[SQUEAKING]

[RUSTLING]

[CLICKING]

ANKUR MOITRA: So here, the way to think about modular arithmetic is that it's about relationships between numbers. So some of these, you've already seen before and are pretty familiar. So, for example, if I take an even number and add an odd number to it, then I definitely get an odd number. And if I take an even number and add an even number, I get an even number.

So even though these things are familiar, even, from elementary school, we're going to talk about generalizations and go a lot further with these ideas. Really, the way to think about this simple example is it's what's called arithmetic modulo 2. So by that, I mean you're doing arithmetic, but you're only keeping track of the remainders when you divide by 2.

So for today's lecture, we're going to have the following key definition that's going to take this idea much further, which is we're going to say that two integers, a and b, well, we're going to write the following expression a congruent to b mod n. So this is pronounced as a congruent to b mod n. And we'll write this expression if and only if a and b differ by a multiple of n.

So that's it. That's our key definition that we're going to explore today. But we're going to see that there's a lot of meat to this in terms of understanding how to work with modular arithmetic. So another way that you'll sometimes hear modular arithmetic described is as clock math. So I'll explain what that metaphor means in a minute.

But let me tell you the key result that we're going to use to reason about modular arithmetic. And this will also be something that's familiar from long ago. But we're going to state this a little bit precisely. This is, really, a corollary of what you learned in elementary school, namely, long-form division.

So what I claim is that, for any integers a and n where n is strictly positive because it's our divisor, what I claim is that there is a unique pair of, again, integers q and r-- and these are suggestively chosen because q is the quotient, and r is the remainder-- with the following property-- we can write a as q times n plus r.

And the crucial thing is that this remainder, r, is between 0 and n and cannot be as large as n. So it's really between 0 and n minus 1. So this is really the corollary of what we learned in elementary school about long-form division. I can take any integer and some divisor n, and I can write a as q multiples of n and then whatever I have left over. And the crucial thing is what I have left over is an integer between 0 and n minus 1.

So these are just the basic facts that will help us make sense of modular arithmetic. And let's go over some basic facts about digesting what this expression means-- what are all the equivalent ways to think about what this expression is saying, that a is congruent to b? So what we're going to do is we're going to learn the language of modular arithmetic. It's a little bit like relearning arithmetic from scratch but trying to remember what rules you're still allowed to do and what rules change.

So let's be careful with this. Let's prove our first key fact, and this will be an extremely powerful fact. So what I claim is that this expression, a is congruent to b mod n, well, we know what this expression means. a and b differ by a multiple of n. I claim that this is true if and only if the remainder of a divided by n equals the remainder of b when I divide by n.

So this is a very intuitive fact. This is what was going on right here is that we think about evens as being a set of numbers that are congruent mod 2. And that's if and only if the remainder when I divide by 2 is zero. And something is odd if and only if its remainder is 1 when I divide by 2. But this is true for modular arithmetic even more generally than this. So this gives us a powerful way that we'll use this fact later to great effect for talking about the Euclidean algorithm. But for right now, let's just make sure that we're on the same page and that we can understand why this fact is true.

So we can prove both directions of this implication. So let's prove the backwards direction. So let's suppose that a is equal to qn plus r and that b is equal to q prime n plus r. So r is the remainder, so I'm just assuming that the remainder of a divided by n is the same as the remainder of b divided by n because they both have the same extra term for the remainder.

And now what we can do is we can just remember what this expression means. We can look at a minus b, the difference between a and b. And we can write this as q minus q prime times n. And the r's cancel. So this gives us what we're looking for. The statement that a and b are congruent mod n is equivalent to the statement right here that their difference is an integer multiple of n and q minus q prime is an integer.

So we're good. We proved one direction. So this proves the first implication, that if the remainders are the same, then they're congruent mod n. And we can prove the converse direction, too. So conversely, let's suppose that a is equal to qn plus r, and b is equal to q prime n plus r prime.

Now, since a is congruent to b mod n, we know that a minus b is supposed to be equal to an integer multiple of n. And then we can remember what a and b are by assumption. And we can write this out as q minus q prime times n plus r minus r prime.

And so now we can just focus on this expression. We know that a minus b is a multiple of n. And we can write the expression alternatively this way. But the key point is what's happening right here, that this term right here, r minus r prime, is an integer that's strictly larger than minus n and strictly smaller than n.

So if k, if a minus b can be divided through by n-- we get some integer multiple-- the only choice-- n divides evenly here, and we get an integer-- the only way that a number that's between minus n and n and is strict-- these inequalities are strict-- and it divides through and gets an integer is if r minus r prime equals 0, in which case the two remainders are the same.

So that's our key fact right here. It's just that we can use the division algorithm to connect very nicely with modular arithmetic. The statement that two numbers are congruent mod n is really a statement about what happens when we run the division algorithm on a versus b. We might get different quotients, but they better have the same remainder. So are there any questions? Does this make sense? Yeah? OK.

So now let's think about how we can visualize what's going on. So the visual to keep in your head is of a clock. So we can think about something like 12, which is congruent to 5 mod 7. And what's really going on is that we take the integers 0 to 6, and we put them evenly spread out along our clock.

And now what's going on is when we take an integer like 5, we start off at 0, and we go around the clock until we hit 5. And if we take an integer like 12, we go around the clock, but we're going to loop one full time around, and we'll end up at the same, exact place. So that's sometimes why you'll hear it referred to as clock math is because, unlike ordinary integers here, we're not actually distinguishing between 12 and 5. These are in the same equivalence class because they land on the same position when we go around the clock.

So this is the basics of modular arithmetic, but so far, we've only talked about addition. So the other piece we should talk about is multiplication. And then we can get to some action.

So when we think about modular arithmetic, we already know how to add. And we can also ask how we multiply modulo n. So it turns out that you can kind of just do it. So if I have something like 10 is congruent to 3-- again, modulo 7-- I can multiply this by 12, which is congruent to 5 modulo 7.

And one thing you can check is that you can just do multiplication as usual. You can multiply 12 by 10, and I get 120. And on my right-hand side, I get 15. But really, 15 is congruent to 1 modulo 7 because I can take away two multiples of 7 and get back down to 1.

So basically, when you're doing addition and multiplication, you can always just multiply and add the way that you're familiar with. Just, at the end, after you're done, you take out as many integer multiples of the modulus as you need to to get back to an integer that's between 0 and n minus 1.

So a lot of things work out very nicely. But the way to think about this, really, is if you were to try and unwrap what's behind this notation, the statement that 10 times 12 modulo 7 is congruent to 1 is really the statement that a number with remainder 3, all happening modulo 7, times a number with remainder 5 modulo 7 gives a number with remainder 1, all modulo 7.

So it's good to not get confused by what's happening in terms of this expression. What this expression really means, when I take any number, like some number like 10, whose remainder is 3 modulo 7, and I take any number-- say 12-- whose remainder is 5 mod 7, and I take the product of those two numbers-- I get this number on my left-hand side, 120-- and that number should have remainder 1 when I take it with the modulus 7.

So that's really what's going on is it's the same generalization of what I wrote down right here is that you can forget what the actual even number was. All that matters is what its modulus is. Mod 2, I can forget what my odd number is. All that matters is that it has remainder 1 modulo 2. And I can still tell you something about the remainder when I add up those two numbers, even though I've forgotten the rest of the action of how many times around the clock I've gone.

So that's the key way to think about it. So if you've seen it before, this is very easy. If you haven't seen it, this takes a little bit of wrapping your head around. So are there any questions about modular arithmetic, or does this make sense? Good? OK.

So I'm trying to give you a lot of different ways to think about it. I'll also mention that you can think about these things abstractly, too. So we'll talk more about this in the next lecture, but when I'm working modulo 7, then I can really think about it as something called a group that we'll talk a lot more about. And, really, I can just think about the elements of that as being all of the possible remainders.

And I have this table of how, exactly, to add up two numbers that belong to this set because when I add up remainders of 0, I get back to 0. When I add up 2 with 2, I get back to a remainder of 4. But see, I have interesting things happening here. When I take 4 and I add 4, I go back to the element 1 because that's the new remainder I would get when I take it modulo 7.

And the same type of thing happens here. This is our multiplication table. When I take something like 3 times 3, that'll give me 9. And when I subtract off the multiples of 7, I'm going to get back to 2. So, really, what's going on is that we're just redoing our addition and multiplication tables, just in the context of only keeping track of what the remainders are. Make sense? All right.

So what I'm going to do is I'm going to tell you some more basic facts about modular arithmetic, and then we're going to do a demo just to give you a bit of a hint about why modular arithmetic is important. So, as I promised, many of the things that you're used to with ordinary arithmetic go through just the same. And I'll point out the things that don't.

So one of the properties that we know and love is associativity. If I have some number modulo 7, I can add two numbers, b and c, modulo 7. And then I can add to their result a, all happening mod 7. And I claim that that's the same thing as parenthesizing the expression and the other way. It's the same thing as a plus b. And then I add c, all happening mod 7.

A lot of the other basic facts we're used to go through, as well, like commutativity. So, in particular, a plus b modulo 7 is equal to b plus a. That's the same thing as even plus odd. It doesn't matter what order I add them up. I still get the same thing, which is an odd number.

And last but not least, we still have the distributive property, which tells us how multiplication and addition interact. So if I'm multiplying over z7 a times the result of b plus c, That's the same thing as a times b plus a times c. So almost everything works exactly the way that you're used to.

So before we go too far into modular arithmetic, because we're going to be doing a lot of it today, I wanted to give you a little bit of a teaser for later about why we care about modular arithmetic.

So it's not totally obvious, but modular arithmetic is a great way to accomplish a lot of things, especially in computer science. So we talked a bit about areas like cryptography in a previous lecture. Cryptography is based on modular arithmetic, so this will be a very important application for us later in the class. There are also some other really fun applications that also are from other areas of cryptography, which have to do with things which were actually invented here at MIT about zero-knowledge proofs.

So I'm going to need some help from the audience for this one. But let's imagine that you're home on a weekend, and you want to watch an embarrassing movie with your friend. Now, you have some favorite movies you might want to see, but you don't want to admit to liking that movie unless your friend likes it, too.

This is a common problem, right? Maybe if you both like the same embarrassing movie, you would then watch it, and that's fine. But otherwise, if you don't both want to watch the movie, you want some kind of plausible deniability-- I don't like that movie.

So what are some examples of embarrassing movies that you guys actually like that you might be a little embarrassed to admit? Don't be shy. You guys don't like any embarrassing movies? What? [LAUGHS] We can't move on with the lecture until you guys give me something here.

AUDIENCE: I don't have anything else.

ANKUR MOITRA: OK. What are embarrassing movies your friends like-- your friends might like?

AUDIENCE: *Shrek.*

ANKUR MOITRA: *Shrek?*

AUDIENCE: *Shrek* is good.

ANKUR MOITRA: *Shrek's* OK. All right.

AUDIENCE: *Barbie,* maybe.

ANKUR MOITRA: *Barbie,* OK.

AUDIENCE: The *Barbie* movie's great.

ANKUR MOITRA: [LAUGHS] All right, we're going to have a fight break out.

AUDIENCE: *Morbius.* This is just the correct answer.

ANKUR MOITRA: *Morbi--?* I have no idea what that is.

AUDIENCE: It's constantly [INAUDIBLE] because it's so bad.

ANKUR MOITRA: OK, *Morbius.*

AUDIENCE: Oh, wait. *The Room.*

ANKUR MOITRA: *The Room,* OK-- *The Room.* I was going to go with-- what's this-- OK, I don't know. All right, let's go with *Morbius.* So can I have two volunteers from the audience, and then I'll explain the game. Who's going to come up here? Don't be shy. Yeah? All right, one more. Let's do it.

So let's explain how modular arithmetic and clock math is going to help us answer this question. What I'm going to do is I'm going to take five cards. Three are red. Two are black. Let's pretend that I don't keep track of the fact that it's a 2, and I just know the color of it.

So what I'm going to do is I'm going to give each of you one black and one red card. And then what's going to happen is I put down this one red card in the middle. Now, what they're going to do is they're going to decide whether they like the embarrassing movie or not, which they want to see on the weekends, which I guess is *Morbius,* which I should look up after class.

And if they like it, what they're going to do is they're going to put their red card closest to the red card in the middle and their black card on the outside. And if they don't like it, they're going to put their black card next to the red card and their red card on the outside.

And so what I'm going to do then is I'm going to stack up the cards on top of each other. And then I'm going to have them both cut the deck randomly. So that way, it's a random cyclic shift when they put it together.

And then, what I claim is that if they both like the movie *Morbius* and they want to see it, what's going to happen is we're going to have two red cards in a row, cyclically. And if either of them doesn't like it, then we will not have three red cards in a row. We will have two red cards in a row. But even the person who liked it and the other person didn't, they'll have plausible deniability because all of those, after the cyclic shifts, it'll all look the same.

So let's try it out. Do you guys want to see *Morbius* over spring break? So I won't look. Put down your cards, and then we'll see if we've got a match. All right. And just for security purposes, I'm going to ask each of you just to cut the deck. So just take up some number of cards, and put them on the bottom. Perfect.

And then cut the deck randomly. We'll pretend we didn't see how they cut them. And what I'll do is I'll put them out in order. And unfortunately, we don't have a match. So can you tell me what you guys answered?

AUDIENCE: Oh, I answered no.

AUDIENCE: Yes.

ANKUR MOITRA: You put yes. OK. So even though he put yes, he can save face and say, I never liked that movie, anyways. So this is what's called the zero-knowledge proof. Thanks for your help. And it uses just clock math types of things.

So it turns out that, once we have the basics of modular arithmetic, there'll be some amazing things we can do with it, like being able to privately compute functions, be able to execute cryptographic schemes. But for this lecture, what we're going to be doing is we're just going to be building up the basic mechanics before we can apply it in action in some of the later lectures in class.

So now, let's roll up our sleeves. All of this stuff was pretty natural. It works the way you would expect it to. But now, we're going to get in to one piece of the puzzle that makes it really interesting and different than ordinary integer arithmetic.

So let me tell you the key question, which we're going to spend a few minutes answering. What I claim is if I look at that table, I'm going to have some examples of elements that have multiplicative inverses. I'll tell you what I mean by that, but my key question is going to be, when I'm working, say, mod 7, does every element have a multiplicative inverse?

So let me be concrete. So let's say we're working mod 7 for the time being. We can take 2 and 4, and we can multiply them. And, as usual, the way we do modular arithmetic with multiplication is we just do it and then subtract off the multiples of 7 later. So when I take the element 2, and I multiply it by the element 4, what I get is I get the element 1.

So that's an example of a multiplicative inverse because what's going on is if I have a number multiplied by 4, I can get rid of it by multiplying the whole thing by 2. The 2 and the 4 will cancel because they multiply to 1. And then I get back the thing I started with.

So this is the key to a lot of what we're going to be doing later cryptographically is because multiplying by a number in modular arithmetic serves of jumbling up the numbers you started with. So I might have some secret message, and then I want to jumble it. And I could try multiplying by some other number, like 4, but then I have to have some way of unjumbling it. And in this case, that's a multiplicative inverse. Now, that scheme is not literally going to work, but that's the intuition behind what's going on.

So in this case, I got lucky, or maybe I didn't, that when I multiplied by 4, there was a multiplicative inverse, which is 2. Does 2 have a multiplicative inverse? What is it?

AUDIENCE: 4.

ANKUR MOITRA: 4-- that's right. Because of commutativity, it doesn't matter what order I multiply them in. So 2 and 4 are each other's multiplicative inverses. I could look at my table and write down the full table of all of the elements. And I could check that every number has some kind of multiplicative inverse. Like, 5 has a multiplicative inverse of 3. And if this table continued, 3 would be the multiplicative inverse of 5.

But when I get to larger and larger moduli, it's not always possible to write down this gigantic table and verify that, in each row, there is some column where it's got a 1. And so what I can ask is whether every element-- and by this, I mean, just to be clear, I mean every non-zero element-- 0 won't have a multiplicative inverse-- but does every non-zero element have a multiplicative inverse?

So this will be a key question which occupies us for a little bit of the rest of the lecture is answering this question because we're going to build up some mathematical theory of not only why does every element have a multiplicative inverse when we work modulo a prime, but how exactly I would find it without actually writing down the entire table.

So that's our plan of action. And let's go back to-- we're going to use this fact I mentioned, the fact that a is congruent to b mod n if and only if the remainders are equal. We're going to use this to great effect. So let's just start with some very natural definitions, and it'll lead us to the answer to this key question.

So first of all, I'm going to define another key quantity, which is called the GCD, the Greatest Common Divisor. So if you give me two integers a and b, well, the greatest common divisor is what you think it is. It's the largest integer that divides both.

So it could be 1, for example, if there is no integer larger than 1 that divides both. But we'll take this as our definition, the GCD. And, really, it's not so obvious right now, but the following question is going to help us answer this problem about multiplicative inverses.

So I can ask the question, how can we compute the GCD? So there's a terrible way to do this, which is I could just take the smaller of the two integers-- let's say it's a-- and I could try all the integers between 1 and a and check. That might work for small numbers, but you're going to have a very hard time once you start working with gigantic numbers, things that are, like, 32 digits because I'll just have to try too many things.

What I claim is that there's a very fast algorithm for computing the GCD. In fact, it's one of the first algorithms that was ever discovered, so it goes back to the ancient Greeks. And we're going to build up to this algorithm with a few key structural lemmas about how the GCD works.

So let's start with the first lemma. So let's assume that b is at least as large as a without lost generality. Otherwise, we'd just switch them. What I claim is that the GCD, the Greatest Common Divisor, between a and b is actually the same as the GCD between a and b minus a.

So this is our first notion of progress. We start off with two huge integers, a and b. We want to compute their GCD. And I claim that their GCD is the same as two not quite as huge integers because now, potentially, I've made one of the integers smaller. So this is our first notion of progress. Once we have this lemma, we'll be able to make a ton of progress, but let's start by proving this lemma. And then we'll prove a souped-up version of this lemma.

So the proof for this is very simple. Let's suppose that k divides a-- this is the expression for k divides a-- evenly into an integer. So a is a multiple of k. And let's assume that k also divides b. Then what I claim is that k must also divide b minus a.

Why is that true? Well, what does this mean? When I take a divided by k, I get an integer. When I take b divided by k, I get an integer. And so, then, if I take k-- if I take b minus a and divide it by k, I get an integer minus an integer, so I definitely still get an integer.

But let's think about what this means. What this means for us is that the GCD of a and b is a lower bound for the GCD of a and b minus a. So this part is easy. I know that k divides b minus a.

How did I get to this statement right here? How does interpreting the statement tell me that whatever the GCD is for a and b, the GCD of a and b minus a is at least as large? Can anyone help me out? So you're with me up to here. How did I get this statement? How did it follow from this? Any ideas?

Let me give you a hint. So, mechanically-- I'm not telling you I know what the GCD of a and b is. I'm telling you, whatever the GCD is-- let's suppose it's some integer k-- that the GCD between a and b minus a must be at least as large. How can I guarantee that fact? Or ask a question. So either give me an idea or ask me a clarifying question. But this is important.

AUDIENCE: That anything dividing the left-hand side by [INAUDIBLE] side.

ANKUR MOITRA: Yes, perfect. So one way to think about this is, what am I saying here? Whatever this GCD is-- let's imagine that it's k-- well, then, I know that k, from the statement, if k divides both a and b, then k must also divide a and b minus a. That's literally what I just proved.

So that means that k also is a candidate for a divisor, for a and b minus a. So whatever the greatest common divisor is, it must be at least as large. It must be-- maybe there's something even bigger than k, but I know it's at least k. So are there any questions about that? Does that make sense now? So this is really-- yes?

AUDIENCE: [INAUDIBLE] k is the GCD of everyone, as sort of this one?

ANKUR MOITRA: I said that this is-- so even though this is just three lines, it's a bit subtle. So here, what I said was that if you give me any integer k that divides both a and b, it also divides b minus a. So, in particular, now when I set k to actually be whatever the GCD is, then I know that that same k had also better divide b minus a. So it's a candidate for the GCD of a and b minus a. So even though this is just a couple lines, the logic is a bit subtle.

So this proves one direction, but really, we want to prove that these two things are equal. So let's prove the converse. So conversely, let's suppose that k divides a and k divides b minus a. Well, then what I claim is that k divides b. And that's the same proof as before, what happened here.

So if k divides a, that means I take a divided by k, get an integer. b minus a divided by k, I get an integer. And b is just b minus a plus a. So that is the sum of two integers when I divide by k, so the statement is true. And now, this allows me to prove the inclusion the other direction.

So what this implies is that the GCD of a and b minus a is a lower bound for the GCD of a and b. Why? Because whatever this integer k is, the largest integer that divides both a and b minus a, I just proved for you that that same integer had better divide both a and b. So it's the same logic as above.

So this is one of the key proofs for today. So again, I will ask, are there any questions about this proof? We good? Speak now, or hold your peace until the next office hours or something. All right. So let's do a souped-up version of lemma 1 because once we have lemma 1 and we understand this logic, we're off to the races. So now we'll be able to get a lot of cool things immediately.

So lemma 1 didn't make that much progress in the grand scheme of things because I started off with two integers a and b, and I made b a little bit smaller because I turned b into b minus a. That's not a huge amount of progress, but let's take a leap instead of a step.

What I claim, same assumption, so b is at least a. Then what I claim is that the GCD of a and b is equal to the GCD of a and the remainder of taking b and dividing it by a.

So now I made a huge leap. I'm not just subtracting off 1 multiple of a. I'm actually making b way smaller, potentially, because I'm dividing by a and just keeping track of whatever the remainder is. Does anyone have any intuition for why this kind of statement should be true? And my hint is the division algorithm that's very conveniently right above it.

Let me give you a hint. So let's say that we write b as equal to q times a plus r because that's what we get when we do the division algorithm. And so what I'm saying is that I can replace b with r. So what I'm doing, really, is I'm not subtracting off one copy of a, but I'm doing something a bit more aggressive. So does anyone have any hints about how this proof ought to go? How should I do it? Yes?

AUDIENCE: [INAUDIBLE]

ANKUR MOITRA: Perfect. That's exactly right. So it's just a repeated application of lemma 1. Perfect. So by the division algorithm, which is conveniently right above, well, what I can do is that the GCD of a and b is equal to the GCD of a and b minus a, which is equal to the GCD of a and b minus 2a, the way down to it's equal to the GCD of a and b minus qa, which is, of course, the same thing as r, the remainder. So it's just the repeated application. So why do I keep subtracting off one multiple of a per time? Instead, I can just jump to the end and do division.

And now I make a huge amount of progress because we won't prove the progress bound, but it's very easy to prove. The point is that if a and b are really close to each other as integers, then I'm going to make a huge amount of progress in one of my two steps. If a and b are very far from each other, all of a sudden, the larger of the integers goes down to being as small as a.

So either way, what ends up happening is you get this gigantic speedup in the algorithm. But let's think about just the numerical example to see what this algorithm looks like. And this numerical example is really just the Euclidean algorithm in action.

So the Euclidean algorithm works as follows. All you do is the GCD of ab is equal to the GCD of a and b1 where b1 is equal to the remainder of b and a. And then this is equal to the GCD of a1, b1 where a1 is equal to the remainder of taking a and dividing by b1.

So we just alternate back and forth. We divide one of the things by the other, take the remainder. And now we take the smaller of the two, which now, in this case, is b1. And we take a divided by b1, and that gives us our new a1. So we just keep alternating back and forth until the end.

So let's actually see a numerical example of this, just to make sure that we're on the same page. So in action, let's do it for some large integers. So we could look at something like the GCD of 46 and 360. Kind of hard to do in your head, but if I were to do out the division algorithm, well, the first step is to take 360 and divide it by 46. And what I'll get, the remainder is going to be 38. So I replace b with its remainder when it's divided by a.

And now I just divide things the other direction. So if I take 46 divided by 38, I'm going to have 8 left over. So I'll get that this is equal to the GCD of 8 and 38. And then I'll divide things the other direction. I'll get the GCD of 8 and 6 because I'll have 6 left over when I take 38 and divide it by 8. And then I'll take the GCD again, and I'll get 2 and 6. And now I get no remainder, the GCD between 2 and 0.

So my Euclidean algorithm stopped. But what's my answer? Two. That's exactly right. Because the key point is that what lemma 2 tells us is that, in each step, we're not changing the GCD. And what is the greatest common divisor between 2 and 0? It's just 2 because 0 gets divided by everything perfectly fine.

So that's the Euclidean algorithm, which we now understand. And what I claim is that it's actually going to help us answer that key question. It's not totally obvious why it does and what it has to do with it. The key is, really, taking this algorithm and doing more bookkeeping. And that's called the extended Euclidean algorithm.

So the key point behind this algorithm is we're going to keep track of how we got here. So, for example, when I start off with the state 46, 360, because I want to compute the GCD between these two things, instead of just replacing 360 with its remainder, I'm actually going to keep track of what integer multiple of 46 I subtracted off to get the remainder.

So that's the key point is that my next state in this transition, 46 stays the same. But instead of just having 38, I'm going to keep track of it as 360 minus 7 times 46. Why am I doing that? We'll get into that in a second.

But now, same thing happens here. So when I had this number, which is 38, and I knew that I had to subtract off one copy of it from 46 to get my remainder 8, so I'm just going to subtract that off wholesale, subtract off this entire expression. And what this will give me is it'll give me 46 minus 360 minus 7 times 46. And then, on the other side, I still have 360 minus 7 times 46.

So this seems kind of strange because I just took a beautiful algorithm that's very simple to understand and analyze, and I just made it a lot uglier by keeping track of all of these things. But the key point is if you look in these steps, the intermediates that I have are all integer linear combinations of the two things I started off with. Originally, I have 46 and 360. And all of the numbers that I get in between are some kind of integer times one of the numbers, 46, plus an integer times the other number, 360.

So at the end of the day, when I get the answer of 2, what I've really done is, when I keep track of these integers along the way, I've found some integers, s and t, so that 46 times s plus 360 times t is equal to 2. And my extended Euclidean algorithm will not just compute that the GCD is 2 but it will find the integers s and t that make that be true.

And so what's really cool about this expression, if you think about it, let's say you didn't trust me. Maybe I'm certainly prone to making algebra mistakes, like everyone else. Maybe I made some algebra mistake along the way here, and I just tell you the answer that I got, 2. One thing you can do is you can check that 2 actually is a divisor of the two things I started with.

So if I have 2 as my candidate answer, I can check that it really does divide both 46 and 360. But how do I guarantee you that there's not a bigger divisor of both of them? And so the key point is right here.

Imagine there were a larger integer that divided both of them-- 4, 6, something like that. Imagine I had some integer K that was larger than 2 that divided both 46 and 360. I claim this expression proves for me that it cannot be a divisor of both because if k really did divide both of these things, 46 and 360, when I take the left-hand side and divide it by k, I'd get an integer times an integer plus an integer times an integer. That's an integer.

But my right-hand side is not an integer. So this is actually a certificate that your answer cannot be any smaller. So you don't even have to trust my work because, at the end of the day, the s and the t I find are the proof that I found the largest possible GCD. Does everyone understand that? OK.

So now, let's connect it back to our key question. And I'll just state this as a lemma first. This is really the thing which came out of our proof is that, given any integers a and b, what I claim is there exist integers s and t with the property that sa plus tb is equal to the GCD of a and b. And that's just the abstract version of what we just did in this specific example.

But now we're in good shape to answer this question. So let's return to this, and let's ask, why does 4 have a multiplicative inverse mod 7? Well, let's think about it in the context of the Euclidean algorithm. So the GCD between 4 and 7 is what? 1 because 7 is prime.

Now I can use this lemma. If the GCD between 4 and 7 is 1, What it means, by lemma 3, is that there are integers s and t so that 4s plus 7t is equal to 1. And what I claim is that this implies that 4 has a multiplicative inverse. We know the answer is 2. But from this expression, what I claim is that my multiplicative inverse of 4 is s. When I take 4 times s, I'm going to get 1 minus 7t.

So it's actually 4 times s is congruent to 1 modulo 7 because all of those integer multiples of 7 go away. So that's actually the proof of why not just 4 has a multiplicative inverse mod 7. Why does every number that's nonzero have a multiplicative inverse mod 7?

It's just because, for any other number b here, I know the GCD between b and 7 is 1 because 7 is prime. And I know that means there's an expression for b-- sorry, bs plus 7t equals 1. And in that case, s would be my multiplicative inverse. So even more generally, the GCD between b and 7 is 1. This is true for any nonzero b. And this would then imply by lemma 3 that b times s plus 7t equals 1. And that means that this expression right here, s is the inverse.

So this actually answers all of our questions. So we had this question, does every element have a multiplicative inverse? That's true, as long as the modulus is prime. I had this question, how could I compute the multiplicative inverse? So if I gave you some large prime p and I gave you some nonzero element a, how would I compute the inverse of a mod p? Does anyone have any ideas? The answer is on the board someplace. Yeah?

AUDIENCE: Use the extended algorithm.

ANKUR MOITRA: Perfect. That's right. So I would use the extended Euclidean algorithm. I would apply it with not 46 and 360 but a and p. And my point of doing this isn't to compute the GCD. I know the GCD is 1. My point is to find this expression, the s and the t. And then that s that I find will be the multiplicative inverse. That's exactly right.

So this answers, basically, all the questions we could want. In fact, there's even a more subtle question, which we can answer, too. If I have a modulus that's not prime, like 6, it turns out that 5 has a multiplicative inverse mod 6 because the GCD of 5 and 6 is 1. So you'll actually have a multiplicative inverse when you don't have a prime modulus just for the things that don't share a common factor.

So this was the main part of lecture, but now what we're going to do is we're going to do one last cool application of modular arithmetic in the last few minutes of class. And this is another really important topic, tons and tons of applications. So I'll tell you the math riddle. And then I'll tell you some applications of this once we solve it.

So let's just start off with a riddle about modular arithmetic. So let's say we've got a treasure chest, and it's got gold coins. And let's imagine I promise you that if this treasure is divided into groups of 5, I'm going to have 4 coins left over. And let's imagine same thing. If it's divided up into groups of 7, what I'll have is I'll have 1 coin left over.

So there are many possible answers to how many gold coins there could be. But what I'll be interested in is what's the smallest number. So what is the smallest number of possible coins? So that's our key riddle for the rest of lecture. Of course, this riddle is going to lead to a very interesting statement in its own right that's powerful.

And what I claim, what's really going on here, is that we have a system of equations. But the key point, which is different than the system of equations you've seen before, is that these system of equations are really in modular arithmetic. So translating this, so we want to solve for n, which is the smallest number of coins that could be the answer.

So we know that n is congruent to 4 mod 5. That's literally just using the definition because what that means is just that 4 is the remainder when we divide n by 5. And the second statement is that n is congruent to 1 mod 7.

So it turns out that we can guess the answer here, but I'll tell you more formulaically how to find it. So if you take, for example, 29, then this works. It solves the system of two equations in different moduli. But, actually, there's a much more general result. I guess today is the day we're doing ancient results. So this, similarly, like the Euclidean algorithm, is one of the oldest algorithms. This is a fairly old statement, too. It's due to a Chinese mathematician, Sun Tzu Shuan in the fourth century.

And what I claim, and we'll prove this, so suppose a and b are relatively prime. By that, I mean they don't both have to be prime, but they share no common factors besides 1. Their GCD is 1 between them. So imagine that a and b are relatively prime. a and b will play the role of 5 and 7 here. So just to be clear, this means GCD of a and b equals 1.

And we have some unknown n that's between 0 and ab minus 1. Then what I claim is that if we think about the system of equations, n is congruent to r mod a and n is congruent to s mod b, then what I claim is that there is exactly one solution.

So this is somehow exactly what you could hope for. I'm only telling you one out of a possibilities here. I'm only telling you one out of b possibilities here. So, at most, I can uniquely determine integers between 0 and ab minus 1. But I'm claiming that every such integer is uniquely determined by its pair of residues, modulo a and b. So really, this is a bijection between all of these ab possibilities of the integers here and the possible pairs r and s of what this moduli are. So are there any questions about the statement? All right.

So let's prove this. The proof of this is clever. Usually, this is called the Chinese remainder theorem. And let's prove this fact. This will use a lot of the properties about Euclidean algorithm and modular arithmetic that we've been developing for today's lecture. And then, if we have time left over, I'll tell you a cool application, just to get some intuition for why this Chinese remainder theorem is so powerful. Actually, I guess I could have kept that. All right.

So let's prove this. So as I told you already, there are ab choices for n and, similarly, ab choices for the pairs s, t. So what we want to show is the following, that for-- I'm going to call this statement star-- I want to show that, for any choices of x and y that are between 0 and ab-- so any possible answers x, y-- well, what I want is that their pair of remainders are not both the same.

I take the remainder of x divided by a. I look at the remainder of x divided by b. That pair of remainders had better not both be equal to the pair of remainders I get when I take y and divide it by a and, similarly, I take y and divide it by b. So maybe I have pairs x, y that are different. I'll assume x not equal to y. Maybe I have pairs x, y that are different that have one of their remainders match, but I cannot get different x's and y's which have both of their remainders match. So that's the key property.

So let's show this. So we're going to show star by assuming it's not true and creating a contradiction. So this proof will be short, but the logic is pretty clever for this.

So let's assume that we have a pair x, y that are different, that are between 0 and ab, which have both of their pairs of remainders exactly identical. And let's produce a contradiction. So if we have such a pair x, y, then what I claim is that x minus y-- which way do I want to say this? oh, sorry, other way around-- a divides x minus y.

Why is that true? So we're assuming that this statement is false. Why is it true that the a divides x minus y? What does it mean for these two things to be equal?

Definitely, their first components have to be equal. So if remainder of x divided by a is equal to the remainder of y divided by a, then that means that they differ by an integer multiple of a. So a divides their difference. This is the same type of thing we did before. And, similarly, we know that b divides x minus y, as well.

And now, here's the key idea. We somehow have to use the fact that a and b are relatively prime. We know, by assumption, that the GCD of a and b is equal to 1. And what this tells us, it means that ab, the pair of them divides x minus y.

So this is the key thing. This isn't always true. If I had two things-- I had, I don't know, 6 and 9-- if I knew that 6 divided my integer and 9 divided my integer, it would not imply that 54 divides my integer because maybe I only have two powers of 3 sitting in there. But the key point is that, because both a and b are relatively prime, the only way that they both divide x minus y is if their product divides x minus y, as well.

So you can think about this in the case where a and b are not just relatively prime but they're prime, like in our example with 5 and 7. If 5 divides your integer and 7 divides your integer, there's no choice. 35 has to divide your integer, too. That's basically what the statement is saying. And this is the key step because, now, we're in good shape. And so if ab divides x minus y, I claim this is a contradiction.

So someone make me happy. Tell me what the statement contradicts. See, the statements I've made are not very complicated, but the logic is subtle. I wanted to prove that star was true, so I assumed it's not true. I assumed that there was a different pair x, y where they were both between 0 and ab where a divides their difference and b divides their difference. And I concluded that ab must divide their difference. But now I claim I'm done, and I've actually contradicted something.

AUDIENCE: If x and y are different, than they have to be off by at least ab.

ANKUR MOITRA: That's exactly right. That's right. If x and y are different and are between 0 and ab, they cannot differ by a multiple of ab. That's impossible. That's the same type of logic we used earlier. So this contradicts what we started off with. And what this means is that, hence, the Chinese remainder theorem is true, so very, very clever proof. Make sense?

And if you have any questions about these things, you should sit down and make sure that you can reconstruct the logic yourself. This proof and the proof of lemma 1 are not hard. But, really, one of the things with doing proof-based math is it's one thing to be able to verify, line by line, the steps as someone is giving them to you.

But to really understand it deeply, you have to check whether or not you can reconstruct these things yourself. These types of things often happen on tests, where you think that you understood something, but when you're forced to come up with it yourself, it's much harder. So that's just a piece of advice. Yeah?

AUDIENCE: Does the solution exist at all?

ANKUR MOITRA: Hmm?

AUDIENCE: The solution exists at all.

ANKUR MOITRA: So the solution exists at all. So the point is that every pair-- that's a good question. So we know that there are ab choices for n, and there are ab choices for s, t. Now, if every choice of n gets mapped to some choice of s, t, and what we just proved is that they don't collide ever.

So what this really means is that we can draw this bipartite graph right here where these are all of the values of n. These are all the different possible pairs s, t. And every n gets mapped to some value s, t just by taking the remainder modulo a and modulo b. And so what this proof just showed is that there are no x and y that map to the same thing. If there are no x and y that map to the same thing and they're the same size, there's no choice but everything maps to exactly one thing. And that means everything has a solution. And we can go back and forth. Good? Very good question.

There's also another way to answer your question, which is the way that I will do next, so thanks for the segue. That'll be our last math topic for today before I tell you an application. See, the same way-- it was one thing to ask this question about whether or not every integer has a multiplicative inverse modulo prime. And then we asked for more because we not only wanted that the answer was yes, we wanted to actually find what that multiplicative inverse is.

And the even stronger thing we could ask for is, OK, the Chinese remainder theorem is true. Hooray. But maybe we can also ask whether we can find the solution. So how can we actually find n? In our case, n was 29. So how can we find n?

And so here is the step-by-step procedure for how we would do it. So first thing is that we write n is equal to kb plus t. So this is just the statement that if I take n and divide it by b, the leftover will be t. And I don't know what k is. So this just captures one of my modular equations, the second one.

And we know that k is going to be between 0 and a, just because we know that our answer is supposed to be, at most, ab. And so what we can do is we can take this expression, which is just an integer expression. There's no modulus here.

And we can take mods on both sides. We know that this implies that kb plus t is congruent to s mod a. This was our first equation. And now this was our second equation in these modular constraints because this unknown integer n, we know it should be equal to t mod b. And same thing, it should be equal to s mod a.

And so what this implies now, if we just rearrange things, is that kb is equal to s minus t mod a. So just to emphasize, we know s, and we know t because those are the constraints in our system of modular equations. And what we're looking for is what k is. So now, does anyone have any ideas how we might solve this equation to figure out what k is? So what's the key idea? How can we solve this equation?

AUDIENCE: [INAUDIBLE]

ANKUR MOITRA: Perfect. That's right. We can just multiply by the multiplicative inverse of b, which, why does it exist? Because the GCD of b and a is 1 by assumption. So we know from the extended Euclidean algorithm that the multiplicative inverse exists, and we can find it.

So, in particular, we can solve this equation. And we get that k is congruent to b inverse s minus t. And this all happens mod a. We know b inverse exists. And because k is between 0 and a, well, k is actually equal to its remainder. So this just is the answer right here. So we can actually compute k using this assumption that k is between 0 and a.

So in the last 4 minutes, I'm going to tell you a really cool application. This is somehow like a baby version I made up of a more real application. But let me tell you about it.

So last time for my demo, I asked you guys, what are embarrassing movies you like. And I got much less exciting answers than I usually get, so you guys gotta work on that. But how many people here stream stuff online? There are people who didn't put their hands up? [CHUCKLES] How many people stream stuff online? How many people stream stuff illegally online? Oh, I saw you. OK.

So let me tell you about a cool application, which has to do with streaming, which is something related to fountair codes. Really, fountain codes are a whole topic in their own right, and this is a hugely simplified version, but imagine I've got some giant file n. And I have not just-- I'm representing the contents of the file as some gigantic integer. So this is a truly huge integer, not one I'd want to compute its factorization for by hand or anything.

But now, I have not just one person but many people streaming. Maybe it just came out, like a new season of *Stranger Things* or something, and everyone's trying to stream the same thing. And one of the ways that you can reconstruct the file is, imagine I'm just going to sit out there broadcasting a bunch of snippets of the message.

So what my snippets are going to look like is it's going to look like some modulus a1. And then I'll take the remainder of n divided by a1. And I'll do this for a whole different choice of these ai's. So I'll choose a whole bunch of primes ai. I'll tell you what the prime is. And then I'll tell you the modulus, what the file contents look like as a huge integer modulo a1. So how can you reconstruct the file?

So once you have-- you've received some set of the messages. These are what you've received. As long as the product of the ai's is larger than n-- that's really the generalization of the Chinese remainder theorem, where I just had two factors, a and b-- but as soon as I have the product of these ai's as larger than n, then the same type of thing works. And I can use the Chinese remainder theorem to reconstruct what n is.

But now the really cool thing about it, which explains fountain codes, is you have all these people streaming. And maybe it's a 1-gig file, but they're all not streaming at the same time. So what you can do is you can just broadcast these messages perpetually. It's like a water fountain that's always spraying water. And everyone comes there. And as soon as they fill up their 1 gig of data, they can reconstruct the file. It doesn't actually matter which 1 gig of data they get.

So these kinds of things are very effective ways of sending one thing to many parties. When those same parties are listening in and getting different snippets of the code, they'll still be able to recover it because they've all filled up their Nalgene bottle with the 1 gig. So this is a very simplified version of what fountain codes are because, really, you care about things like errors in the messages. But you can already see why things like the Chinese remainder theorem and modular arithmetic are extremely powerful.

So next time, on Thursday, we're going to be starting group theory. And we'll talk a little bit more about modular arithmetic, especially the parts we need for crypto. But see you all then. All right.