MITOCW | 11. Integer Arithmetic, Karatsuba Multiplication

PROFESSOR: I hope you're all recovered from the quiz. Our apologies for hard questions, but it's just the first quiz. We have a lot more fun things to do in 006. So today's lecture today's lecture is a light lecture, I would even call it a recreational mathematics lecture. And so, thanks for coming. I'll make it worth your while, all right?

So we're going to talk about irrationals. The motivation behind this is really that every once in awhile we are going to have a situation where we want to compute with numbers that are much longer than 64 bits, which is really the word length in standard computer these days. It used to be 8 bits back in the day. For most of my adult life it was 32 bits. And recently Intel and AMD have gone to 64-bit processors. But 64 bits ain't near enough if you want to do what's called high precision computation, you want to find precisely the weight of a neutrino, if you're a physicist, for example. And that, you're talking about literally a hundred decimal digits, which is obviously much more than 64 bits. And that's just one example.

So what happens if you want to compute the square root of 2 to a million digits, or pi to 10 million digits? How do you do that on a computer? So that's what we're going to do for this module, which is a short module on numerics.

We'll have a lecture today and another one on Tuesday telling you about techniques that use, among other things, Newton's method to compute irrational numbers, as one example, to arbitrary precision And for your problem set you are you going to look at different application that corresponds to encryption, RSA encryption, where you have to work with primes. These are now integers, but you work with primes that are thousands of bits long, so again, much more than 64 bits. And so you have to think about how you're going to multiply numbers that are thousands of bits long, how you're going to divide numbers that are thousands of bits long, and that's really the purpose of this module.

So let's start off by talking about irrationals. And Pythagoras, whom I'm sure you've all heard of is credited with the discovery that a square's diagonal and its side are incommensurable. So you can't really express the ratio as a rational number, as a ratio of integers. Now it turns out that the Babylonians and the Indians knew this way before Pythagoras, but he gets credit for the Pythagoras theorem.

And there's also a Greek philosopher, in fact, maybe he was first a philosopher and then a mathematician, and he the espoused a philosophy that, I guess, is called Pythagorean mysticism, that said that all is number, so the world is about numbers. And he worshipped numbers, his followers worshipped numbers. And the problem here was that he didn't really like the square root of 2 because he couldn't express it as a number, of what he thought of as a number, which was 1, 2, 3, et cetera, integers, whole numbers.

So he called this ratio speechless, something that he really couldn't explain. And irrationals were actually considered a threat to the mystics because they couldn't really explain what square root of 2 was. They'd try and measure it, and they would come up with the right answer because the next time around it would be a little bit different If they did things a little more precisely, or not so precisely. And it bothered them no end.

And so they tried to find patterns in irrationals because they considered them a threat. And they obviously didn't find patterns, but imagine if we could actually find patterns. I mean, that would be a really big deal, it would be better than p equals np, if you know what I mean. If you don't it doesn't matter.

So that's another of motivation for high precision arithmetic. Let's try and find patterns in irrationals if you go to millions and trillions of digits, maybe it's just a matter of time before we discover that there's no such thing as irrational numbers. Who knows? So let's do that for the rest of this lecture. Let's try and figure out how we're going to compute irrational numbers, or things like square root of 2, to arbitrary precision. So we can go play around, and we'll give you some code, and you can play with it.

So if you look at square root of 2-- I'll just put this up here-- so it's 1.414, you probably all know that. Then it's 213,562,373,095,048 et cetera. I mean, I don't see a pattern there. I see a zero, a couple of zeroes here. It's hard to imagine-- you'd probably want to think of a computer program that generates square root of 2, and then maybe a different computer program that's looking for patterns.

So let's not worry about the square root of 2. I want to digress a little bit. I did say this was a bit of a recreational mathematics lecture. And let's talk about something completely different, which are Catalan numbers. So these are really my favorite numbers in the world. And people like primes, some people like irrationals, I like Catalan numbers.

Catalan numbers are-- they show up all over the place. How many of you know what Catalan numbers are? Oh good. Excellent. So Catalan numbers have a recursive definition. You can think of them as representing the cardinality of the set p of balanced parentheses strings.

And we're going to recursively define these strings as follows. We're going to have lambda belonging to p where lambda is the empty string. And that's rule one. Rule two is if alpha and beta belong to p then I'm going to put a paren, open paren, alpha, close paren, and then beta. And that belongs to p.

So you iteratively or recursively apply this rule over and over, and you start getting strings that are balanced. So this is balanced. Now that's not balanced, this is not balanced, and so on and so forth. Obviously things get more complicated in terms of the parentheses if when you have more brackets or parentheses.

And so, the nice thing about this definition is that you can get every non-empty balanced paren string, via rule two, from a unique alpha, beta pair. So as an example, suppose you want to generate the string that looks like this-- So that's a little more complicated than the strings that we have looked at so far-- then you obtain that by having alpha be this simple string. And then you put the brackets around that, and then your beta corresponds to this.

So now alpha and beta were previously generated, so if you applied rule two to the empty string, with alpha being the empty string and beta being the empty string, then you get this thing here. And obviously you could get beta by setting alpha to be the empty string and beta to be this string that you just generated, and so on and so forth. So you just keep going and the strings get longer and longer. The cardinality of the set gets bigger and bigger. And those of the Catalan numbers. OK

And so, this is a non-trivial question, which is I'd like to enumerate the Catalan numbers, and compute, and get an analytical form for the cardinality of the set. And that's really what the Catalan number is. It's a cardinality of the set.

And so cn is the number of balanced parentheses strings with exactly n pairs of parentheses. And I have c0 equals 1, which is my base case. And that's just setting-- it's an empty string. I'm going to say that empty string is a string, and that's just setting up the base case.

And now I want an equation for cn plus 1. And I need to use the fact that I can get cn plus 1, a particular strain that belongs to this set, where I have n plus 1 parenthesis in a unique way from a string I've previously generated that was part of either the set that had n parentheses-- or it was combined using strings, where alpha was in some set that was maybe generated awhile back with a small n, or something significantly smaller than n, and another thing that was generated, beta, that was generated later, or maybe at the same time, et cetera.

So can someone tell me what an equation would be for cn plus 1 based on the cis that are less than n? So what about c1? Maybe I'll ask-- what about c1? What's c1? 1. c1 is 1 because all I have is this string, that's the only balanced string.

Now I have c0 and c1. What's an equation for c2 in terms of numbers. I want a number for c2 based on c0 and c1. Someone? Yeah.

AUDIENCE: C0 plus c1.

PROFESSOR: c0 plus c1? No, not quite right. How many strings do I have? c0 plus c1. Yeah, actually you're right. Well, the number is right but the equation is wrong. It's not quite that. You get a-- Yup.

AUDIENCE: So when something like c-- c1 times c1 plus c1.

PROFESSOR: c1 times c1 plus c1?

AUDIENCE: Yeah.

PROFESSOR: I think you want to use a c0.

AUDIENCE: OK. c0.

PROFESSOR: c0 Well, that's not quite right either. Someone else. Yeah. OK. You.

AUDIENCE: c0 times c1 plus c1 [INAUDIBLE].

PROFESSOR: c0 times c1-- c2 would be c0 times c1 plus c1 times c0. OK. And if you're setting the alpha-- So here's the thing, you set the alpha, and you choose the alpha, and then you choose the beta. And there's a couple of different ways that you could choose the alpha.

You could choose the alpha from-- you could make it a string that's empty, or you could make it the one string that you've generated so far, which is the standard simple string, the non-empty, the non-trivial balanced string. And you could do that in a couple different ways with alpha. And that's why you have two terms over there.

So the number, in terms of all of the equations I got, they all came out to be the same. It's 2, and that's correct. But this is the equation for it. And so now, tell me what a general equation is for cn plus 1 based on what we've learned so far for the c2 equation? Yeah, back to you.

AUDIENCE: So c0 times cn plus c1 times cn minus 1 all the way to cn times [INAUDIBLE].

PROFESSOR: Perfect. Good. That deserves a cushion. That wasn't me, it was you. And put it right there, breadbasket.

So cn plus 1 equals sigma, so you give me a summation, k equals 0 through n, ck, cn minus k where n is greater than or equal to 0. And you can figure this out, it's not particularly important as to exactly why this is true. You can think about it offline.

But the interesting thing is that this is a generator of, obviously, and it's going to give you a nice looking set of numbers. And I came in early and wrote that the Catalan in numbers up on the board going from c0, c1, c2, et cetera, just in case-- just in case you ever see these numbers in real life, or when you're writing computer programs. Or you're driving on the road, the next time you see a license plate 4862, turn around and tell your mom or dad, hey, that's the Catalan number. And maybe she'll be impressed.

This, of course, you're not going to see on a license plate, but you can always make up a bumper sticker or something, and you can have the c17 as being a bumper sticker on your car. I mean, that's the kind of thing that I'd do. Anyway, so it's just in case you see these numbers-- we might come back to this a little bit later in the class, but--

AUDIENCE: 42 is on that list.

PROFESSOR: 42 is on that list. Yeah, it has to be. 42 is on every list. 42 is the answer to every question. I am glad you guys didn't put 42 down in the answer to every quiz question. It doesn't quite work all the time, all right, but most of the time 42 is a good answer. Most of the time. OK good.

So let's get down to business. So we talked about Catalan numbers as a digression. If you see them you'll recognize them, I think. Let's talk about how we could compute. Let's go back to irrationals and talk about how we could compute square root of 2 and other things to arbitrary precision.

So what I want to do is to talk about Newton's method. And Newton's method is something you probably learned about in middle school, high school. And let's say you have a function y equals f of x where this is x and that's y, the coordinate axes. And we want to try and find the root of fx equals 0 through successive approximation.

For example, we might have f of x equals x squared minus a. And if a is 2 then you're trying to use Newton's method to find the root, and you're going to end up trying to compute square root of 2 or plus minus square root of 2, in this case. But you can go for a particular root, and you're try and converge to that.

So the way Newton's method works is it tries, geometrically speaking, it tries to find tangents-- and a different color chalk would be useful here but I don't seem to see one-- So what would happen is, let's say you are sitting out here, and it's successive approximation method, so this would give you x of i. And now you want to compute x of i plus 1. And what you're going to do is draw a tangent, like so, and find the intercept onto the x-axis, the x-intercept. And that is going to be your xi plus 1.

And you have to write an equation for that tangent. And this is, I guess, trying to figure out how much of middle school math or high school math that you remember. What is the equation for that tangent? Anybody? The equation for that tangent? What do you do in order to compute that tangent? Give me a name.

AUDIENCE: Derivative?

PROFESSOR: Derivative. Thank you. So what's the equation for that tangent? y equals-- Someone?

AUDIENCE: [INAUDIBLE]

PROFESSOR: It's a good think your middle school, high school teachers aren't here. Not you. You already got a cushion. Someone else. Someone else. All right. I'll start giving you some hints here. f of xi plus-- plus--

ALL: f prime.

PROFESSOR: f prime xi. Thank you. Thank you. All right. You get a cushion. I'm getting them. Whatever it takes. Here you go. Yeah. That was left-handed, by the way. I'm actually right-handed, as you know.

So what do we have here? So we have f prime xi. Now come on, let's finish it, finish it.

AUDIENCE: Times i minus xi.

PROFESSOR: Times xi is xi. Thank you. OK. So now I get it. You thought this was too simple a question. You guys were insulted by the question. So that's why you didn't tell me what it was At least, that's what I'm going to pretend to make myself feel better.

So y equals f of xi plus f prime xi, which is the derivative of f evaluated at xi times x minus xi, that's the equation for the line. And that's essentially what we have to do to compute things like square root of 2, which is iteratively applied Newton's method. The only problem is this is all good in theory that we can do-- take that equation turn it into xi plus 1 equals xi minus f of xi divided by f prime of xi. And if you end up doing f of x equals x squared minus a, then you have xi plus 1 equals xi minus xi squared minus a divided by 2xi, which is the derivative of x squared minus a evaluated at xi. And finally you get this equation, xi plus a divided by xi divided by 2.

So it's fairly straightforward. xi plus 1 equals xi plus a divided by xi divided by 2. And if you look at this, remember that a is typically a small number. I mean, it's two, in this case, if you're computing square root of 2. it may even be an integer, maybe a fraction. But you have to do a division here. And remember that since we want to compute things to millions of digits, potentially, these numbers, the xi numbers, are going to have millions of digits.

And so if you end up running this Newton method on a equals 2, then if I simulate what happens without worrying about the implementation, and this is what you'll get. You have x0-- you start with x0 equals 1 with a bunch of zeros. xi equals 1.5 with a bunch of zeros, et cetera. And then, x2 equals 1.41666 et cetera. That goes on. And we're not talking about fractions here, we're talking about floating point numbers that are integers with a certain amount of precision.

So you've decided that you want to compute this to d digits of precision where d may be a million. And so, really, here you would have a representation that's a million digits long that where, basically, everything is zero, and here everything but one is 0, or maybe a couple if you count the 1 here. And here you have all these sixes here, and so on and so forth. And you keep going, and you get x2 equals 1.414215686. And I want to write one more because I want to point out something that's important. 1.414213562.

And what's nice about this, if you go compare it with what you have up there, is that you have quadratic convergence using the Newton's method. And what do I mean by that? Well, quadratic convergence is a nice property. It's much better than linear convergence. Linear convergence would mean that you get an extra digit of precision for every iteration.

So in this case, actually, a quadratic is better. Usually we think of quadratic algorithms, and we are going to throw up. But linear algorithms is what we like. But in this case it's actually a good thing. You have this extended this quadratic rate of convergence where the number of digits that are correct doubles with every iteration, as you can see.

So here you have-- you start with, roughly, if you rounded this up you would get 5. So you're saying that that's one digit of precision in terms of the decimal. And then, now you're talking about 41421356, so that's 1, 2, 3, 4, 5, 6, 7, 8, so that's eight digits of precision here, four here, and so on and so forth.

So that's nice. That's why the Newton's method is actually used in practice because the number of digits doubles. This is a precision. So now you don't get too scared about calculating things that are a million because you kind of go and say, whoa, I mean, that's not so bad, that's only a logarithmic number of integrations. And it's not like you have to run for a million iterations. You go 2, 4, 8, 16, and now that exponential helps you, the geometric series helps you.

So that's the nice thing about the Newton's method. Haven't quite told you how we are going to compute the most important thing here with respect to a divided by xi. So this is just additions. You can imagine that if you have long numbers you'd end up doing addition fairly straightforwardly because you only have to worry about one carry, and so you go off.

And I'm sure you've added up numbers that are in tens, if not hundreds, of digits long but I'm guessing you haven't manually multiplied numbers that are hundreds of digits long. And if you have I'd be impressed, though I won't believe your result. But that's something that is kind of a painful thing to do. And so that's why we need computers. And that's multiplication and-- has anybody divided a hundred-digit number by another 50-digit number? No. So you need computers.

So we've got to be able to do that division there, a divided by xi, using computers. And so that's really the purpose of this and the next lecture If you're only going to do a high precision multiplication here. And try see what an algorithm would be for high precision multiplication because what we're going to do is, essentially, take the reciprocal of one over xi and then multiply that by a. And we need-- division is going to require multiplication. You don't really see a multiplication there other than a times 1 over xi.

And you can say, well, do we really need multiplication? Well, the answer is the division algorithm that is used in practice in most division algorithms have multiplication as a subroutine. So we're going to have to figure out high precision multiplication first-- It's a little bit easier than division-- and use it as a subroutine for high precision division, which we'll do next time.

So now we're sort of back in 006 land, if you will. We have a problem that is high precision multiplication, and we want to find an algorithm for it. And there's the vanilla algorithm that is going to take certain amounts of time, asymptotically speaking, and then there is better and better algorithms. As you can imagine, multiplication is just such a fundamental operation that people have looked at trying to reduce the complexity of high precision multiplication. So if you have n bits-- So what is the standard algorithm for multiplication take if you have n bits of precision?

AUDIENCE: N squared.

PROFESSOR: n squared. But you can do better. And the people are work on it. You can do fast

Fourier transforms. We won't quite do that here. If you take 6.046 you'll probably learn about that. But we'll do a couple of algorithms that are better than the order n squared method, and we'll do one of those today.

So the way they're going to work with integers-- one little point that I need to make before we move ahead here is, generally, we going to say, for the purposes of 006, that we know the digits of precision up front because we want d digits of precision, maybe it's 42, maybe it's 125. In the case of our problem set in RSA we going to have it 1,024-bit numbers, maybe 2,048.

So we know d beforehand. And so what we want is an integer, which is 10 raised to d times the square root of 2, a floor, and which is essentially the same as that, 2 times 10 raised to 2 d. So we're going to treat these as integers, so we don't want to worry about decimal points and things like that. All of these things are going to be integers.

And there's no problem here. We can still use Newton's method. It just works on integers. And let's take a look at how we would apply Newton's method in standard form.

And we won't to really get to the part where we're going to go from division to multiplication today, as I said, we're just going to look at how you can multiply two numbers. So I didn't mean to say that we're going to look at Newton's method. We're going to look at high precision multiplication, and then, eventually, we're going to use that to build Newton's method which requires the division.

So I have two n-digit numbers, and the radix could be-- the base could be 2, 10. Normally, it doesn't really matter. 0 less than x, less than y, strictly less than r raised to n. That's standard for the ranges.

And what I have here is-- the way I'm going to do this is use our favorite strategy, which is divide and conquer, because I have n, which is large, need to break things down, break it into n by two-digit multiplications. And when, finally, I break things down, I get down to 64 bits, I just run one instruction on my computer to multiply the

64-bit numbers. And the standard machines, you would get 128 bits of result back when you multiply 64-bit numbers. So in some sense you only go down-- you don't go down to 1 bit, you go down to 64 and your machine does the rest.

So what we have here is you set x to be x1 where x1 is the high half, r raised to n over 2 plus x0. So x1 is the more significant half and x0 is the low half. And the same thing for y. y1, whoops, r raised to n over 2 plus y0. Now the ranges change. x0 and x1 are smaller, so that's what you have for x0 and x1. Same thing for y0 and y1.

So that's a fairly straightforward decomposition of this multiplication operation. And again, if you do things in a straightforward way you will create a recursive multiply, as I'll write here.

And what you do is you say let z0 equal x0 times y0, z two equals x2 times y2, and-- I missed z1, but z1 equals x0, y1 plus x1 y0. And I have, overall, z equals y equals x1 y1 times r raised to n plus x0 y1 plus x1 y0 times r raised to n over 2 plus x0 y0 zero. And this part here was z0, this part here was z1, and this part here was z2.

And if you look at, it you need four multiplies, 1, 2, 3, 4. And you need four multiplies of n by two-- n by two-digit numbers. And by now, after you have prepared for quiz two, I will just say that it would take theta n square time because you're recursive equation is tn equals 4t of n over 2 plus the linear time that you take for addition.

So this is tn equals 4t of n over 2 plus theta n. And you're assuming that the additions here take linear time. So that's how you get your theta n square algorithm. And we're not happy with that. We'd like to do better. And so how do you do better?

Well, there's many ways of doing better. The simplest way off fairly, substantially lowering the complexity is due to a gentleman by the name of Karatsuba. This is one of those things where if you were born early enough you get your name on an algorithm.

And what happens here is, using the z's that I have out there, you essentially say, look, I know z0 equals x0 and y0, I'm going to go ahead and multiply. z2 equals x2 and y2, go ahead and do that. And now I'm going to write z1 as x0 plus x1 times y0 plus y1 minus z0 minus z2. So you're actually computing z0 and z2 first, and then using them to compute z1.

So someone tell me why this is interesting? And just take it all the way to the complexity of an algorithm. Explain to me why this is interesting and why Karatsuba's algorithm has-- I'll give it away-- a lower complexity than theta n square? But tell me what it is. Someone? Someone other than you. Someone way at the back. Yup. Out there.

AUDIENCE: It's n to the log base 2 of 3.

PROFESSOR: n raised to--

AUDIENCE: Log base 2--

PROFESSOR: --log base 2 of 3. That's exactly right. And now, why did you get that? Explain to me how you got there.

AUDIENCE: If you're using three products--

PROFESSOR: That's right. So first insight is that we're only doing three multiplications here. Additions are easy. And we're doing three multiplications as opposed to four. So tell me how that equation changed, of tn. tn equals--

AUDIENCE: 3 [INAUDIBLE].

PROFESSOR: 3 times t of n over 2 plus theta n because you're doing three multiplications rather than four. Multiplications are the complicated operation. Divisions are even more complicated, but additions are easy, and you could do those in linear time for n-digit numbers.

And so, if you do that and then you go off and you say, well, that tells us that tn equals theta of n raised to log two of three, which is, by the way, theta of n raised to 1.58, roughly speaking. And I do not want to compute that to arbitrary precision, though I could. But that goes on and on. Why don't you grab this after you're done.

But that just goes on and on. 1.58 is a rough approximation. That's an irrational number too, assuming you think that irrational numbers exist. That's an irrational number.

So good, that's really all I had. By the way, it's 1.58490625. I really should have written that down. 1.58-- in the context of this lecture I think it's important that we get at least a few digits a precision.

Now you can imagine that you could do it better than this. And it turns out that-- we'll talk a little bit about this next time-- But you can imagine breaking this up into not n over two chunks, but n over three chunks. Why don't I just break up x into the top third, the middle third, and then the bottom third, and then try and see if I can get away with fewer than eight multiplications? Because the original thing would have taken eight, and if I can do less than eight, maybe I can reduce that 1.58 number. So that's a little bit of a preview for what we'll do next time.

But what I'd like to do is do a demo. And I want you to run that-- it's out there so blank that out for a second. What I'd like to do is really look at a different problem than square root of 2, and show you a demo of code that Victor wrote that computes this particular quantity that, you would argue, is irrational, to arbitrary digits. Though we'll probably only go up to about a thousand today.

And if we just look at-- root 2 is kind of boring, right? It's been around for a while. Let's go back and remember high school geometry. So I think your high school teachers would like this lecture. Nice little review.

So what is that about? That's supposed to be a circle, I think, as you've forgotten. That's supposed to be a circle. And the circle here is a really big circle. It's a trillion units long. I'm into big numbers today, big numbers.

And the center of the circle is c, c for center. That is what's called a radius, in case you'd forgotten. And that's b. And this is also a radius, and that's a.

And what I'm going to do is I'm going-- I want to make a drop a little, I guess, perpendicular down, which is one unit high. So the way that this is structured is that this is one unit high, this obviously is-- someone tell me what that is, CB.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Half a trillion. That's half a trillion. And obviously, CA is half a trillion. And if I call this D, somebody who remembers high school or middle school geometry give me an equation for AD. And that's definitely worth a cushion. so what's an equation for AD guys? Yeah. Go ahead.

AUDIENCE: The radius, half-trillion minus the square root of--

PROFESSOR: Square root of-- Yeah?

AUDIENCE: Minus the square root of [INAUDIBLE] square minus 1?

PROFESSOR: Perfect, perfect. OK good. So AD equals AC minus CD, and that is going to be half a trillion-- that is 500 billion, a little bit less than bailout money, but it's close-- 500 thousand minus square root of 500-- this, when you start squaring this, of course, is not even real, real big money. But 500 thousand squared minus 1. So forget the square root of two. You can put the screen down. So is it on, the projector?

AUDIENCE: It's on, it just needs a [INAUDIBLE].

PROFESSOR: OK. You can turn that on.

AUDIENCE: Rolling.

PROFESSOR: So what we're going to do is, I'm just going to show you the digits of this crazy quantity being computed to tens of thousands of digits. And you argue that this is it something clearly that isn't a perfect square. You took a perfect square, subtracted 1 from it, and so you have an irrational quantity that is going to go on and on. And let's see what that number-- what it looks like. OK?

Get out of the way. I hope you can see from the back.

AUDIENCE: Oh man, really?

PROFESSOR: Looking pretty good so far, looking pretty good.

AUDIENCE: That's crazy.

PROFESSOR: Somebody see the numbers somewhere else? Have you see these numbers before? Like 20 minutes ago, like right in front of you? OK All right.

So I think that's a good place to stop. If you want an explanation for this I think you can go to section tomorrow. I'm going to use a some attendance tomorrow. All right. Happy to answer questions about the rest of the lecture, and thanks for coming.