 4, four commands. So in this particular case, you're better off doing that as opposed to doing four commands.

So first straightforward question is-- and I'm not talking about code yet-- algorithmically, just in terms of pseudocode-- and I'm happy with English-- how would you determine the minimum set of commands given a particular set of people? There's an arbitrary set of people.

You want to make sure that you have the absolute minimum set of commands, and you're going to have to use these intervals. So you definitely want to look for all the contiguous intervals because obviously, if you say 1 followed and then that's a separate command, then you say 2 and that's a separate command, you end up having more commands than necessary.

So anyone want to tell me I guess roughly, in terms of pseudocode, how this would work? Yeah, go ahead.

AUDIENCE: Well, you could go down the line and look at the groups. So start with the 0. You've got to write 0. Group them. And then whatever the second one is-- so whatever the second group is, if it's F or B, those are the ones that need to be flipped because that's going to be the minimum number of commands. And then so from what you pass through, whenever you get to those, you just tell them to flip.

SRINI DEVADAS: Sounds good. Do you have anything to add to that?

AUDIENCE: No. I was going to say, you find the backwards interval or forward interval. And whichever is the lower, then--

SRINI DEVADAS: Lower. That's good. Good. Excellent. And so both of you had it right. So let me explain what these two gentlemen-- what was your name?

AUDIENCE: Ganatra.

SRINI DEVADAS: Ganatra? Yours?

AUDIENCE: Fadi.

SRINI DEVADAS: Fadi?

AUDIENCE: Yeah.

SRINI DEVADAS: So Ganatra and Fadi said-- so basically, this is, as I said, the natural algorithm that you would use. The natural algorithm would be that you walk through this list, and you start computing what I would call-- you called it "groups," Ganatra. But I'd call them "intervals," and an interval is something that is unbroken.

It's contiguous. It's a contiguous set of people who all have their caps in the same orientation, be it forward or backwards. And what you would do in this case is you would compute the-- you would say, I want to look at the forward intervals. And the forward intervals would be 0, 0 because that's essentially what you have in here.

And going through that list, you would also say, I'm going to be computing the backward intervals. And in this case, you discover 1, 2 as the backwards interval. So I'm going to write them out like that. And in general, in the mathematical notation, when you put square brackets, these are closed intervals. So 0 is included.

And we're going to see a puzzle where you might see something a little bit different, so I wanted to point that out. So 1, 2 are both inside this interval. And these two you would generate as you would go through the list, and then you would generate 3, 3 over here.

And then you would generate 4-- yeah, that's right, just 4-- I'm sorry, 4, 4 over here, if you want to call it an interval. You could represent this differently. There's nothing that stopping

you in Python to have these slightly more complicated representations where you just have 0 here as a number. And if it's just a number, then it represents an interval of length 1, but let's just be uniform about this.

Usually, when you have special cases and if you do things heterogeneously, then you have more code to write. It might get more efficient, but there's usually more code to write. So you do 4, 4 here. And then you'd go up, and then you'd have 5 comma 6. And then you would do 7 comma-- I'm sorry, 7 comma 9.

So this is the first time you actually have something interesting, in the sense that the interval has a representation that only has two numbers in it, whereas you actually have three people in that interval. And obviously, this could be arbitrarily large in the context of people getting into a baseball game-- and so 7 through 9 and then finally here 10 through 10.

And so now you go ahead, and you count and you realize that there's four intervals here and three intervals here. And then you say, I'm going to go ahead and go with the backwards and asking the people who are-- well, according to you-- backwards to flip their caps so we're all good.

So I'm going to show you code that implements exactly this algorithm. And while you see the code, I want you to think about this harder question. This code is going to do exactly what we describe. It's going to go compute all of these things, and as you can imagine, he's going to make a pass through this entire Python list, this group of people, this queue of people, and is going to do this computation.

Then it's going to say, 4 is greater than 3. And then it's going to go through and start calling out commands for each of the backward intervals. So in some sense, it's going to make two passes over the list.

So the first pass is to get all of the intervals together. Then there's a check. And then the second pass is to take those backward intervals, in this case, and call out those commands. So that's what I mean by two passes. Yeah, please.

AUDIENCE: You don't necessarily need to count them, the 4 and 3 because whichever orientation interval comes second is always going--

SRINI DEVADAS: Brilliant. Brilliant. So I was going to say-- if you didn't understand what Ganatra said, wonderful because then you can think about the question I'm going to ask and ignore what he said. And

then we'll get to, I guess, the more efficient code.

But what I was going to ask was, is there a way that I'm looking at this, and I'm just going-- and I'm going to just start calling out by-- the moment I see that there's an interval, the moment I see-- this is an interval. And I'm going to make a call with respect to whether I'm going to call out a command or not.

And I'm going to do this in one pass through this array. I want to do this in one pass through the array. I want to look at this, and I say, I see the interval 0, 0. Is that going to be something that I need to worry about in terms of that person having to flip his cap?

And then I see B, B. And I say, oh, the interval is 1 comma 2. Is that something I'm going to have to deal with in terms of a command? And so someone else other than Ganatra, tell me if there's a one-pass algorithm. And explain to me why you can do this in one pass through the array.

And just in terms of code, I'll preview this. The first algorithm needs-- according to my coding, which, isn't great-- 26 lines of code. The second one needs eight lines of code. So yeah, go ahead.

AUDIENCE: So after you see the first interval, you know that the second interval at the very least is going to have the same number of commands as the first one. So you might as well always go with the second one. So once you've identified what type the first interval is, you iterate through the list seeing-- and once you identify the first and second intervals, you iterate through every type of the second interval and do a command for that.

SRINI DEVADAS: That's absolutely right. What was your name?

AUDIENCE: Kevin.

SRINI DEVADAS: Kevin? So you had it right. And Kevin has it right, as well. So the observation here is actually-- it was said a little bit differently by Kevin, but I'm going to say it a little bit differently. The observation is simply that if you look at the very first orientation, the very first orientation is something which, at best, is going to be a tie.

So if you had it ending with 10 people, 0 through 9, then the number of forward intervals is the same as the number of backwards intervals. And if you went here, obviously, the number of forward intervals is greater than the number of backwards intervals. So you really only have to

look at the first person in line not to determine the intervals because when you see the first person in line, you have no idea what the intervals are that are coming after.

You have to generate those intervals, so you have to make your pass through the array. But the first person in line gives it away in terms of what the final result is going to be with respect to your decision as to what set of commands you're going to call up. And so because this is an F, you basically say, I'm going to go, and I'm going to make the people with the B's flip their caps.

And that's a small observation in terms of its insight. But obviously, it's potent, in the sense that it's going to give you a one-pass algorithm versus a two-pass algorithm. And as I mentioned, it's going to give you a substantial reduction in the amount of code that you write.

So I'm going to show you a bunch of code now. So this is usually how this is going to go, by the way, this entire class. And this is kind of my way of teaching, at least this type of material that's algorithmic and has data structures and so on. I like explaining to people what they're going to see in the code without having to deal with Python syntax and worrying about whether it's while loops or for loops and things like that and tuples or arrays and things like that.

So let's get into that. I don't need this. Ah, good.

So the naive algorithm first-- so what you see here is an input. So "caps" is simply an input Python list that has, as you can see from our examples, the F's and the B's. And so F is Forwards, and B is Backwards. And there are a couple of different Python lists so you can run them with different things. And all of this hopefully is not at all surprising to you because we went through all of this in terms of the pseudocode.

So the first initialization-- and I can just highlight that to make it easy. So that's initialization. And this part here-- the comment sort of gives it away-- is you're making a pass through the array, computing the intervals. And so how do you actually get the intervals? And there's one line of code that does that.

And oh, by the way, if there are any questions about Python syntax, just don't feel shy. Just tell me. Ask me a question, and I'm happy to explain it.

I don't want to explain things that you all know, obviously. But if any one of you doesn't know any particular thing, please don't feel shy. Syntax is something that sometimes you even

forget. I tend to forget things.

So this thing here is essentially something that decides on what the intervals are. And the insight here is simple. You know that this interval ended when you see something here that is different from what the previous one is. So that's when you know. So you generate the interval when you go and find something that is different from the current location that you're pointing to.

So that's really what's going on out here. In this line, you say, you're setting a "start" here. "Start" is your counter, and remember, "start" is getting set again here. Initially, it was set to 0. And if you ever get to the point where you're looking at "i," it's the next thing you're looking at, and if "caps start," which is the start of the interval, is different from "i," it meant that the interval that began with "start" ended.

And so you essentially say, well, that needs to end at i minus 1 because obviously, "caps start" is different from "caps size." So this is the end of the interval right here. And this particular thing, our interval, is something that includes both these two numbers, so it's not exactly that.

But it also includes-- where did my chalk go? It also includes whether this is an F or not. So each of these things has-- I'm not going to write it out for all of those things-- but has the orientation inside of it.

So that's what you have out here. That's why we have three of these in this tuple. This is a tuple because it's got round brackets around it. I could have made it a list if I wanted. And so that's essentially what we have here.

And now, here's what happens. In this particular piece of code, when you get to the end and you fall off the end-- so if you just end, then this code essentially skips over and doesn't actually put in the last interval because you don't see something. When you come up here and you're off the end of the array, you fall off the edge of the array, you don't see a B that is different from F.

So if you just wrote that code-- so it's something that I'd say many people would have a quote, "bug," where they wouldn't add this part of the code here because they didn't realize this particular point that I made. And so that last interval doesn't get added in. And so you have a bit more code here that decides on the last interval.

And then you count, and then you go ahead and call the commands. So this all looks good?

Yeah, we're good with this?

Is there a way that I could somehow eliminate these four lines of code which are a little bit annoying and just do things in the array? How could I, with a little trick, eliminate those four lines of code? I might need a little bit more in, quote, "pre-processing," but how would I eliminate that?

Someone else? I'll get to you. Yeah. Go ahead.

AUDIENCE: One way is you can check thoroughly you've reached the end of the-- the forward is equal to the backwards.

SRINI DEVADAS: You could have an extra check. What's your name?

AUDIENCE: Kanishka.

SRINI DEVADAS: Kanishka? So Kanishka says that before you reach the end of the array or when you reach the end of the array, you go ahead and do a check that you've reached the end and generate the interval inside of it, which is kind of like moving this code inside of it.

So that's reasonable. I'm not sure that's going to be as good as the way that I want and which I coded. Think about pre-processing. Think about taking the array-- yeah?

AUDIENCE: I guess you could add a wrong interval at the end.

SRINI DEVADAS: You could add on a wrong interval. What was your name?

AUDIENCE: Kevin.

SRINI DEVADAS: Kevin, too? Just how many Kevins do we have here? So you could do this. So let me show you something that's a little bit more-- so that was 26 lines of code.

And so this says-- this is the line that's interesting. So I had F and B, and so I don't want to add F or B there. But what is convenient to do, as it turns out, is to just have something that essentially says, there's an ending point. So you have an explicit end to the array that is actually not something that would crash if you tried to access it.

You had n people in line, and now you have effectively n plus 1 people in line. But that n plus one person is a dummy, or you can call them "end." And then if you do that, then that's it. The

previous code works. You can just remove those four lines, and it'll all just work because your F or your B is not equal to "end."

So whatever you had at the end-- I'm sorry. I'm overloading terms here. Whatever you had at the last element-- whether it was a B because it ended here or F-- when you equate that to "end," you're going to get an inequality. And so that last interval gets generated-- so small optimization.

Tony Hoare was a very famous computer scientist-- said that "inside every program, there's a smaller program waiting to get out." So this is kind of one of the themes that I'd like to harp on in this class, which is it's nice to write compact code. Usually, if you can write compact code, there are fewer bugs in it.

And you don't want to go overboard. You don't want too many subtleties, but it's nice to write compact code. And so this is a little bit more compact.

And then finally, here's the one-pass algorithm. So the whole thing, the "pleaseConformonepass," is the smarter algorithm that essentially says, look, if I just look at the very first thing, I'm going to be able to skip that and then move on. So this is fairly complicated.

You know the algorithm. Trying to map that to this code is non-trivial, so we'll spend a minute or two on it. But here's how this works. And by the way, just to make sure-- so I've repeated things here.

So the optimized algorithm for that particular example produced these first three commands. And then the one-pass algorithm produced exactly the same commands. So this is good to do. It's always nice to have a couple of different ways of solving a problem because then you can verify things.

Anyway, so this piece of code is essentially something that does a similar trick to what we had before, in terms of adding to the original list, adding an element. But we're not using "end" here. We're actually saying, what we want to do because we're going to go ahead and skip this, anyway-- we know we're going to skip the F.

So whatever that is, we could just sort of add up here. And so that makes it easy. You don't have to worry about what the actual characters are and cooking up this "end," which is different from the F or the B because hey, it's possible that people might use whatever they

want to represent a forward cap or a backward cap.

So this is very clean. You go ahead and put exactly that over here. You know you're going to skip that interval because you're going to skip the first interval, effectively, because that is what the algorithm allows you to do. And then this part here is doing what we had before.

We're not even generating the intervals. We're just directly making the commands. So there are no tuples in terms of the tuples and the appending of the intervals data structure that we had. It's all gone. I'm just going to go ahead and fire off these commands. And those commands are written in kind of a funny way because I'm going to go ahead and if I wanted to print these things out, you'd have to do this in this funny way.

If you're willing to wait and collect up the intervals, you can certainly do that. But this is absolutely the most compact code that I could think of, and I'm printing a command in two different parts. I'm printing the first part of the command that says, people in positions. And that's the start of the interval "i," and I'm just using a Python construct.

This "end" here is something that the print command can recognize. And this simply says, don't give me a new line. I want to print it exactly. As you saw from when I ran the code, the printing was exactly the same as the original algorithm.

And so this "end" simply says, don't print a new line at the end of people in positions, whatever this number is. Call it 7. And you don't print the new line.

And then you have a space here because you didn't print a new line or a space. And you go through i minus 1, which is exactly the same as we had before. You only discover that the end of an interval after you see the person that comes after the interval. So you have to go back to i minus 1.

So this makes sense? Any questions about this code? Yeah, go ahead.

AUDIENCE: So the first line, like you said, if we're going to move the first--

SRINI DEVADAS: Not move, duplicate--

AUDIENCE: Duplicate the first--

SRINI DEVADAS: Yeah. So the plus there is a concatenation, and you have two lists. "Caps" is a list. And when you use a plus operator in Python, you can only add things that are of the same type.

And so "caps" is a list, and "cap 0" is an element of the list. And so you need to make it a list in order to use the plus operator. There are also other things you could do using a pen, for example. So you could take that line, and you could do "caps start" at "pen" "cap 0." And you wouldn't need all of the square brackets, but that's just neither here nor there. But I'm obviously not answering your question, so go ahead. Yeah.

AUDIENCE: So I was going to say that we skip the first interval. And then we flip all the second-type orientation.

SRINI DEVADAS: That's right.

AUDIENCE: But here, we're focusing on the first element. Why--

SRINI DEVADAS: Ah, but that's OK because remember, the if statement is going to skip this one, too. So if I had two F's in the beginning-- I think your question is, what would happen if I had two F's or three F's in the beginning? So I'm going to go ahead, and first thing I'm going to do is I'm going to put an F at the end here, and then maybe there's a bunch of stuff.

But then the nice thing is that I'll skip this F, too, because the not equal to is not going to fire. The next line, the "caps i not equal to i minus 1," this is going to be equal to. So I'm going to go ahead and go to the next iteration of the loop.

That makes sense? Did people understand Fadi's question? It was a good question. So the question was, you skipped the first one.

But what is making you skip this one because you want to skip the first interval? And so that if statement is making you do that. Good.

So as you can see, even a fairly straightforward puzzle-- this is the simplest puzzle we're going to do here is the first one. And there are nuances associated with how you solve it but also how you code it. And this is kind of-- like I said, this was my eureka moment a few years ago, where I said, I think the way you want to teach programming, at least some aspects of programming, is in this fashion.

Anyway, good. So we're done with this. Any questions about this puzzle? All right. Good.