[SQUEAKING]

[RUSTLING]

[CLICKING]

Erik Demaine: All right, welcome back to 6.1200. Today we are talking about number theory. And number is, I guess, an old word here for integers, in modern terminology. This set black board bold z, which is all positive and negative integers. Although mostly we think about natural numbers, the non-negative integers.

Number theory is a big topic. We're going to spend three lectures on it, today and two next week. And it's a really important branch of-- subbranch, I guess, or related branch of discrete mathematics in particular, because it's found a lot of applications in modern days in cryptography and error correction, when you use your cell phone, probably several different things in that software is using number theory at some level to do crypto, to access your bank, to do error correction, to talk over radio waves, all sorts of fun things.

Number theory is also extremely old. It's probably the oldest branch of mathematics, though, past 4,000 years it's a little hard to tell exactly what people were doing. But in particular, the Babylonians, 3,800 years ago, were doing number theory. Pythagoras, 2,500 years ago, is doing number theory.

It had a lot of exciting activity in the middle of last century. Hardy is one of the famous number theorists of near our time. He wrote, "Number theory is great because it's one science whose very remoteness from human activities should keep it gentle and clean." He was a pacifist, and he wanted to do math that was completely useless from the perspective of warfare. Sadly, that's not the case for number theory anymore, but it used to be.

But the things we'll be doing here are pretty fundamental. Today, we're going to talk about divisibility and something called GCD, which you may have heard of before. It's a pretty basic notion. Both of these are. But we're going to define everything carefully and prove all sorts of cool properties about divisibility, and use it to solve at least one fun problem.

So the notation for divisibility is a divides b. We use a vertical bar. We use a vertical bar for a few other things like absolute value and set notation, so try not to be too confused. This will only be for a couple of weeks.

Divisibility means a evenly divides b, or b is evenly divisible by a. So for example, let's say, 2 divides 4 is an example. But in general, what we would like is a formal definition of this. So we'll say a divides b If there exists some integer multiple such that b is that multiple of a, so b equals k times a. Very important that this is an integer, otherwise, over reals this is always the case pretty much.

And we define divisibility in terms of multiplication, instead of division. You could divide both sides by a here. And so, for example, this is saying b divided by a is an integer. But that's not such a great definition because it doesn't work well when you try to divide by 0. This definition works even when a is zero, and if you do division, it doesn't. So that's why we define divisibility this way.

So more interesting examples are 2 divides n means what we have been saying, n is even. Whenever we say n is even, what we really mean is 2 divides n. There's some integer multiple of 2 that equals n.

Some other weirder cases, every natural number negative n divides n. It's maybe not so intuitive what divisibility means for negative numbers, but here are some examples. We allow this integer multiple k to be negative. So in particular, k can be minus 1, and minus 1 times negative n is n. Also n divides n, but that's a little less exciting.

Another fun fact is that every number divides 0. This is also maybe not the most intuitive, but it comes from the definition. k in particular, can be 0. And if you take 0 times n you get 0. So everything divides 0. A number also divides its negative, things like that.

A consequence of these two things together is that 0 is even. If you were ever worried about whether 0 was an exception to evenness in the alternation of these numbers. Yes, this is even. This is even. This is even. Because in particular, 2 divides 0. Everything divides 0, but in particular, 2 divides 0. That's the meaning of even, OK, great.

Let's state some theorems about divisibility. I'm going to call these divisibility facts. We're going to use them several times today. Let's say we have two numbers divisible by the same thing. We call this d is a common divisor of a and b. So if d divides a and d divides b, we call it a common divisor.

Then I claim, for example, d divides a plus b, their sum. Why? This is very easy to prove if you just work through the definition. So let's say a equals j times d, and b equals k times d. For some integers j and k, they might be the same. They might be different. Then a plus b, well, it's just the sum of these two things, right? It's j plus k times d, not very hard.

More generally, d divides sa plus tb for any integers s And t. This is just a simple extension of this property. I guess you can write this thing equals sj plus tk times d.

And it turns out this structure is interesting. It's what we call an integer linear combination. And this is a concept we will also be using throughout today. So much so that I'm going to give an acronym. I don't usually like acronyms, but this is so long, that it's going to save me a ton of work to write ILC for Integer Linear Combination.

If you know linear algebra, this is just linear combination, but where the coefficients need to be integers. So it's exactly this notion. So an integer linear combination of a and b is some number s times a plus t times b for any s and t that are integers.

So if I give you two numbers a and b, we'll only be doing integer linear combinations today of two numbers, then this is what they always look like. And given two numbers, there's some numbers you can construct via integer linear combinations and some you can't.

Actually fun fact, divisibility is like integer linear combinations of one number. This is saying b is an integer linear combination of just a. But that's so special we call it divisibility. Two numbers is where integer linear combinations really get interesting.

All right, so to motivate these concepts more, turns out these are key concepts to solving a famous puzzle, it's at least 80, 90 years old, called the water-pouring puzzle, but it became particularly famous through *Die Hard 3, Die Hard with a Vengeance.*

So we're going to watch a little clip. And if this clip is not included in this video, you can go stream or purchase *Die Hard with a Vengeance* and start at the one-hour mark, and you will see an exchange like the following.

So we have, of course, John McClane here, who's trying to save Manhattan along with his friend Zeus, who's an electrician who just happens to get brought into this whole chaos. Basically, the bad guys are trying to distract the good guys by forcing them to solve puzzles, so here we go.

All right, so we're going to focus on the problem at hand, which we can formalize as follows. So we have two jugs, we'll say, of capacity a and b, and we'll assume they're integers. And in the example that saves Manhattan, we have a equals 5, and B equals 3. And we'd like to represent some number, in this case 4, by a sequence of pouring operations.

So how could we represent this water-pouring problem using techniques that we have from previous lectures? Any ideas? I heard something. Yeah?

AUDIENCE: State machines.

Erik Demaine: State machines, good, so we'll use a state machine. And so we'll have a state x and y, means that there's currently x gallons in the first jug and y gallons in the second jug, or let's say the amounts in each jug/

So we start at the state 0,0, let's say. The jug started empty. We're given infinite water supply because they're at a fountain, conveniently. And let's see, we want to finish at a state that has 4 in one of the two values. And as John McClane said, it has to be in the bigger one, in this case, because 4 is bigger than 3.

So we'd like to-- goal is to get to 4 comma something. And the tricky part is to write down the transitions here. So let's write down some of the transitions, transitions. So given x and y, what are some of the things I could do? Any suggestions?

I have some partially filled jars. These are two jars of 5 and 3 pumpkins capacity, respectively. Maybe I'm at some position here. What could I do to these drugs? Yeah, in the back.

AUDIENCE: Fill one to the top.

Erik Demaine: Fill one to the top. I go to the fountain. I pour it up until it's completely filled, or this one, I could go to the filled point. Other suggestions? Yeah?

AUDIENCE: Empty it.

Erik Demaine: Empty, I could empty one of the jars. Well, hopefully I won't need that. So in the transitions, this corresponds to going to, I guess, a, y if I fill the first one. Or I could go to x, b if I fill the second one. I could empty by going to 0, 7 if I empty the first one. Or I could go to x, 0. So these are all-- sorry. This is all transitions from x, y. Anything else? Yeah?

AUDIENCE: You could put them-- you could switch one the jugs in one of them to the other one.

Erik Demaine: I could pour the contents of one jug into the other, cool, yeah. So here, for example, I filled up a 3. I poured in 3. I could fill this one up again and pour it in again. And at some point, I'll run out of capacity in this jug. And that's an event that I can detect when I reach level with this jug.

So to write that down more carefully, given x, y, we can transition to one of two cases. Either we get to completely empty the left jug into the right jug, so this would be x plus y. But that's only possible if it fits in the second jug if x plus y is less than or equal to b.

Otherwise-- I have to do arithmetic. This is too hard. --we move over b minus y or something. So we have x minus b minus y, b, otherwise. If we end up filling up the second one, then this is how much of the original x that I started with remains.

And you can do the-- there's one more rule, which is if I pour from right to left. Those are available state transitions in this puzzle. And now maybe I've talked about it long enough. You have some intuition on how to solve this.

Start empty, any suggestions on how to construct the number 3? Yeah.

AUDIENCE: So you fill up the 5-gallon jug.

Erik Demaine: Just fill up the 5-gallon jug, OK.

AUDIENCE: And transfer it to the 3-gallon jug.

Erik Demaine: OK, we're going to transfer the 3-gallon jug, until the 3-gallon jug is exactly empty, which if you're using water, it would be level.

AUDIENCE: Empty the 3-gallon jug.

Erik Demaine: OK, empty the 4-gallon jug.

AUDIENCE: And transfer the 2 into the 4-gallon jug.

Erik Demaine: OK, move the 2 over. This now feels like Towers of Hanoi, OK.

AUDIENCE: Fill the 5.

Erik Demaine: Fill the 5.

AUDIENCE: And then transfer.

Erik Demaine: And then transfer like this from 5 to-- did I get the right number? There we go. Water's particularly unsettled today. All right, so we've got 5 on the right if I can count. And we fill over here, until this one is filled. And now we've got the number 4, yay.

That's how we save Manhattan with this water-pouring problem but while the bad guys are off stealing money from the bank. But sometimes this puzzle isn't solvable, so let's take another example.

Let's say we want a equals 6, b equals 4, and we want the number 3. Is this possible? I already gave away that this is impossible. Why is it impossible? Any feelings? Hint, it has something to do with this stuff. Yeah.

AUDIENCE: There are no integer linear combinations. Is it possible that there were 6 and 4 [INAUDIBLE]

Erik Demaine: OK, you said there are no integer linear combinations where of 6 and 4 that make 3. That is true. But this problem isn't directly about integer linear combinations, is it? I don't know. Yeah?

AUDIENCE: There's no way to get an odd number out of this.

Erik Demaine: There's no way to get an odd number from these two gallons. Both of you are correct. The simpler reason, in this case, is it's impossible to get an odd amount.

I guess, the proof of both of these things is you look at all the possible transitions, and you observe that if you start with a and b being even, you can never construct an odd number by adding and subtracting. All we're doing here are taking numbers that we've constructed, adding them and/or subtracting them with a and b or each other.

And so if you start with even numbers, you add and subtract them, you always stay within the even numbers. But also, as you were saying, if you start with an integer linear combination of a and b, all you can get are integer linear combinations of a and b.

So here is a key lemma about this puzzle and why that is impossible, more generally, is all reachable amounts in this puzzle, are integer linear combinations of a and b. Let's prove this.

Any suggestions on how to prove a property like this? If you remember all the way back to state machines, hopefully you've been studying for the test. State machines are on the test. This lecture is not. That will be on quiz 2.

AUDIENCE: Invariance.

Erik Demaine: Invariance, yeah, in particular-- this is a statement about invariance. This is saying, for all reachable x, y, x and y are integer linear combinations of a and b.

So this is an invariant property. This is a state predicate. We want to prove that it's invariant, that it's true for all reachable states starting from 0, 0. And we usually do that using the invariant principle which requires two steps.

One is base case. We need to check that this predicate holds initially. And the initial state for us is 0, 0. And we observe that 0 is indeed, a linear combination of a and b, namely 0 times a plus 0 times b, great. Base case is true. And so now we just need the inductive step, or the I guess, the preserved, the fact that this predicate is preserved.

So suppose that we're at some state x and y, where the predicate is true, so x equals sa plus tb, and y equals s prime a plus t prime b. And then we just look at all of the available transitions. And we show that every number that you can make, is indeed an integer linear combination of a and b.

So let's do one example. We can construct x plus y. x plus y, we can rewrite as s plus s prime times a and t plus t prime times b. So that's if this transition happens to be possible, it may not be possible because there's this if condition. But if we show that every number we could possibly build has this property, then, of course, all the ones that we can actually reach also have this property.

So x plus y is indeed an integer linear combination of a and b, et cetera. OK, I don't want to prove it for every single transition here, just because there's a bunch of them. But for example, a is indeed an integer linear combination of a and b. It's 1 times a plus 0 times b. This one looks messier. It's actually just x plus y minus b. So it's almost the same. You just add a minus 1 here.

But the general property we're using here is, that all of the constructible numbers here are integer linear combinations of x, y a and b. OK, I lied. We're using integer linear combinations of four things. But all of the x and y's are also integer linear combinations of a and b.

So what we're really taking is integer linear combinations of integer linear combinations of a and b. And it turns out that if you take two integer linear combinations of a and b like, x and y, and you add them together, or in general take an integer linear combination of them, you still get an integer linear combination of a and b. As you can see from this proof. It's actually pretty much the same proof we were doing over here.

This was integer linear combinations of one number, d. And divisibility was preserved even when you took integer linear combinations, so same idea. And for any puzzle where you have that property, you can only construct numbers that are integer and linear combinations of a and b.

Now, for this puzzle, you can actually construct all of them that fit in the jars. So I'll just mention this as a claim. And if we have time at the end, we'll come back to it. We can obtain a value x if and only if, let's say, x is an integer linear combination of a and b and 0 is less than or equal to x, is less than or equal to max of a and b. So we might try to prove that later.

So now the central question is, how do we know whether one number is an integer linear combination of two other numbers? And when it is, can we actually construct the values s And t, so that sa plus tb equals x? Because that's what we needed to solve this puzzle was some way to-- we were only given the capacities of the jars and the target value. And now we want to represent-- let's check that this is actually possible for this example.

Sorry, not this example, the one that's actually solvable. So for this solvable puzzle, 4 equals 2 times 5, that's 10, minus 2 times 3. That's 6. 10 minus 6 is 4. So that's intuitively why this puzzle is solvable. Although that's not a proof. I mean, we just saw that it wasn't limited by this same issue.

So for example, if we started with 6 and 4, we know by this lemma, because 2 divides 4 and 2 divides 6, 2 will divide any integer linear combination, and therefore, any reachable value in this puzzle. And 2 does not divide 3. 3 is odd, not even. And so we will never be able to reach 3 because all the reachable states are even. That's a corollary now of this lemma, OK.

OK, but in the good case, and the bad case is easy to see, I guess, how do we know whether we're in a good case or a bad case? How do we know whether x is a integer linear combination of a and b, given x and a and b? That is the purpose of-- well, some motivation for the rest of this lecture. We can use it for all sorts of things.

All right, and I claim that this is going to be true if and only if this property is true. So now I'm going to restate this in terms of divisibility. This equivalence is not obvious, so we'll prove it by the end of today.

So what we really want to show is, that every common divisor of a and b-- remember this is our definition of d is a common divisor, a common divisor of a and b, divides a and it divides b --then it should also divide x. So in the example, we saw that 2 was a common divisor of 4 and 6, and so 2 better divide anything that we make.

What are the common divisors of 5 and 3? 3 doesn't work. 5 doesn't work. 2 doesn't work. 3 doesn't work. I guess 1 is the common divisor. And 1 divides everything. So yeah, this is true in our good case.

And this, we can rewrite in yet another way, which motivates our next topic. The GCD of a and b divides x. So I claim, in particular, this is going to be a way to tell whether x is an integer linear combination of a and b. You compute something called the GCD of a and b, and you see whether it divides x. So let's talk about GCD.

GCD is another acronym. It stands for Greatest Common Divisor. I assume it comes from Britain, in particular, where they use "great" to indicate large. So we might call this the largest common divisor or the biggest common divisor, all synonyms for the same thing.

I'm going to give two equivalent definitions of GCD. To me, the more intuitive one is this. But it's a little bit uglier because it has a special case. So what I'd like to say, is the GCD of a and b, is the largest integer d such that d divides a and d divides b And again, this is common divisor. d is a common divisor of a and b. And I want it to be the largest common divisor. That would be GCD.

But there's one annoying special case, which really won't matter for the rest of the lecture, but to be technically correct, we should say, what happens when you compute the GCD of 0 and 0? Because remember, every integer d divides 0. Everything divides 0.

So what's the largest divisor of 0 and 0? I mean, infinity? I don't know. It's like there is no such integer that's the largest. So there is no largest integer. So in this case, it turns out the right thing is to define GCD of 0, 0 to be 0. That will be a little counterintuitive maybe, until you see the next definition.

So here's another definition. GCD is the only non-negative common divisor of a and b divisible by all common divisors of a and b. So another way to think about the greatest common divisor, is it's greatest in terms of the divisibility relation, the one that is the most divisible by other common divisors. So it turns out there's going to be one common divisor that's both non-negative and is divisible by all common divisors of a and b.

So in these examples, for 4 and 6, GCD of 6 and 4, is going to be 2. Because the only-- let's see. The common divisors, what are all the common divisors between 4 and 6? It may be obvious that there's two. There's minus 2, there's 1 and minus 1.

0 divides nothing except itself. And you can check all the numbers between negative 6 and 6. Those are the possible divisors. And these are the only ones that divide both a and b. And the largest non-negative 1 here is 2. And also it has the property that all of these other common divisors divide 2.

So whichever way you want to think about it, you can use either definition. I'm not going to prove that these definitions are equivalent. We're just going to take it as an axiom. But you can prove it. This may be an exercise. But when we're proving things about GCD, we can use either of these definitions, whichever is more convenient.

So let's do one more interesting example, which is GCD of, I think, I want to do 0 first, 0, a. So suppose I want to compute GCD of two numbers. Maybe I should put a b there. And one of them is 0. Any guesses what the GCD of 0 and b should be? Remember, everything divides 0.

AUDIENCE: b.

Erik Demaine: b, it's a good guess, almost right. What if b was negative 7? This should be absolute value of b. So if we want to handle positive and negative things, this turns out to be the right answer. Here is a mini proof.

There are two things we want to check. One is that absolute value of b is a common divisor of 0 and b. So well, the absolute value of b-- this gets notationally pretty gross --divides 0. That's true because everything divides 0. And absolute value of b divides b, namely, when we let k be negative 1 or 1, depending on whether b was positive or negative or 0. Cool, so good it is a common divisor.

And now I want to claim that it's the greatest common divisor according to the second definition. So let's suppose that we had some number d which divides 0 and which divides b. So d divides 0, and d divides b. Then I claim, in particular, d also divides the absolute value of b. Because if you divide a number, you also divide its negation, just by negating the value k that you're multiplying by. If I take k times d, and I get b, then I take negative k times d, And I get absolute value of b if b happened to be negative.

OK, so that means if you're a common divisor of 0 and b, then you're also a common divisor of this thing, and that was our second definition of GCD. So great, this means your greatest.

OK, now the question arises, how do we compute GCD of a and b? Because I claimed, over here, that what we need to check is whether GCD of ab divides x. So given a and b, we need to be able to compute this thing. How do we find the greatest common divisor?

It turns out there's a great algorithm to do it. And the base case of that algorithm is this. If we can get one of the numbers down to 0, then we found the GCD. So our goal now is given some big numbers, to make them smaller somehow. And the big idea here to make a GCD algorithm, is subtraction. How do we make numbers smaller with arithmetic? Subtraction or division seem like good ideas. We're going to use both, starting with subtraction.

So I claim GCD of ab equals GCD of a minus b. b for all integers ab. So if a and b are big numbers, this is pretty cool. This is saying we can subtract one of the numbers from the other, and we get the same GCD somehow.

We'll prove this in a second, well, now, actually. I'll leave a little bit of space. Let's do a proof. So then we're going to turn this into an algorithm. But let's first see why this is the case.

So this is a statement about GCD. GCD is this complicated definition. It's a little easier to think about all common divisors. If you think about the common divisors of a and b, those are all the numbers d, where d divides a and d divides b, and you think of the common divisors of a minus b and b, I claim these sets are the same. So the set of common divisors of a and b, equals the set of common divisors of a minus b and b.

OK, this is a kind of proof we actually haven't seen in this class yet. This is the brace, not a minus sign. When you want to prove two sets are equal, a very common template for such a proof is to show that every element of this set is also an element of this set that, if you recall back to our set definitions in lecture 1, is that this set is contained in this set and also to prove that every element of the second set is an element of the first set. That's saying the second set is a subset of the first set.

So we're going to do a proof in two parts. Part one is to show containment, that the first set is contained in the second set. Part two, over here, is to show containment in the other direction. This set is a superset of this set.

If I can prove that these sets are equal, then in particular, the maximum element of these sets, if it exists, is also going to be equal. If you have two identical sets, you take the max, you're going to get the same thing.

Now, I'm using the first definition here. Of course, there's this one case. when the two numbers are 0, then there is no maximum. Because all integers actually, will be common divisors of 0 and 0. But in that case, again, if the sets are equal, that means that a and b-- or these numbers must be 0. And then of course, these numbers are also all 0, And this statement is trivially true when everything is 0.

It's just saying GCD of 0, 0 equals GCD of 0, 0. That one I'm fine with. But as long as we're not in the 0, 0 case, these sets will be finite. And if they're equal, then their maximums will be the same. So this is all I need to show. Let's do it in two parts.

So first, we suppose that we have a common divisor of a and b. So that means d divides a and d divides b for some number b. Well, I claim that also d divides a minus b because a minus b is an integer linear combination of a and b. And we had, over here, in our divisibility facts, if d divides a and d divides b, then d divides every integer linear combination of a and b. So we're plugging in S equals 1, t equals minus 1 here, and we get that d divides a minus b, great

So we have d divides a, and d divides a minus b. So in other words, we have divides a and d divides a minus b and that-- sorry, wrong one. I wanted d divides b. And this is the meaning of a common divisor of a minus b and b.

So we assumed that d was a common divisor of this, and we proved that d was a common divisor of these two numbers. Therefore, we have containment. Every element in this set, is contained in this set. Now let's prove the reverse, and that will imply that these sets are equal.

So now suppose that d divides a minus b, and d divides b. Then I want to show that d divides a. Why? Any suggestions? Can I use integer linear combinations? Yeah.

AUDIENCE: [INAUDIBLE] Like t equals 0 in front of it.

Erik Demaine: T equals 0. I agree, if I put t equals 0 and s equals 1, then this is an integer linear combination of a and b. But in this case, I don't know that d divides a. That's actually what I want to prove.

So I can't use integer linear combinations of a and b. But--

AUDIENCE: Yeah, you can't do it with a minus b.

Erik Demaine: I can take an integer linear combination of a minus b and b. If I do 1 times this plus 1 times this, a minus b times 1 plus 1 times b is a. So in other words, because this equals the sum a minus b plus b, and because I know that d is a common divisor of a minus b and b, I know that d also divides any integer linear combination of those two numbers, and in particular, a, cool.

So both of these are using the divisibility facts that if you have a common divisor, you also divide all integer linear combinations of those two numbers. So in particular, this means we have d divides a and d divides b. I'm just taking this thing that we proved and this thing that we assumed.

And this says we have a common-- it's saying if we had a common divisor of a minus b and b, we also have a common divisor of a and b. So that says every element in this second set is an element of the first set. And because we have containment in both directions, in fact, these two sets must be exactly the same set. So if you ever have to prove that two sets are equal, I recommend a common template is to do it in two halves like this, OK, great.

So I claim we can get an algorithm out of this, out of this subtraction idea and GCD of 0, b. So let's just do an example of that. I won't write a careful algorithm because we're going to do better.

Let's say we want to do GCD of 9, 6. OK, oh, and maybe I want to write another fact here. Does it matter? Yeah, I do. So GCD of a, b is also equal to GCD of b, a by symmetry of this definition, a and b can be swapped, and you get exactly the same things we're talking about. And if I apply this subtraction lemma with the elements swapped, what I get is GCD of the first argument, which is b, minus the second argument, which is a, comma the second argument, which is a.

So all these things are equal, just from the same subtraction lemma and using the symmetry. But what this is saying is either I can subtract the second argument from the first and get a minus b, or I can also rewrite this as GCD of a, b minus a. Alternatively, I could leave the first argument alone and subtract it from the second argument.

So I'm free to subtract whichever way I want. And I find it much easier to think about non-negative numbers, so I'm always going to subtract the smaller number from the bigger number. I can do it either way.

So let's do it for 9, 6. So I'm going to subtract 6 from 9. And I get. This is equal to-- so this would be a minus b. This one minus this one is 3. 9 minus 6 is 3. The second argument stays the same.

OK, I still don't know the answer, so I'm going to keep subtracting. In this case, I'll take the smaller number, subtract it from the larger number. So this is b minus a, I guess, replacing b with b minus a, I get GCD of-- sorry, I wanted. Yeah, 3, 3. Subtract 3 from 6, I get 3.

OK, that looks pretty easy. I'm guessing GCD of 3 and 3 is 3. But if we go all the way, we can subtract one of them from the other. I guess, I want to do b minus a again, and we get 3, 0. And this we actually analyzed. Over here, we said GCD-- oh, I got it backwards. --but of 0 comma something, is the absolute value of something. So absolute value of 3 is 3. And so GCD of 3, 0 is 3, OK, great.

Another example, GCD of 1, 100. OK, well, I'll take the smaller one, subtract it from the larger one. I get GCD of 1, 99. OK, now, I subtract 1 from the larger number, I get GCD of 1, 98, and so on. I won't bore you with the details. In the end, the answer is 1. But this is a very slow way to arrive at the answer of 1. I have to do 100 steps to figure out the GCD of 1, 100.

So the number of steps you get in this algorithm, which I haven't precisely defined, is going to be roughly, at least in the worst case, the sum of a and b, or the maximum of a and b, it doesn't matter, up to constant factors. And this is what we would call a slow algorithm in number theory.

Because generally, we want to think about numbers that are huge, 100 bits long. If you have 100-bit long number, that is as large as 2 to the 100, and that's a very large number. It's big, roughly the number of particles in the known universe. I think that's about 2 to the 128 or something.

So we don't want to spend that much time to compute this. In fact, we can do a lot better using a better technique, which is division. So we've seen here, the idea that we can subtract one number from the other, but if we subtract, and then we subtract, and then we subtract, and we subtract, that sounds a lot like division, right? This is like the bad way to do division is to repeatedly subtract one number from the other.

And that's exactly what happens in the 100, 1 case. We have to subtract 1 from 100, subtract it from 99, keep subtracting 1, until the left-hand side becomes larger than the right-hand side. And then we could stop. In fact, they become equal because 1 divides 100.

But in general, we want to keep subtracting until we go beyond, or we would go beyond what we can afford. And that's exactly division. So let me give you a very precise version of division with something called the division theorem. This is one of the core theorems in number theory. We won't prove it here. It's kind of tedious.

For every integer n, and for every positive integer d-- so I'm going to write z plus to mean d is greater than 0 -- there is a unique-- and usually we would write this as exists exclamation mark. There is a unique. So this means there is, exclamation mark. It's really exists. But there's only one of them. --there exists a unique pair of integers q and r-- I don't want the parentheses, actually. I want to say those are both integers --such that n equals q times d plus r. and r is between 0 and strictly less than d.

OK, you actually have seen this in other terms probably before. This q is what we would call the quotient. And this r is what we call the remainder. I'm sure you've heard those terms before. This is like a very precise way to say what division gives us in all cases.

Given a numerator and a denominator, that's why they're called n and d here, and as long as d is not 0, we're going to Furthermore assume that it's positive, so it's easier to think about, then there's a unique quotient and remainder you get from dividing a over b. A q is how many whole copies of d you can fit inside n, and r is whatever's left over. So that's roughly a proof that this is true.

And if you think about what the remainder should be, it could be 0. And the 0 case, that's when we have divisibility. If d divides n, if n equals q times d, that's exactly divisibility if there's no r. That's when r equals 0. That's what we get.

But in general, we know that r should also be less than d. Because if there was d left over, we would just increase q by 1. That would be a whole copy of d. So the remainder is always going to be between 0 and strictly less than d.

I guess the interesting part here, is that there's only one-- sorry, I should have put an i in unique. There's only one way to do this. That with these conditions, there's only one quotient and remainder,

OK, I also want to give you some definitions, some notation, which is for the quotient, we write the numerator div d div for division. And for the remainder, we write n rem d, rem for remainder. This is also sometimes called n mod d. So if you ever see that, we probably won't use it in this form. We're going to use mod to mean a slightly related, slightly different notion. Also, in the textbook, they write rem as a two-argument function but same idea.

So we're going to use division theorem a lot from now on and as a way to reduce our numbers again, in a more powerful way than subtraction. So we would like to do subtraction repeatedly. And what will we be left with? We'll be left with the remainder. If you keep subtracting until you can't subtract anymore, that's remainder. So let's do that.

I probably still need these divisibility facts, so I'll leave it there. So this is GCD division theorem. Let's suppose a is less than or equal to b is less than or equal to 1. Is that what I want? No. Other way around, that seems weird. It'd be weird to be at most 1. I want things to be non-negative, so I'm going to assume a is the bigger one of a and b. And I want to compute GCD of a, b.

And I'm going to assume we haven't gotten down to 0, in particular, because I can't divide by 0, but also because when we get to 0, we're done. If one of the numbers is 0, we're happy. So let's assume they're both at least 1, and let's assume, by symmetry, that the one on the left is bigger or equal, then I claim GCD of a, b, is GCD of a remainder b, b.

So before, we had a minus b, and now we have a remainder b. Seems like a small difference. And indeed, the proof is basically the same. I'm going to do a proof sketch, which is to say, well, GCD of a, b is equal to GCD of a minus b, b. And if we apply by the subtraction theorem, which we just did over there, and if we apply the subtraction theorem again, we can subtract the right argument from the left one, again, a minus 2b, b and so on, eventually, we get to GCD of a minus a div b times b.

a div b is how many whole copies of b I can fit into a. And so if I take that number, multiply it by b, that's how much I can subtract away before a would go negative if I subtracted one more time. And what I'll be left with, is exactly the remainder, a minus a rem b times-- oh, no times, comma b. If I subtract off that many copies of b, I will be left with a remainder b, sorry, no, a minus.

That's what we're writing here, in fact. Let me rewrite this. We have n, or maybe I'll write it as a or b. Let me stick with that. What this over here is saying is, n div d times d plus n rem d equals to n. I'm just rewriting this formula using the notations for q and r.

And if I rearrange this, I get that n minus n div d times d equals n rem d. And that's exactly what I'm using here. The remainder is what you get, what's left over. If you take the numerator and subtract off the div many copies of the denominator.

So I use dot, dot, dot here. You can't really use dot, dot, dot in a proof. Technically, this is a proof by induction. But I think, hopefully, you're familiar enough with induction that you could turn this into a formal proof. It's not hard, OK.

So we can divide. And this will make this GCD algorithm much faster. Because now if I want to say, compute GCD of 1, 100, this is the same thing as GCD of 100, 1. And what that says is, I can divide 100 by 1 and see what the remainder is. Well, 1 divides 100, so the remainder is 0. So I instantly get that this is GCD of 0, 1.

Boom, I'm done. This GCD of 0 comma anything is the anything. So in one step now, one division, instead of having to do a bunch of subtractions, I compute the GCD. Great, let's write this down as an algorithm.

This is called Euclid's algorithm, or the Euclidean algorithm. It's called Euclid's algorithm because Euclid was not the first to discover it, of course, but he's still pretty early, around 300 BCE. But I'll mention an earlier version in a moment.

So what this does is it computes GCD of a, b for a greater than or equal to b greater than or equal to 0. OK I'm going to assume everything's non-negative. If it's negative, just negate it. I'm going to assume the left argument is bigger than the right one or equal. If it's the other way around, swap a and b, so this is enough for computing GCD of any two integers.

And I'm going to write this, of course, as a state machine. So in general, I'm going to have a pair of integers, a and b, where a is greater than or equal to b is greater than or equal to 0. So I guess I should say the states are x, y where x is greater or equal to y, is greater or equal to 0.

And then the transitions, go from x, y to x rem y-- oh, wrong order. --y, x rem y. OK, almost the same as this theorem, except I'm going to rewrite this as GCD of b, a rem b, just flip the arguments because I would like the left argument to always be greater than or equal to-- I would like the left argument to be always greater than or equal to the right argument.

And if I do them in this order, if x was greater or equal to y, I do this division, this division will, of course, be less than y, and so the left one will be bigger than the right one, actually strictly in that case. So that's basically just following this rule repeatedly, and those are my transitions.

Now, this only works when y is bigger than 0. If y is 0 at x, 0, we return x as the GCD. That's the final state. Or these are various final states, any x, 0 is going to be a final state. There's no transition out of there because we're not allowed to divide by 0. Because I wrote this for all y greater than 0 rule. But in that situation, we know the GCD of 0, b or b, 0 is the absolute value of b because everything here is non-negative. The answer is x at that point.

So why does this work? Well, because we just proved an invariant that as this machine runs-- I want to say p of x, y is the state predicate, which is the GCD of x, y equals the GCD of a, b.

We designed the change in arguments here. We started with a, b, so this is true initially. In our initial state, a, b, of course, GCD of a, b equals GCD of a, b. And then every time we do a transition and we do this division and take the remainder, we know that the GCDs are equal by this theorem. And that's exactly what this tells us.

And so this is saying, this theorem tells us that this state predicate is preserved. And because it holds initially, that means it's invariant, OK, true initially, and it's preserved by this theorem. And so by the invariant principle, we know that this will remain true for any reachable state x, y, we'll have GCD of x, y equal GCD of a, b.

And furthermore, when we get to a final state-- so this is the final state. --we know that this answer is correct because GCD of x, 0 equals x for non-negative x. So this is partial correctness. If we get to a final state, GCD of x, 0 equals x for x greater than or equal to 0 by this theorem that we proved.

So if this machine terminates, we're done. And I claim this machine does terminate. So let's prove termination in a couple of ways. So how do we prove termination of state machines? We define some value, some function of states, that's an integer, a non-negative integer, and we show that it strictly decreases. That's our main tool for proving termination of algorithms.

So for example, we could take the function x plus y. This strictly decreases. Why? Just think about a transition. We start with x, y. y doesn't change. And then the other term in the sum used to be x, and then it becomes x remainder y. So we're dividing x by y, so x gets smaller in that process. And so the sum of these two values, is going to be smaller than the sum of x and y.

So that's cool. We now know that this algorithm terminates. But stating it using this function, is not so great because it tells us that the number of steps is at most, a plus b. But that's not great. That's what we had before. From our slow algorithm, we got a running time of a plus b.

So that's still true. It's an upper bound on the running time. But I claim that, in fact, our running time is much smaller by a more clever proof. Let me erase subtraction.

So I claim, in fact, If we look at the bit count of x and the bit count of y in summation, this also strictly decreases. What is the bit count? This is if you write x in binary, how many bits are there? What's the length of the binary representation of x?

This is essentially 1 plus log base 2 of x ceiling. A little gross, but asymptotically, this is log base 2 of x. Or when you use asymptotic notation, you don't even need the base because it only makes a constant factor difference. So this is roughly log of x is theta log of x. This is theta log of y.

And so the number of steps is order log x plus log y. And that's really good. That's exponentially faster. Sorry, I should have written log a plus log b because that's where we start is a, b. So if I give you two numbers a and b, I can compute the GCD in roughly logarithmically many divisions. Whereas before, we did a plus b, now we have log a plus log b. So that's what we call an exponential improvement. And this is what we would call a fast algorithm.

Because now if I have 100-bit integer, it only takes me-- or two 100-bit integers it only takes me around 100 steps. Whereas before, it was taking me 2 to the 100 steps, big difference. OK, why is this true? Why is bit count of x plus bit count of y strictly decreasing?

Because I claim-- let's just say because x rem y is less than x over 2. This is another fact about division. I think the intuition here is y is smaller than x when we're doing this division. And so we can remove at least one whole copy of y from x.

And maybe we remove more, but yeah, you can think about if y is more than 1/2 x, then just in the first step, we've removed 1/2 of x. And so we've gotten down to less than x over 2. If x is smaller, if y is really small, like 1, then we're going to be able to eat all of x. In both cases, you'll end up eating at least 1/2 of x And so in general, the remainder will be less than 1/2 of x over 2.

And so if we start with x, and in the next step we get to x over 2 or smaller, that means we removed one bit from the binary representation. Or if you think about it in log base 2, if we halve the number, log base 2 goes down by 1. So even with the ceiling, it goes down by 1. So that's why this Euclid algorithm is fast and good. That's why we love it.

I have for you a little animation of the algorithm, and then we'll talk about extensions. So here, on the screen, we're computing GCD of 189 and 606. If we think of that as a rectangle, 189 by 606, then doing a division is like square packing.

So here, if we have an a by b rectangle, and b is the smaller one, then a b by b square is like subtracting off b. And so if you think of the whole rectangle, it's how many squares can I subtract off? And what we'll be left with is the remainder.

Because one dimension stays the same, the other dimension gets reduced to the remainder. So hopefully that's clear why the geometric view of squares is the same as what we've been doing with Euclid's algorithm. Makes for a much nicer picture.

And then you take that rectangle, and you flip it so that the horizontal dimension is bigger. You take off as many squares as you can, and you flip, and you take off as many squares as you can. Here's just one. That's just a subtraction. You flip. You take off as many squares as you can. You're left with a remainder of 3 by 6. And now here we get even division, and so one of the dimensions becomes 0. And then the GCD of 3 and 0 is 3, cool.

So if that helps, great. If not, don't worry. Hopefully, you can now compute GCD. We will do another example in a second. I would like to first strengthen this algorithm and make it do more.

Because remember, we wanted to compute GCD. Why? We wanted to see whether GCD of a, b divided x. Because if the GCD of a, b divides x, then all common divisors of a and b will divide x.

And the reason we cared about that is I claimed it was the same thing as x being an integer linear combination of a and b. But where did the integer linear combinations go? All we've done so far is compute GCD of a, b.

Well, it turns out there's a connection. And the connection is called Bézout's theorem or lemma. It's called Bézout's lemma because Bézout's is like the last person to discover it in 1779, after another French mathematician, Méziriac in 1624, and after two Indian mathematicians around 500 ACE. But we call it Bézout's lemma.

This is the connection. GCD of a, b is an integer linear combination of a and b. And we can prove this using Euclid's algorithm, which is pretty cool. We're going to, by Euclid's algorithm.

OK, let me do this quickly. Let's assume a is greater or equal to b, again. If not, we can just swap a and b. And I guess, we'll assume greater than or equal to 0. So we can apply Euclid's algorithm.

Euclid's algorithm tells us, or the theorem we proved before Euclid's algorithm, tells us GCD of a, b equals GCD of b, a rem b. And we can use induction now. By strong induction on a plus b.

So this is very similar to what we were saying here. x plus y strictly decreases. That's all we need for induction to work here. So we're assuming by induction, that if a plus b is smaller than the current value, then GCD of a, b is an integer linear combination of a and b.

And so here, we have GCD of a and b, and we've reduced it to GCD of some smaller values, or whose sum is smaller. And so we can apply strong induction to this pair. And so we'll get that this GCD is an integer linear combination of its arguments. So let's continue here.

So by strong induction, we get that GCD of b, a rem b, is an integer linear combination of b and a rem b. And this value, of course, is equal to GCD of a, b, by this equality. And so we get that the GCD we care about, is an integer linear combination, not of a and b but of this reduced version, b and a rem b.

That's kind of annoying, but it turns out a rem b is an integer linear combination of a and b. Why? By the division theorem. I actually wrote it out over here. We wrote that n rem d is n minus something times d. That is, this thing is an integer linear combination of n And d. And so if I'm doing a rem b, I get that it's a minus something, the div, times b. So that's an integer linear combination of a and b.

So what we're left with is-- we have that this GCD that we care about is an integer linear combination of b and something else that is an integer linear combination of a and b. So in other words, we have an integer linear combination of two integer linear combinations of a and b.

Because b is definitely an integer linear combination of a and b. It's 0 times a plus 1 times b. a rem b is an integer linear combination, as written over there. And so now we're just taking an integer linear combination of those two things. And that's also an integer linear combination of a and b, as we showed before.

So that's maybe a little abstract, but it does prove the theorem. Now I'm going to turn that into an algorithm, cool algorithm, maybe over here, which is called the Pulverizer.

I mentioned Euclid's algorithm was discovered before Euclid. In fact, a stronger version of Euclid's algorithm called the Pulverizer, was described by Aryabhata, an Indian mathematician, around 500 ACE, so a long time ago. And he called it Kuttaka, which is Sanskrit for pulverization. So we're going to call this the Pulverizer. Most people call this the extended Euclidean algorithm.

And it's basically, run Euclid's algorithm but also keep in mind, what's going on in Bézout's lemma and always keep track of x and y as an integer and linear combination of a and b. OK, so that's the intuition. As a state machine, it's quite simple, kind of simple, a,1, 0 b, 0, 1 is where we start because a equals 1 times a plus 0 times b, and b equals 0 times a plus 1 times b. So that's how we represent a and b as integer linear combinations of a and b, pretty trivial.

And then our transitions, if we have x, s, t so this means x equals s times a plus t times b. And we have y, u, v. So in other words, we've represented y as u times s plus v times-- sorry, v times a plus v times b. Then we transition to-- oh, that's a mistake. This should be y, u, v.

As usual, we just copy y to the left argument. We don't change it. And then x changes to x rem y. And then it turns out you can write, you can figure out how x rem y is relative to-- as a linear combination of a and b like this. It's kind of a little gross.

So we started with representing x as s times a plus t times b. And we take s, and we subtract off a corrective term, which is x div y times either u or v, where u or v is what represented y. If you work it out, that's exactly what we were doing here.

We started with one number, and we subtracted off the div times what we had before. So if you stare at this equation long enough, this is exactly what we're doing but in integer linear combination space of a and b.

So I'm not going to justify this anymore. And I won't even show an example. But it gives you-- this is a particular way to compute it. And I will say at the beginning of an example, if you have 5, 1, 0 3, 0, 1, and you run this a few times, you get to 2, 1 minus 1, 1 minus 1, 2. And this part tells us that 1 equals minus 1 times 5 plus 2 times 3. I hope that is correct. Minus 5 plus 6 is 1, yep.

OK, So this is how you represent 1 as an integer linear combination of 5 and 3. 1 is interesting because it's the GCD of 5 and 3. This is connecting to the water pouring. Water pouring, we had a container of size 5, container of size 3. Their GCD is 1. They have basically no common divisors except 1 and minus 1.

And we can represent the GCD as an integer linear combination of 5 and 3, namely this one. Therefore, we can represent 4 as an integer linear combination of 5 and 3. We just multiply both sides by 4, and we get minus 4 times 5 plus 8 times 3, is one way to do it.

And that gives you at least a way to solve this problem. It turns out you can take eight copies of 3 and subtract off four copies of 5. How do I do that? Well, I just fill this up 8 times in total. And each time I pour all that I can into the thing of size 5. And each time it overflows, I empty out the 5.

And so that's exactly how I can build any integer linear combination of 5 and 3 as long as one side is positive and the other is negative, which turns out to be necessary, then you can just take that many copies of filling up one bucket, pouring it into the other, removing whenever it overflows, and you will get exactly the number you wanted.

For these numbers, pours of 3 into 5, will give you one. If you multiply this by 4, I get the number 4, and I save Manhattan, great. Good luck on your quiz on Thursday.