VICTOR COSTAN: Hi, everyone. I am Victor. My full name is Victor Costan. I'm a third year Ph.D. student. I'm in CSAIL, and I work in systems. I do mostly security and distributed systems. And I'll be your algorithm TA for this term. Yay.

How many people went to lecture yesterday? All right. Are there any parts of the lecture that were particularly unclear that you guys want to go over? So that I schedule what I talk about in such a way that I'll cover that more than the parts that are boring and obvious.

AUDIENCE: Can you explain complexity in equations, because I took 042 like 3 years ago.

VICTOR COSTAN: OK.

What else? Thank you.

AUDIENCE: Maybe some basic outlines for, like, writing proofs relevant to like the [INAUDIBLE].

VICTOR COSTAN: All right.

AUDIENCE: Sort of methodology or something.

VICTOR COSTAN: OK. Anything else? OK. We're going to get started then.

So the hardest, and I think most important of the whole course, is learning how to deal with asymptotic complexity. So asymptotic complexity is this very nice tool where you have a function that looks like this plus-- what'll I have here? 10 plus sin x. So you have this guy, which is quite long and horrible and would make your life

miserable if you'd have to reason about it. And you can say that this is theta of n squared. x, sorry.

So everything that's here goes away. And this constant factor here goes away, and it's nice and small function. Why is that the case, first of all? If I plot this function-- and since I don't have my projector you'll have to take my word for it-- I will get something that looks like this. If I plug this function, I will get something that looks like this. This is a scale from 0 to 200.

However, if I make my scale 0 to 20,000-- Let's say 200,000 to make sure that is going to be correct. x squared looks like this. And this function looks like this. And as you make the scale bigger and bigger, you can't even see this noise anymore. The reason is that all the noise is in this term. In the x to the power of one fifth term. And as n-- pretend this is n, x, n same thing-- so as the input to the function gets bigger and bigger, this grows at a much faster rate than this. So if I don't care about this, how the function grows, I can ignore anything aside from the dominant term.

Do people remember this from lecture? Does this make sense? OK. If it doesn't, that's great. It means that I'm not wasting time covering it. So we learned about three symbols, or we will learn about three symbols, when talking about asymptotic complexity. The three symbols are theta, O, and omega.

Let's say this complicated function is g. And this nice simple function is f. If I say that g of x is theta of f of x, that means two things. It means I have a lower bound, like I showed here, and the lower bound looks something like x squared. And it means that I also have a higher-- an upper bound for it that also looks like x squared. So the upper bound would be something like this.

And if I say this, then I'm making a promise that the function stays between the lower bound and the upper bound, and the bounds look sort of the same. I can't really have x squared here, and x squared here. Right? That would be impossible. So that's why I said they look the same. They're not the same. And they're not the same, because they're allowed to differ by a constant factor.

So if this is f, and say this is f prime, f of x might be x squared. And we know for sure that this is going to be lower than this. And f prime of x might be 1.2 times x squared. And because I have a 1.1 here-- and I know that these get smaller and smaller in relation to this as the function-- as the inputs to the function get bigger-- I know that this is going to be the dominating factor. And because this is bigger than this, this function is going to eventually be larger than this function.

So I have a bound from above, and I have a bound from below. And the bounds look almost the same. They're only different by a constant factor. This is theta.

To make things spicier-- by the way, theta is really hard to pronounce as opposed to O, so we usually say and write O instead of theta. We'll talk about what O really means in a bit. But if you see O in a lot of places, including our notes, know that most of the time we actually mean theta. And sorry for that.

AUDIENCE: When do we you actually mean O?

VICTOR COSTAN: You can ask us, and we'll tell you.

[LAUGHTER]

Fortunately, it doesn't a matter too much. So we're trying to get an idea of how the functions look like in the bigger picture. So most of the time it doesn't matter. I'll give you an example though of where it matters, and how it could matter.

So suppose I have another function that looks like this. So this is kind of ugly, too. This function is g of x, say x times 1 plus sin of x. So I can't say anything about the lower bound of this. This is going to hit 0 at an infinite number of values. Right? So I can't say that-- I can't have a nice lower bound for it. The lower bound for it is 0. But I'm pretty sure that it's never going to be bigger than 2x. Right? And that's still worth something.

So there's this other function here that's supposed to be a line. Pretend it's a line. f of x equals 2x. And I know for sure that this is never going to exceed these, this.

And in that case, where I only have one upper bound, and I don't have a good lower bound, I say that g of x is order of f of x. And this is where-- This is what O really means. It means that I only have an upper bound and not a lower bound.

In most of the cases when you're coding something, you don't really care how well your program can run if you're really lucky. What you really care about is how does it run in the worst case? Or how does it run most of the time?

So suppose you're Google, and you're writing your search engine, and you have to return the result to the user in 200 milliseconds. Because they did a study that shows that if you return the result in 400 milliseconds, they lose some number of users, which means they lose some number of billions a year. So to them a guarantee that their algorithms complete in 200 milliseconds is pretty important, because it means a ton of money.

So if you have a running time that looks like this, that's OK, as long as this is 200 milliseconds for the number of pages that Google indexes. Does that make sense? OK. No answer means yes. So if it doesn't make sense, please say something.

OK, I will draw another function. Again--

AUDIENCE: Victor?

VICTOR COSTAN: --wavy. Yes?

AUDIENCE: It says x times 1 plus something [INAUDIBLE]. So what?

VICTOR COSTAN: So it's--

AUDIENCE: Oh it's sin.

VICTOR COSTAN: 1 plus sin of x. It--

AUDIENCE: It looks like [INAUDIBLE].

VICTOR COSTAN: Yeah. Sorry, my handwriting is not very good.

AUDIENCE: It's OK. Mine isn't either.

VICTOR COSTAN: And it doesn't really matter what it is, it just needs to be sin so that it looks wavy like this.

AUDIENCE: OK.

VICTOR COSTAN: So that I can make something that looks complicated. OK.

AUDIENCE: Got it.

AUDIENCE: I have a question, too. In the first example, is it 1.2 times x squared?

VICTOR COSTAN: Yes. I wanted something that's higher than 1.1.

AUDIENCE: It's like [INAUDIBLE].

VICTOR COSTAN: Yes. OK. Now that I got a break, does anyone else want to ask questions? OK. So I'm going to draw another function. This one looks like this. And this is f of x equals 1 plus sin of x. I have to use these signs to make them look like that. Times x to the power of-- What do I have there? 1 5 plus x to the power of one fourth. So for this function I know for sure that it is greater than x to the power of one fourth. Right? So if this is--

I know for sure that it's greater than that. Also if I look at this, and I know that sin is never bigger than 1, I know this whole thing is going to be, at most, 2. So if I have another function that says f prime of x equals, say, 3 times x to the power of 1.5, I know that it's going to bound it from above. So I have two bounds. So it sort of looks like this case.

Except when I use theta, I have to make sure that the bounds are very similar. They can only differ by a constant factor. In this case my bounds differ by the exponent here, which is more than a constant factor, so they're not the same.

So if I take my function-- which by the way is g-- if I want to say that g to the x is theta of something, I can't. Because no matter what I put in here, it's not going to be

correct. The lower bound is this, the upper bound is this. But what I can do is I can say that I know for sure that it is theta of x to the power one fourth. And by the way I can also say that it is order of x to the power of one fifth.

So if you have weird running times that looks like this, then you can only use O and omega. And with that being said, we will use O for pretty much the rest of this course. And for most of the cases we'll use it to mean theta.

Oh, by the way, these formulas look ugly and complicated, because I wanted the graphs to look complicated to illustrate the idea that asymptotic notation makes things look simpler. In this course the running times that you'll be dealing with are all look like this. They will either be order 1, order log n, order n, order n log n, n ordered n squared.

You might have one or two cases where you get something else, but most of the time if you have something else coming out of your occurrence, it means you're doing it wrong. So nice tip to have. And you can see that I already started doing it. I'm already using O instead of theta. So I guess I'm getting into the habit.

AUDIENCE: So why did you leave the theta one blank for g of x?

VICTOR COSTAN: Because whatever I put here is going to be wrong. If I put x to the power of one fifth, this is wrong, because I can have an upper bound that has x to the power of one fifth in it. But I can't have a lower bound like that, because this function hits x to the power of one fourth in infinitely many points.

If I try to do x to the power of one fourth, same deal. Lower bound yes, upper bound no. So whatever I put in here is going to be wrong. Thank you. Good question. Yes?

AUDIENCE: Is that 1.4 or one fourth?

VICTOR COSTAN: 1.4. They are 1.4 and 1.5

AUDIENCE: OK. Because I thought you were saying 1/4 and I was--

VICTOR COSTAN: Oh, sorry. 1 4, 1.4. 1.4, 1.5. OK. Are we good so far?

OK. Let's go through a couple of exercises. So I'll give you a few functions-- Yes?

AUDIENCE: Do you make omega-- Is that-- It's a lower bound already without multiplying by any constant, or something? Is that what it is?

VICTOR COSTAN: So for all of these, what you put here is the simple, is the simple form. So you're lower bound can be this function multiplied by any constant factor.

AUDIENCE: That-- So omega means lower bound?

VICTOR COSTAN: Omega means lower bound, O means upper bound, theta means both. OK. Any other things to clear up before I ask questions? So it's your turn before it's my turn.

All right. I'm going to go over a few functions, and I want you guys to tell me what-- to help me write them in asymptotic notation. So I will have f of n. And I want to know it's theta of what? So let's start with f of n equals 10 to the 80. Does anyone know why 10 to the 80? Yes.

AUDIENCE: That'd be O of 1.

VICTOR COSTAN: OK. That's the correct answer. You 're going ahead though. I was wondering if-- I was wondering if you know why? Why did I choose this? Does anyone know why I chose this? Yes?

AUDIENCE: Is that [INAUDIBLE]?

VICTOR COSTAN: I think that's 100.

AUDIENCE: Oh, OK.

VICTOR COSTAN: You're getting close though. It's something.

AUDIENCE: It's a really big number.

VICTOR COSTAN: It's a really big number. All right. So it's a really big number. That's the estimated number of atoms in the universe.

So realistically any quantity that you deal with that will represent the real world, is not going to be bigger than this. So if you want to get all philosophical, you can say that for any algorithm that you write in any realistic input the running time will be theta of 1. But if you do this, we're not going to be happy with you on exams.

[LAUGHTER]

So let's choose another one. 20 n to the power of 7. What's this an asymptotic notation? And why? Yes?

AUDIENCE: Order O of n to the seventh.

VICTOR COSTAN: OK. Of that same thing, n to the seventh. Thank you. Why is that?

AUDIENCE: Because if you distribute the exponent 7, 20 to the seventh is a constant.

VICTOR COSTAN: Excellent. 20 to the 7 time n to the 7, this is a constant so it goes away.

OK. And before I do another one, I want to give you guys some help and remind you about logs. How many people remember the laws of working with logs from algebra or some math class? All right. You guys will be helping me write them for everyone else.

So if I have log of n to the 100, and I want to get to the theta notation for it, what is it and why?

AUDIENCE: Log n.

VICTOR COSTAN: OK. Excellent. Why is that the case?

AUDIENCE: It's 100 log n.

VICTOR COSTAN: OK. So when I have an exponent here, I can pull it out in front of the log as a constant factor. So this is the same thing as 100 log n. OK. Now if I have the logarithm to the base using base 5 of n-- How about this?

AUDIENCE: [INAUDIBLE] multiply. That's just log n?

VICTOR COSTAN: OK. So this is-- can you say that again?

AUDIENCE: Log n.

VICTOR COSTAN: OK. So this is theta of log n because it is--

AUDIENCE: Log n over log 5.

VICTOR COSTAN: So that's log n over log 5. So exponents inside the logs turn into constant factors, bases turn into constant factors. So if you have an ugly expression inside a logarithm, you can, most of the time, simplify it to something really nice. And this contributes to the fact that these are the only functions that you will see when the answer is cleaned up in asymptotic notation. OK. Yes?

AUDIENCE: So when you're writing log, are you assuming the second base then?

VICTOR COSTAN: Yes, there is a certain base for logs. We're all computer scientists. Right? So what's the natural base for our logarithms?

AUDIENCE: 2. 2D.

VICTOR COSTAN: OK. Sweet. Math major?

AUDIENCE: No, you know I always heard natural base--

VICTOR COSTAN: OK. Yeah, it's a math joke. So for everyone else a natural logarithm is in base E. For us CS people, a natural base is 2, because that's how many values you have for a bit. So log means log n is to log base 2 of n. The good news is that most of the time we're working with asymptotic notation, so it doesn't really matter. We can communicate with our math friends, and we don't have to talk about this. OK. I need to get another complicated example. And my memory fails me.

OK. So log-- See, I'm using Ln to mean the math natural logarithm. So we still have that somewhere in here. Of log n to the base 100. They made me add all these parentheses in the last section to be clear, so I'm adding them now from the start.

Who wants to help me clean this up? Yes?

AUDIENCE: Log, log n.

VICTOR COSTAN: All right. Wow, one step. Log, log n OK. Can you tell me why, very quickly?

AUDIENCE: So we can simplify out the base and the exponent in there.

VICTOR COSTAN: OK.

AUDIENCE: Then--

VICTOR COSTAN: Base goes out. Exponent goes out.

AUDIENCE: Log, log of something is another [INAUDIBLE]. It's not exactly up there. But can you--

VICTOR COSTAN: Yep.

AUDIENCE: --occassionally [INAUDIBLE].

VICTOR COSTAN: Yep. What does a theory student say when they're drowning? Log, log, log, log. Some of you are laughing already, because know this. OK. What if I want to make this a little bit harder? And I say log [INAUDIBLE] 5 of log n to the 100. You have to do a bit more work now, you can't just use the tricks on the right. Anyone else? Yes?

AUDIENCE: It's still log, log n.

VICTOR COSTAN: OK. It is still log, log n. But for a different reason. Right?

AUDIENCE: [INAUDIBLE].

VICTOR COSTAN: Almost. OK. Maybe not. So the base still goes out just like before. But this guy now goes out here. So I will have this be, sort of like-- Notice that I'm not saying equal, I'm saying like because I got rid of the base. So it's like 100 times log n, which is log 100 plus log, log n.

And this is a constant factor, this grows as n grows. So this is going to become incredibly small as n becomes larger and larger, so I can get rid of it. OK? OK. One more, and this one's going to be less boring.

AUDIENCE: [INAUDIBLE] out into a separate term-- the log of 100? Going from it in the last step, how are you able to cross that out?

VICTOR COSTAN: If my memory-- If my memory works, I think, log of a b is log a plus log b.

AUDIENCE: Gotcha.

VICTOR COSTAN: Am I right guys? Sweet. Good question. I wasn't sure, but now that you asked, everyone else helped me make sure it's right. OK. I'm erasing so much, because we're going to need more room for this. So I have a log again, but this time what I have inside the log is n choose n over 2. Does any brave soul want to come up here and solve this for me? No? You guys are going to make me write for the entire hour?

[LAUGHTER]

VICTOR COSTAN: I'll offer help.

AUDIENCE: It's n-- [INAUDIBLE]

VICTOR COSTAN: So this is--

AUDIENCE: Is that n factorial over --

VICTOR COSTAN: Yep.

AUDIENCE: --n divided by 2 factorial?

VICTOR COSTAN: n factorial over n divided by 2 factorial, times n divided by 2 factorial. Close. And I'll give up another hint, which is that n factorial can be approximated as square root of 2pi n multiplied by n over E divided by n to the power of n. But I think this will make you guys not want to come up and solve this. Yes?

AUDIENCE: OK. [INAUDIBLE].

VICTOR COSTAN: I was-- Is that it? Is that not it? Did I get it wrong? OK.

AUDIENCE: [INAUDIBLE] minus n.

VICTOR COSTAN: I think that's an asymptotic. That's what we'll-- I think we'll get to that. But factorials grow exponentially. So it should be something like this. OK. So we have that nice approximation. Does anyone want to work it out, or will I have to do it? I'll keep that in mind when I'm grading your homework.

[LAUGHTER]

VICTOR COSTAN: OK. So it's log of square root 2 pi n multiplied by n over 2 to the power of n. I'm going to put these two together. So 2 pi n over-- There's an n over 2 there, right? So it's n over 2E times n over 2 to the power of 2.

OK, I can get rid of this guy, because it's a constant. I can get-- No, I can't get rid of this guy yet.

AUDIENCE: In the denominator shouldn't it be 2pi n over 2 under the radical?

VICTOR COSTAN: Yep. 2pi n over 2, so this simplifies. OK. Thank you.

AUDIENCE: Square at the bottom term, you can get rid of the square root on the end--

VICTOR COSTAN: Yep.

AUDIENCE: --and then cancel the some of the exponents.

VICTOR COSTAN: OK. Do you want to come help me?

AUDIENCE: Sure.

VICTOR COSTAN: Thanks so much.

STUDENT: OK. So we can distribute this exponent here to cancel this. And then we get square

root of pi n times n over 2. And it'd be n-- sorry about my handwriting.

VICTOR COSTAN: I think it's better than mine by quite a bit.

STUDENT: Square root of pi n. And this is square root squared, so that cancels . Times n over E to the n.

AUDIENCE: Do you need parenthesis around n over 2E?

STUDENT: Yeah. That's probably good. So, since this is n over E to the n, and n over 2E n. Well since they're to the n, both the n we can give them a single exponent and cancel stuff. So we just get square root of pi n to the times 2 the n, over pi n.

Can do some more cancelling and just get 2 the n over-- Going to simplify this to the n over the square root of n. We can take out the pi, because it's in our natural logarithm. It's a constant factor. And this is log of-- and then-- Since it's a log of, it's just log of 2 to the n. So it's-- say since it's log you can turn it into log of 2 to the n minus log of square root of n. Log of 2 to the n is a much greater complexity than square root of n, so we can--

VICTOR COSTAN: Sweet. Thank you. So see? Complicated formula-- Even more complicated formula came out to be one of the numbers that I promised it would come out to. Thanks for the help. So you can get rid of the pi as soon as you want to. And this is the last step.

I just want to say it again, just make sure everyone heard it. So once you get here, you get this, order n minus log of square root of n, which is-- This is-- This is order n minus one half log n. And the log n is much smaller than n, so this goes away completely. And you get this as your final answer.

OK. Does this make sense? Any questions? Anyone?

How many people are still paying attention honestly? Wow. Nice. You should come to all my sections.

OK. Let's talk about something we covered in class. Let's talk about peak finding. And we'll start with 1D peak finding, and then we'll go to 2D peak finding. So does anyone remember the pseudocode for the faster 1D peak finding by any chance?

ID peak finding. I think everyone said they went to lecture, so I don't have to explain what peak finding is, right? OK. So that's go for it. So suppose I have a few numbers-- 3-- let me write this one-- 3, 4, 5, 6, 7, 6, 5, 4.

So the peak finding problem says there's an array of numbers, and I want to find the local peak. And the local peak is a number that's surrounded by numbers that are smaller or equal to it. So there shouldn't be any larger numbers than it. And I want to find a peak as quickly as I can.

So what's the faster algorithm that we talked about in lecture? Yes? OK. You didn't answer, right, as much, at least so.

AUDIENCE: Binary search?

VICTOR COSTAN: OK. It's exactly like binary search. Yep. Do you wanna help me write the pseudocode? You can tell me what it is.

AUDIENCE: Sure. So, I guess well start in the middle.

VICTOR COSTAN: OK. Start in the middle. If you-- If you write this on your Pset you might want to be a little bit more formal. Someone asked about proofs. I would say something like, start by looking at the middle element of the array. Same thing though. OK. So let's pretend this is the middle. Now it is. What do I do?

AUDIENCE: Then you can check one side in the middle. So it's [INAUDIBLE] the right side.

VICTOR COSTAN: OK. I'm looking at the right side. I see a 7 there, so I know for sure that this is not a local peak. Right? What do I do now?

AUDIENCE: So now you can start over, and using 7.

VICTOR COSTAN: OK. What if-- This would be small. What if I would look at 5 first? Or you should I

look at the other side too?

AUDIENCE: Well you don't have to in this case, but if you-- But if 7 were smaller then you would go and look at the other side, and check to see if 5 is also smaller. In that case you've already found the peak.

VICTOR COSTAN: OK. So I have to look at both sides and make sure-- if none of them is smaller, than I-- I'm sorry. If none of them is bigger, I found the peak. If any of them is bigger then I can go whichever way I want. So I saw my 7, I'm going to cut this in half. And now I look at this. You were saying, right?

So look at the neighbors-- By the way, can anyone understand my handwriting? Is it even worth bothering to write?

AUDIENCE: It's pretty good.

VICTOR COSTAN: OK. I see three nods. I'm going to write for those three people.

AUDIENCE: It's just we haven't taken any cursive in about 12 years. So--

VICTOR COSTAN: OK. Thanks. Not feeling well at all. OK look at neighbors, figure out if you're a peak. You're one of the people that said you can read it, so now there's only two people that can read my handwriting and that I like.

[LAUGHTER]

VICTOR COSTAN: Look at the neighbors. See if local peak.

And if not, figure out which way to go in. And I'll say if not recurse. So what's the running time for this? Suppose I have n elements to start with, that's my array size. I will use t of n as my running time. So to figure out the running time for the whole algorithm, I have to look at the running time for each step and add them up. Right? What's the running time for the first step? Looking in the middle.

AUDIENCE: [INAUDIBLE] the middle elements.

VICTOR COSTAN: 1. 1. I would not say 0. I still have to look at it. Or to be on the safe side you say 1, theta of 1. It's a constant. OK. Now I have to look at the neighbors, and I have to see if one of them is bigger or not. How about that? How much time is left?

I heard the constant. OK. Very soft voice, but I heard a constant there, so I'm going to take it. So this guy is a constant. If it's a local, if you come down. If it's not a local peak, what's the running time for this? n over 2, since--

AUDIENCE: If you FOIL that you divide by [INAUDIBLE], right?

VICTOR COSTAN: Yep. So I'm looking at this guy. It's clearly not a peak. So I'm going to look on the right side, because I see a 7 here that's bigger than a 6. So now I have the same problem, except my array size just dropped to half.

So same problem, because I already have a name for it, except the input size is different. So the running time for this entire step is theta of 1 plus T of n over 2. So the running time for the whole thing is-- I should have said theta of 1 here, sorry. So theta of 1 plus theta of 1 plus T of n over 2. So this is theta 1 plus T of n over 2. So I wrote out my pseudocode, looked at the running time for each step, added them up. Now I have a recurrence, and I have to solve it. Let's solve it. We sort of glossed over that in lectures, so we're going to do it now.

So T of n equals theta of 1 plus T of n over 2. I'm going to expand this using the definition. And theta of one is long, so I'm going to call this c instead. It's a constant, right? So c plus-- expanding this-- c plus T of n over 4. I'm going to expand it again.

And again-- Anyone getting bored of this? So if I expand it a lot of times, I'm going to get i times c, plus T of n divided by 2 to the n. And you can expand it as many times as you want until we see the pattern, but this is the pattern.

So I have a recurrence. What else do I need in order to get a function? I need a base case. So the problem size gets smaller and smaller and smaller, but at some

point it becomes so small that I can solve it just by looking at it. What's that case?

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: OK. So if I have an array of one element, it's a peak. Right? Job done. So T of 1 is a constant. So I want to write this in such a way that this becomes this. So I want to figure out an i such that-- Yeah. T of n divided by 2 to the i equals T of 1. And the easiest way of doing that is to say that n divided by 2 to the i is 1. So i is the inverse of an exponential-- Anyone? AUDIENCE: [INAUDIBLE].

VICTOR COSTAN: Logarithm, thank you. Log n. So I have i, and I'm going to plug it back in here. And I get constant times log n, plus T of 1, which I know is theta of 1.

I'm going to make this theta of 1 again. So theta of 1 times log n, plus theta of 1. And I know that log n dominates a constant, so this is going to be theta of log n. Solved. OK. How are we doing? Yes?

AUDIENCE: In the second to last from the bottom, I don't understand [INAUDIBLE] of c times log n replace i by log n. In the second part of the equation, how did you--

VICTOR COSTAN: OK. So in the jump from here to here, where I said i equals log n-- Right?

AUDIENCE: Yep.

VICTOR COSTAN: OK. So i becomes log n, c stays c. You're wondering about this part, right? So if i is log n, then 2 to the i is n. So I have T of 1, which I have here. So I want this to have this happen. When I'm solving for i here, I set this guy to my base case, so that I can get this to happen.

The idea here is that you're expanding the recursion forever, and you know that at some point you're going to have to stop. When do you stop? You stop at the base case. So you force this to become the base case. OK? OK.

Let's do 2D peak finding. Let's talk about it a little bit.

So 2D peak finding is exactly like 1D peak finding, except you have a 2D matrix instead of a 1D array. Let's see if I can do this right.

So suppose this is our matrix. And suppose these are the rows. Suppose we have m rows and n columns.

Can anyone remind me what the psuedocode for 2D peak finding that we talked about in lecture? So we had two attempts. The first one was a clear failure. It didn't work. The second one worked. And we talked about its running time a little bit. And then [INAUDIBLE] asked you to think of a variation to the second one. So let's do those straight up second version of the pseudocode without the variation. So 2D peak finding.

What is the algorithm? All right. One person was not asleep in my section at the end of lecture. I'm proud.

AUDIENCE: OK. Well, Which one of us? We both spoke at the same time.

VICTOR COSTAN: You had your hand up first though.

AUDIENCE: So you just pick a middle of the column.

VICTOR COSTAN: OK. And then you find the peak there.

AUDIENCE: And then you shift the neighbors, I think.

VICTOR COSTAN: Find peak neighbors.

AUDIENCE: Neighbors in the same room?

AUDIENCE: Yeah, yeah.

VICTOR COSTAN: That's the horizontal line, right? OK. So what do you mean by find the peak here?

AUDIENCE: Like within the column?

VICTOR COSTAN: Yes. So find the peak using what we talked about before?

AUDIENCE: Yeah.

VICTOR COSTAN: OK. Find 1D peak.

So you're thinking ahead. You're thinking of the variation that [INAUDIBLE] asked you guys to think about at home. I'm glad you thought about it. That's good. But what's the one that we had in lecture? I don't want to think about-- I don't want to give you guys the answer yet. So what did we say we were going to do in lecture? Yes?

AUDIENCE: I think we just picked a spot,

checked the neighbors and kept walking uphill. That makes sense. Every time we found a neighbor that was bigger, we moved to that spot. Checked its neighbors.

VICTOR COSTAN: So this is the-- This is the first algorithm that's slow. It works. You're going to get the correct answer. But it's slow because if you have some handily crafted matrix you can go like this, and you're basically going through the whole matrix. Yes?

AUDIENCE: I think there's like an algorithm that's find a-- look through the column for its maximum, and then look to either side.

VICTOR COSTAN: OK. So instead of finding a 1D peak, we're going to find the maximum value.

OK. And this is correct, and let's look at its running time. Does anyone want to help me? So this is T of n m. Right? Then rows, then columns. So what's the running time for the first step? Come on. This is the easy one. Yes? The constant. Perfect.

OK. What's the running time for finding a maximum amount of n numbers?

AUDIENCE: n.

VICTOR COSTAN: OK. Perfect. And suppose we do that here? Suppose these rows don't exist. Suppose this is my entire matrix. And I'm here, and I found the maximum.

What do I do? I look left, I look right. If both of them are smaller, then it means I found my peak, because I know that number was a maximum in my column. Now if that's not the case, if I find a number that's bigger, I'll go in that direction. Does that sound reasonable?

OK. So maximum for my column, and I'm going to go right. So check neighbors, and I have the same two conditions there.

If it's a peak, then I'm happy. Otherwise recurse. So what's the running time if I have to recurse? How does the new problem look like?

It's an identical problem. I still have a matrix, except-- What changed? So suppose I got here-- and this is the middle of my matrix-- and I see that this guy is bigger, so I'm going to recurse to the right side. What happened to my problem size? OK?

AUDIENCE: Sorry I'm not answering your question, but shouldn't find x be theta of m, not n? Is that [INAUDIBLE]?

VICTOR COSTAN: Let me see.

AUDIENCE: I can't tell

VICTOR COSTAN: You are right. Thank you for paying attention. Theta m. Sweet. That would have tripped me up later. Thanks for saving me.

AUDIENCE: OK. So what was your question? Sorry.

VICTOR COSTAN: So the question is, if I have to recurse--

AUDIENCE: Uh-huh.

VICTOR COSTAN: --what's the running time? So I'm-- If I have to recurse, I'm in this array that was originally n by m, and I'm only going to look at the right side. How many these-- how many rows do I have in the [INAUDIBLE]? Start to the easy one.

AUDIENCE: m.

VICTOR COSTAN: All right. Same as before. How many columns?

AUDIENCE: n over 2.

VICTOR COSTAN: n over 2. Sweet. So there recursion for this is theta of 1 plus theta of m plus T of n over 2 and m. I know that a constant can be absorbed by this other guy, so it's theta of m plus T of n over 2 m. How would I solve this? What's the first thing that we need? Yes?

AUDIENCE: Need to figure out the base case.

VICTOR COSTAN: All right. Excellent. Base case. What's the best-- What was the base case?

AUDIENCE: T of 1 comma T of n-- m. Sorry.

VICTOR COSTAN: Nice.

AUDIENCE: Equal to O of m.

VICTOR COSTAN: OK. Sweet. So if I only have one column, then I can do-- I can find the maximum and I know for sure that's going to be a peak. Job done. That's my base case. So I'm going to reduce the problem size in half, until I get to a single column. Then I find the peak there, and I'm happy. How would I solve this recurrence?

It's a one minute answer, because I think that's exactly how much time we have, as a hint. Come on guys, this looks familiar. Yes?

AUDIENCE: Going to be something like the recurrence over there--

VICTOR COSTAN: Exactly.

AUDIENCE: [INAUDIBLE] of just n.

VICTOR COSTAN: Exactly. m never changes, right? So this is basically a function of n. So if m never changes, and I pretend that it's a constant, this looks exactly like the recursion-- the recurrence that we solve before. T of n is a constant, plus T of n over 2. So I'm going to get that constant, multiply it by log n, which is the solution that I had before. So the running time is theta of m log n.

All right. I think I have 30 seconds, so what if I would choose-- What if I would try to do this algorithm instead? It's another one minute question. So I guess I'm cheating. I'm taking time away. Sorry. So what if I would be finding a 1D peak instead of finding a maximum?

I'm looking at the running time, because if the running time is going to be the same, it's not even worth thinking if it's correct or not. If it's a more complicated algorithm with the same running time, why care about it? So is the running time going to be better? OK. I heard a nod. Why?

AUDIENCE: Won't it be log m times log n?

VICTOR COSTAN: All right. So 1D peak is exactly what we have there. Thank you. 1D peak is exactly like you said, log m. So this constant here becomes this constant here, so the algorithm would be a lot faster.

As an order of magnitude idea, if your m and n are one million each, then log n is 20. So that's a 50 time max improvement. And these are real input sizes you might get. And that's it. Thank you guys.