TheFourierTransformAndItsApplications-Lecture22

Instructor (Brad Osgood):All right, by popular demand, today, I'm going to talk about the basics of the fast Fourier transform algorithm, the famous FFT. We're not gonna do it in all detail, that is, I'm not gonna carry it out to the bitter end. And in fact, it's actually a highly refined art. It's almost more art than science these days. There's actually, these days, not one single FFT algorithm. There are a number of them. They share common ideas, they share similar properties, but they are often tailored to particular applications for each context.

But what we're gonna talk about today is the main idea that underlies any of them and it's the one that was sort of originally put forth by Cooley and Tukey back in the '50s I guess it was, something like that. I can't remember the dates on this.

It's very interesting. You should read about the history of these things and as you've probably heard, I mentioned in the notes, the idea behind the fast Fourier transform algorithm actually was known to Gauss way back when, when he was trying to carry out calculations of estimating the orbit of an asteroid. So a lot of interesting history.

Now, what this means in particular, I have to give you a little caution on this, is that there's sort of one more general topic on this fast Fourier transform that I would ordinarily talk about and that is convolution, circular convolution, so I'm not going to do that. I will leave that up to you to read. There's not much. There wouldn't be much to do other than talk about the formulas and basic applications. The applications we'll do in the context of linear systems coming up, so you'll see some of that later.

But the definition of convolution and some of the basic properties of convolution again, is they're similar to the continuous case. I'll let you just read the notes about them, all right? And I think there are some problems on that. I think there are already some problems on that, actually.

Okay, so that's the plan. This is going to put us a little bit behind because I was going to start talking about it sooner. I was gonna start talking about linear systems, linear time and variance systems, so we'll try to trim things a little bit after we leave the discreet world, after we leave this discussion of DFT.

So let me explain what's going on here. Let me set the stage for you. Here's the FFT algorithm. Now, let me remind you that – well, there are a couple ways of looking at it. Maybe the simplest way of understanding what the challenge is, as I mentioned last time, to write the DFT as an N x N matrix. So you can write the DFT as an N x N matrix. It's a very simple entrance, very simply described. F is just Omega to the minus NM, where remember I'm using this notation now just to simplify things so I don't have to constantly write complex exponentials.

So Omega is the primitive N root of unity, so Omega to the minus nm is just e to the minus 2pi nm over capital N. And the way we index things is from 0 to N minus 1. Indexed from 0 to N minus 1.

So to compute discreet Fourier transform, the discreet signal is to multiply the matrix by the column vector, is to regard F as a column vector and multi-matrices. All right, so there is an end-by-end matrix that would seem to require N2 computations. Never mind that you just also have to add up the N elements, but the main computation and intensive aspect of it is the multiplication.

So this requires – would seem to require – so let me just use the big O notation here, meaning that some multiple of N2 to the dominant term is N2, O(N2,) capital O(N2), big O(N2) operations.

And how could it require any less than this? I mean everything has to be accounted for here. I mean you can't just throw things out while computing the DFT. It would seem to require every element of the matrix be involved, when in fact, what the FFT algorithm does is reduce this enormously. The FFT algorithm reduces this to big O of N log N operations, usually meaning log base 2 here, but it doesn't really matter.

Now, if we're saying why something like this might be possible because it's not obvious, you have to realize that that's an enormous savings in computation. To go from N 2 to N log N is a huge savings. For example, it's a huge savings – so N is equal to 103, if N is equal to 1,000, and of course, N2 is a million, that N log N is 103 log 103, log of 103 is 3, so it's just 3 times 103. So it's a thousand operations, or a constant times a thousand operations, versus a million operations. All right, that's an enormous savings. And of course, the larger N is, the savings are even more dramatic.

Now, why should you be able to do this? Well, there are a couple of different sort of intuitive reasons why this should be the case. The intuition is one thing; carrying it out is something else. Let me tell you one way of thinking about it that we're not going to do, although it's discussed a little bit in the notes, and this is a very common way of approaching the DFT and the FFT algorithms, and that is sticking with a matrix, notation of the matrix idea.

One way is that there are a lot of structures to the matrix and there's a lot of structure to the DFT, so when you reduce it, you want to exploit the structure in the DFT. Well, that's not – that's a vacuous statement. Of course, you're going to exploit the structure in the DFT. It depends on only parameter. It depends on the parameter Omega and this complex exponential, and that has a lot of nice algebraic properties via ray of properties of the complex exponential. That's the main thing that's involved.

Now, in the matrix approach, what this amounts to, the reason why there can be such a dramatic reduction is that you can factor the DFT into a product of other matrices, which have lots of zeroes in them. So one approach is to write the DFT matrix as a product of simpler matrices; that is, to use the algebraic properties of the exponential, to carry out this matrix factorization. The simpler matrices have lots – as it turns out, the simpler matrices have lots of zeroes in them and when matrices have lots of zeroes in them, they effectively don't count in the calculation. When you multiply by zero, you're not doing anything. Simple matrices have lots of zeroes, so fewer multiplications are required. Fewer multiplications are required to carry out the matrix multiplication, to carry out the product. That's one reason, that's one approach to the FFT algorithm, and in fact, that's explained in the notes. I'm not gonna do this; I'm not gonna go through that particular approach, but that's often done, all right? And in some of the sort of highly refined approaches to the FFT, you often find they're presented in terms of matrix factorizations, and if any of you have taken 263, or are taking 263, you know that matrix factorization is a big business. Writing a matrix, writing a complicated matrix as a product of simpler matrices is something that can often, not only simplify a calculation, but can also often really illuminate some ideas that you wouldn't see, that are a little bit, that are somewhat subtle. They can't be somewhat subtle.

So it's a standard approach that we're not gonna do, but you can easily find many references to the literature on this approach. We are gonna take really a more directly algebraic approach that's based on the formula. So although I'm thinking in terms of writing a matrix, although we did the basic counts of the number of operations in terms of the matrix, I'm actually not going to work with a matrix to develop the algorithm. I'm just gonna work with the formula for it for the DFT as we have written it before and I wanna use the – I wanna exploit the algebraic properties of the complex exponential to write it in a more – I don't know – simple form or just – well, you'll see. You'll see.

It's one thing to have the idea and it's not an obvious thing to have the idea. It's not a thing to really carry it out. It's really quite beautiful and I say this as someone who's orientation in life is not Fourier algorithms necessarily, so I really have a grand appreciation for it and the whole idea behind it.

All right, now, let's write down the formula. So once again, as always, the Fourier transform of the discrete signal is given by – we write it like this. The F component – when we write it out like that – is the sum from – so I'm indexing from 0 to N – 1, so the sum from n = 0, N – 1 of F(N), the N component of F, times – if I use this notation for the complex exponential, just the Omega to the minus N x N. It's a simple formula. All right, now, what we're gonna do is we are going to write the sum as a sum over even and odd powers. We're gonna write the complete sum as a sum over even and odd indices and then also, therefore, powers of Omega. Well, let me just say even/odd indices. And in doing so, it's the algebra that's gonna allow us to do this in a helpful way. You can say I'm gonna do that. It's one thing to, again, it's one thing to say that; it's another thing to actually give you any resulting savings. You'll see in just a sec.

The purpose of this is to try to write the N for order DFT as a combination of two DFTs of order N over 2, capital N over 2. The purpose is to write the order N DFT as a combination, really a sum – there's a little bit more to it than just a straight sum – as a combination of two DFTs of order N over 2, all right, and then iterate. And as you write these DFTs of order N over 2, you apply the same rules to the DFT of the order N over 2 to write them as combinations of DFT of order N over 4 and then iterate and so on and so on and so on. Okay? Now, so first of all, to make this work, you have to assume that N is even and if you want to keep iterating, you have to assume actually that N is a power of 2. So to do this, you need to assume that N is even. Even to get started, we need to assume it's even. To iterate, you need to assume that N is a power of 2.

So this is where all these special things come in when you have a signal and you wanna apply it and you wanna compute numerically a discrete Fourier transform, and your signal, your discrete signal doesn't happen to have the length of a power of 2. What do you do? They zero padding; they do all sorts of things like this. There actually are other Fourier transform discrete Fourier transform algorithms, or DFT algorithms, that don't require this, but they all require something like this. So sometimes you have to modify your signal and again, I'm not gonna get into this. I'm just gonna talk about the algorithm per se assuming all these things. But there are ways of trying to address this and they can introduce some extra complications, so if the signal is not a power of 2, you add a bunch of zeroes, so-called "zero padding," until you get it to be a power of 2. Of course, that's not the original signal, so you've changed things a little bit and have those changes affected things? Well, that's something that depends, again, on the particular context. So we're gonna make this assumption. So sue me. I'm gonna assume this now. I'm gonna assume that N is even and then in general to iterate, I'm gonna assume that N is a power of 2. Okay, now, it depends – the algorithm and this way of arranging things, way of rearranging things, depends on the algebraic properties of the complex exponential. I want to introduce some notation here. Sorry; you know I'm always introducing some extra notation, just so I can keep track of things without – a little bit more easily, a little bit more neatly. I wanna keep track of things, so I use a notation to keep track of powers, really, of the complex exponential. That's what's involved. All right, so let me write – sorry, but let write this. I'm only using it for this performance only. Omega P (q) is e to the 2p q over p, because what's involved is both the denominator here and the power that you're raising it to.

I'm gonna use that notation and I'm gonna write my Fourier transforms in terms of that notation. Now in terms of this, how we write down the usual properties of complex exponentials because I'm gonna use them all, well, if I put down p, say q1 + q2, that's taking e = 2pi(q1+q2) over p, and of course, complex exponentials being what they are, that multiplies. So this is Omega p (q1) times Omega p (q2). Nothing there. Absolutely everything I do today is gonna be – is dead simple, okay? A high school student could do this if you explain to them at every stage what to do. But there's nothing here that's at all involved. It's just extremely clever. Now, let's look what happens when I change the power, or change the order maybe I should say. First, by the way, let me just comment here. So Omega, for the terms that are entering into the Fourier transform, DFT, Omega and n and m, minus nm is what is the power that occurs here, so that's Omega to the minus nm that is equal to each of the e = 2pi nm over N. Okay, so what is our – I wanna follow my notes here because I don't wanna get this wrong. So Omega, so I'm assuming that N is even, capital N is even, so what is Omega, the sort of N over 2nd order when I have such a power here, which is what it comes up nm in the transform, so that is – let's work it out – e to the minus 2pi nm divided by N over 2. So it's 2p nm over N over 2.

The two comes upstairs. This is e to the minus 2pi times 2nm divided by N. That is to say, if in terms of my notation, this is Omega, the F order complex exponential weighs to the power of minus 2nm. So in words, a power of the half order exponential is an even power of the four exponential. That's one way of saying it. If I think of N as sort of the order of exponential that I'm worried about, then in words, say a power, in this case nm, half order exponential, that's N over 2, is an even power, that is, 2nm, of the 4 exponential that's N.

I think I'm developing tennis elbow, by the way, over the course of this quarter. I'm in a lot of pain, but I can play through it. All right. So that side is the even powers. What about odd powers? Well, odd powers – I'm thinking about N being odd. Let's go over to odd powers, the power being 1. So then I'm gonna look at Omega – careful, careful here; careful, careful here – Omega N of 2n + 1. Now be careful. I don't want to get any wrong. 2n minus – 2n + 1 because all of my powers are minus m. So that's an odd power of N. See, once again, powers of N over 2 here led to even powers of N, so I wanna look at what happens to odd powers of N and relate those to certain powers of N over 2. So what is this? This is Omega N – 2nm – m, okay, right. Now, remember, complex exponentials being what they are, this is the product of the two, the product of the N for exponential N raise to this power and raise to this power. So that's to say this is Omega to the nth order exponential to the power -2nm times the nth order exponential to the power of minus m. I did get my minus signs right over here, right? Yeah, okay.

All right, now, this you've already seen. This is the first term is Omega, the half-order exponential at the power nm, minus nm, times – this one I leave alone; nothing I can do with that. Okay, so once again, what happens to even order powers? An ordinary power of N over 2 is an even power of N, all right? Or you can say it this way. An even power of nth order exponential is an ordinary power of the N of the half-order exponential. Here, an odd power of the 4 exponential is a product of an ordinary power of the half-exponential, half-order exponential times this extra factor here. Okay. All right, now, let's go back to the formula for the DFT and plug this in. I'll put this into the formula for the DFT. All right, so what is that? That is the DFT, the nth component once again, is to just write it down again, is sum from N=0 to N-1 of the nth component of F times the power of the complex exponential, so I'm writing this in the form Omega N – nm. Okay?

All right, now, I'm gonna write this sum as a sum over even indices and odd indices. All right, so I'm gonna write this as sum over even indices plus the sum over odd indices. So what does that mean? Well, that means – I mean what does it mean in symbols? That means that the discreet Fourier transform is gonna be – I'm gonna take the sum from N=0 to N over 2 minus 1. We'll write it down and make sure we have it right. So this is N over 2 minus – let me write this a little more – give us a little more room here. The Fourier transform of F, the nth component is gonna be the sum over even indices. That's half the indices are even; half the indices are odd. So if I go to the half indices, there are N over 2 of them here, from 0 to N over 2 minus 1 or N over 2, those are the even indices and that is gonna be F of 2n, right, since I'm summing over the even indices, times Omega N, -2nm plus – to be continued on the next board – the sum over the odd indices, and again, half of the indices are odd, so it's gonna be the sum N = 0 up to N over 2 minus 1, the odd indices. So an odd number is the form 2N plus 1. Omega N, minus 2n plus 1m. Now everything's accounted for there, right? Everything's accounted for. These indices, 0, 2, 4, 6, 8, and so on, these are the indices. The components 1, 3, 5, 7, 9 and so on and so on; they're all there. I haven't lost anything. Okay, now, watch this. Put that up there and we put this up there. How about that for professionalism? And now – almost. Okay, now use the way that I have written the nth order complex exponential in terms of complex exponentials of order N over 2, the kind of stuff that I'm just about to erase.

So the sum over the even indices gives me the sum from N = 0 to N over 2 minus 1. F of 2n and this is the ordinary power N nth power of the half-order complex exponential, capital N over 2. Plus the sum of the odd indices over 2 minus 1 F of 2n plus 1. So what happened here, here was Omega n minus m times Omega N over 2 minus nm, if I had that correct. Do I have that correct? I believe I do, so that is what happens when you look at this complex exponential for an odd index, for an odd power. Now this doesn't depend on the sum over N, this doesn't depend on N, so that comes out of the sum. So let me just rewrite this quickly. So this is the sum from N = 0 to N over 2 minus 1 F of 2n times Omega N over 2(n-nm) plus Omega N minus m times the sum – very similar sum – and equals 0 to N over 2 minus 1 F(2n) plus 1 Omega of half-order exponential minus nm. Make sure I got all this right. Okay. Okay, nm, mn, doesn't matter. Again, everything is there, right? Everything is accounted for, mn, everything is accounted for. All the indices are there; all the components are there.

All right, now, this is looking like you've expressed an nth order DFT in terms of two DFTs of order N over 2 because the DFT of order capital N over 2 involves exactly the complex exponentials of power, to the power of the denominator or the power you're dividing by is capital N over 2. So this is looking like – this is almost, but not quite, for reasons you have to understand, this is almost the Fourier transform of – let me write it like this – of N, all right. The nth order of DFT looks like the N over 2nd Fourier transform of the even indices. If I take F, but I just divide it by the even indices, plus this power Omega and minus m, the DFT of order N over 2 applied to the odd indices, which is what I'm trying – I'm trying to get something like this. I'm trying to say you can evaluate an nth order DFT in terms of a combination of two DFTs of order half that. Not quite right. It's not quite right because – you have to think about this – the DFT of order n accepts an n-tuple and returns an n-tuple. The DFT of order N over 2 accepts an N over 2 tuple and returns an N over 2 tuple. So how are you getting an n-tuple out of two N over 2 tuples, and how would you like to say that a couple times fast? F (N over 2) accepts an N over 2 tuple and it returns an N over 2 tuple. So writing this statement can't be right, all right? You're missing something because if I just had the right hand side, the right hand side makes sense because F even is an N over 2 tuple, F odd is an N over 2 tuple, but what returns are two N over 2 tuples, not an n-tuple, which is the nth order Fourier transform of F. So we have to tickle this a little bit. We have to tickle the formula. Tickle the formula. Sounds so cute. Sounds so human. All right, now, I have to find – I have to get the formula for the Fourier – for the nth order Fourier transform of F for m going all the way from zero to N minus 1. I need to get all N components of it.

Now, for the first N over 2 minus 1 components, these are okay. That formula would apply. For m = 0 up to N over 2 minus 1, we can use the above formula, the formula above. That is to say – let me just rewrite it; make sure I haven't missed anything here – we can write, and all is well, the formula of the Fourier transform, the discreet Fourier transform of F, the N for this Fourier transform is exactly the N over 2nd, the Fourier transform of order N over 2, DFT order of N over 2 at m plus Omega(n-m), the N over 2 half-order Fourier transform of F odd, the nth component of that. All right, that's correct. That's correct for m going from zero to N over 2 minus 1. Okay? Now, that gets half of it. So what about the second half? So what about indices m indices from N over 2 up to N minus 1? All right. Write these in the form m plus N over 2, where m goes from zero up to N over 2 minus 1. That is, if m is ranging over the indices that index the half-order DFT, write the indices that index the second half of the nth order DFT in this form, all right? That is, the indices from N over 2, N over 2 plus 1, up to N minus 1 are of the form – if I started with zero here, I get N over 2, N over 2 plus 1. Then I would get N over 2 minus 1 plus N over 2 gives you m minus 1. It covers all of them. Okay? That covers all of them.

All right, now, what I mean by this, you have to write right now, what happens if I substitute this expression, an index in this form, into the expression that I have for the nth order DFT, which I am now stupidly erasing? Then the nth order Fourier transform at an index of the form m plus N over 2 is according to what I've already written down, which is valid for all of these indices, sum from N = 0 to N over 2 minus 1, F of 2n – this is what I had before – Omega N over 2, m plus N over 2, N times N plus over – minus – there were 2n, right? Plus Omega N minus m plus N over 2 times the sum from N = 0 to N over 2 minus 1 of F of 2n plus 1 the odd indices Omega – what was here? The same sort of thing, right? N over 2 minus m plus N over 2n. All right? Once again, let me make sure I have not screwed the pooch on this. Not, good, all right. Everybody with me? So that was the formula for the DFT writing it as the sum over even indices plus the sum over odd indices, but now I'm seeing what happens if you plug in an index in the form m plus N over 2, and again here, m is gonna range from zero to N over 2.

Well, now, let's just see what happens to these complex exponentials. That's what we have to answer. All right, but it's not hard. A high school student could do this if you told him what to do at every stage. Omega – a lot of research is like that. It's a big secret. N over 2 minus m plus N over 2n, so that's Omega N over 2 minus mn plus N times N over 2 – minus N times N over 2. Now once again, these exponentials being what they are, this multiplies. This is Omega of N over 2, minus mn times Omega N over 2, minus N times N over 2. But this is a complex exponential. This is an N over 2 root of unity raised to the N over 2nd power, or an integer times N over 2nd power. So this is equal to 1. This is like e = 2pi over N to nth power. So this is 1. So what remains there is just Omega to the N over 2 minus nm. Omega to the N over 2 minus nm – mn, same thing. And that comes up in both these sums, right? Yeah, it's the same thing in both these sums, so in the sum over the odd indices and the sum over the even indices, the only other thing you have to figure out is this up front. That bracket out front. This comes up in both sums, even and odd indices. This expression. The only other thing that comes up is the factor out front, which is Omega – what is it? Omega N minus m plus N over 2. So that's Omega N minus m minus N over 2, and again, complex exponentials being what they are, this is Omega N minus m times Omega N, N over 2. Minus, it doesn't matter, minus.

Now, what is that second expression? This expression, fine, that's what we had before actually. That's what we had before as a factor out front. What about this expression? Well, this is e = 2pi over N raised to the minus N over 2 power. So what happens is the N cancels out the N, the 2 cancels out the ½ there and it's e = pi, which is also known as minus 1. So the factor out front changes to minus what is was before. So Omega, in other words, Omega N, minus m plus N over 2 is just minus Omega N minus m. Correct? Correct. All right, so what is the actual retail value of the expression? So we have Fourier to nth order of the DFT of F and an index to the form m plus N over 2, which is what I'm worried about here, for little m going from zero to 1, 2, 3, up to capital N over 2 minus 1 is equal to the sum over one half of the first sum – the first sum didn't really change very much – over N over 2 minus 1, F(2n) – that's the sum of the even indices – Omega N over 2, minus nm. Was that right? Was it right, the first sum didn't change, the first factor didn't change? Help me here. Thank you. Yes, I believe that's the case that the first sum didn't change. The second sum – the only thing that happened in the second sum was this factor out front and the factor out front changed to minus itself. So minus Omega nm times the sum from N = 0 to N over 2 minus 1, F of 2n plus 1 Omega N over 2 minus nm.

That's the result of plugging in an index of the form little m plus N over 2 into the original form we had to decompose the nth order DFT into the sum over even indices plus sum over odd indices. Okay. But look! Once again, this is okay – m is ranging from zero to N over 2 minus 1 here. This is again a DFT of order N over 2 returning an N over 2 tuple and this is a DFT of order N over 2 returning an N over 2 tuple. So this says – it's exactly what we want – this says that Fourier transform, the discreet Fourier transform of order N at m plus N over 2 is equal to the discreet Fourier transform of order N over 2 of F of the even indices applied to m minus Omega N, minus m, the Fourier transform of N over 2 of F applied just keeping the odd indices at m, and this is again, m from zero up to N over 2 minus 1. All the components of the nth order DFT of F have now been accounted for and they are computed by finding half-order DFTs. The first half of them – let me summarize. The first half of them are computed by the first formula that we had up there. The second half of them are computed by this formula and the only difference between the two is this is a minus sign here instead of a plus sign. So in summary, I'm gonna erase this, but I'm gonna write it again. Glutton for punishment that I am. So to summarize, for m equal zero, 1, up to N over 2 minus 1. The Fourier transform, the nth order Fourier transform of F at m, the first half of the coefficients, is the Fourier transform of order N over 2 applied to the even indexed elements of the original input plus Omega nm, the Fourier transform order N over 2 of the odd indexed version of the original input, and the second half – so that's the first half of the indices. For the second half of the indices, the nth order Fourier transform of F at m plus N over 2. Again, m is just ranging from zero to N over 2 minus 1. That's the index range for the half-order DFT. This is equal to the half-order Fourier transform F applied to the even indices, evaluated m, minus Omega minus m of the half-order Fourier transform of F the odd indices evaluated at m. And that's all of them, okay? That's all of them. That writes the

DFT as a combination of two half-order DFTs or maybe I should say two DFTs of half the order.

And now, iterate. If capital N is a power of 2, then you cut each one of these things down, okay? You cut each one of those things down and so on and so on and so on till you're down to a single 2. Now, here's what I'm not gonna do because we're almost out of time, although that's never stopped me before, but it's what I'm not gonna do. So I'm not gonna show you – so this is the FFT algorithm, or rather maybe I should say this is the basis for the FFT algorithm. All right, again, the idea is that half the work is twice the fun somehow. I mean you're cutting down the nth order DFT to 2 N over 2 DFTs and then iterate. Now, so what I'm not gonna show you is why, theoretically and practically, this gives you the savings going from N2 to N log N. That's discussed in the notes and if you have any experience with what they call computational complexity, you'll see how the argument goes. If you don't, we can work through it some time, but I'm not going to go through that part. I don't want to somehow detract from this beautiful arrangement of the algorithm we have here. The second thing that I'm not gonna – so I'm not gonna do that, but this savings, cutting it down by 2 each stage, reduces the number of steps from N2 to N log N. I'm also not going to show you the sort of final form of the algorithm. If you look – and I actually don't have this in the notes, but if you look in other books, it's quite common to display, to sort of carry the step out, and there's this gorgeous – or not, depending on your point of view – diagram called the "Butterfly Diagram" about how the individual inputs, original inputs, of the discreet signal sort of snake their way through the algorithm if you keep iterating it like this.

You know, what happens to the zero of output, what happens to the first output, what happens to the even inputs – it shouldn't be even – but the zero input, the first input, and so on, what happens to the even input, what happens to the odd input and so on. So I'm also not gonna show you that and that I actually don't do in the book, in the notes, but you can find that elsewhere if you're interested in it. And finally, I'm also not gonna show you – I'm sorry for all these things I'm not gonna show you – I'm also not gonna show you how this leads to the matrix factorization, all right? But that's exactly what happens. I mean you can translate from one to the other, that is, the fast Fourier transform algorithm leads to a factorization of the DFT matrix into a product of matrices where there are lots of zeroes. And the fact that you have lots of zeroes is another reason why there's such a savings in computation because when you have zeroes in your matrix, that's effectively not an operation, not a multiplication that happens. So those are the things I'm not gonna show you. But what I did want you to see is I wanted you to see how this arrangement works. The crucial first step in going from an nth order DFT to an order to 2 DFTs to order N over 2, that's really the crucial observation here and how this was done. It really is quite striking and of course, how much of the modern world's economy depends on this algorithm. That's also really – I think it's right up there at the top of the list for the number for the algorithms that are in constant day-to-day use. So we say goodbye right now to the DFT and its properties. Next time, I'm gonna talk about linear systems. We have to do a little rearranging here because I wanted to be sure to cover this, by popular demand, and then we'll move on to a discussion of linear systems, linear time variance systems, and then higher dimensional Fourier transforms. Thank you all.

End of Audio

Duration: 51 minutes