Lesson: DC5: FFT – Part 2
(Slide 1) FFT: High-Level
***

FFT: high-level

Goal: Evaluate poly $A(x)$ of deg. $\leq n-1$ $(n=2^k)$
at n points $=$ n^{th} roots of unity

Define $A_{even}(y)$ & $A_{odd}(y)$ of deg. $\leq \frac{n}{2}-1$

Recursively evaluate A_{even} & A_{odd}
at $(n^{\text{th}} \text{ roots})^2 = \frac{n}{2}^{\text{nd}}$ roots

Then, $O(n)$ time to get $A(x)$ at n^{th} roots
$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n\log n)$

We now have all the pieces to define the FFT algorithm.

Let's start with the high level idea of the algorithm once again. We're given a polynomial of x. We're given this polynomial by its coefficients. Let's assume this polynomial is of a degree at most n-1, where n is a power of 2. We want to evaluate this polynomial at n points.

Now, in the end, when we do polynomial multiplication, we actually want this polynomial A(x) at 2n points. In order to obtain at 2n points instead of n points, we can just pad the polynomial, the coefficients with zeros, so that we view the polynomial is a degree 2n-1 polynomial.

Now, what are the n points that we're going to choose? We're going to choose the nth roots of unity as our n points which we're going to evaluate the polynomial of A(x) at. Now since n is a power of two, so n=2^k for some positive integer k, then, we know that these n points, the nth roots of unity, satisfy the +/- property. So the first n/2 are opposite of the last n/2. And, the other property is that the square of the nth roots are the n/2$^{\text{nd}}$ roots.

Now we're going to take this polynomial A(x), and we're going to define a pair of polynomials, Aeven, and Aodd. We take the even coefficients, and that defines this polynomial Aeven. We take the odd coefficients of A(x), and that defines this polynomial Aodd. And the degree of these two polynomials is at most n/2 - 1. So we went down from a polynomial of n-1 degree to two polynomials of degree at most n/2 - 1.

Now what we saw earlier is that in order to attain A(x) at these n points, we need to evaluate Aeven and Aodd at the square of these points. So what we do is we recursively evaluate Aeven and Aodd at the square of the nth roots.

What's one of the key properties of the Nth roots of unity? It's that the square of the nth roots equals the $n/2^{nd}$ roots. And there are n/2 such roots.

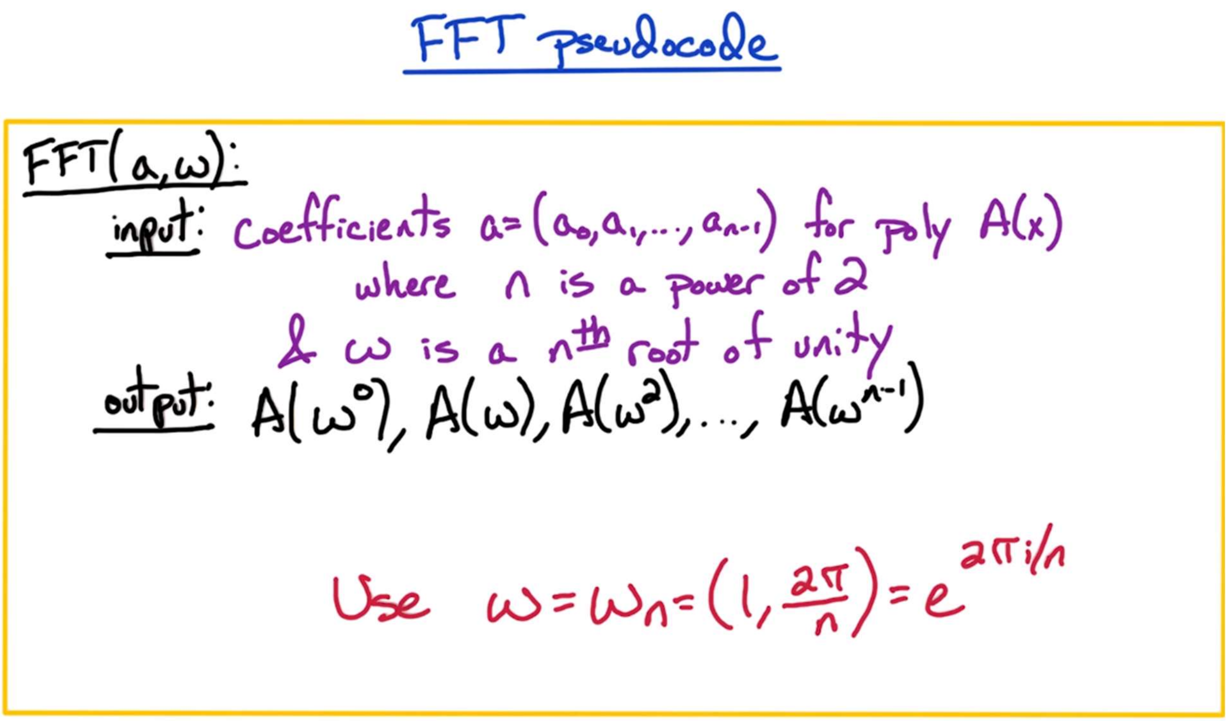
So in order to obtain A(x) at n points, we need to evaluate these two polynomials, Aeven and Aodd, of half the degree, at n/2 points. So, we've got two subproblems of exactly half the size, and these subproblems are of the same form. We want A(x) at the nth roots, Aeven and Aodd at the n/2nd roots.

Finally, given Aeven and Aodd at these $n/2^{nd}$ roots, it takes O(n) runtime to get A(x) at the nth roots. We simply use this formula from before: A(x) = Aeven(x^2) + x Aodd(x^2). So it takes O(1) runtime to compute A(x) for each x, and there are O(n) x's. So, it takes O(n) total time to compute A(x) at the nth roots of unity.

Finally, what will be the running time of this algorithm? Well, for the original problem of size n, we defined two subproblems of size n/2. We recursively solve those to get the polynomials Aeven and Aodd at the $n/2^{nd}$ roots. And then it takes us O(n) to merge the answers to get A(x) at the nth roots.

This is the common recurrence (T(n) = 2T(n/2) + O(n)) that you've seen many times, probably for mergesort, and stuff like that. And this solves to O(n log n). And this is the sketch of the algorithm to take a polynomial in the coefficients form and return the valuation of the polynomial at n points where the n points are the nth roots of unity. And, it does so in time O(n log n).

(Slide 2) FFT: Pseudocode
***



Now we can detail the pseudocode for the FFT algorithm.

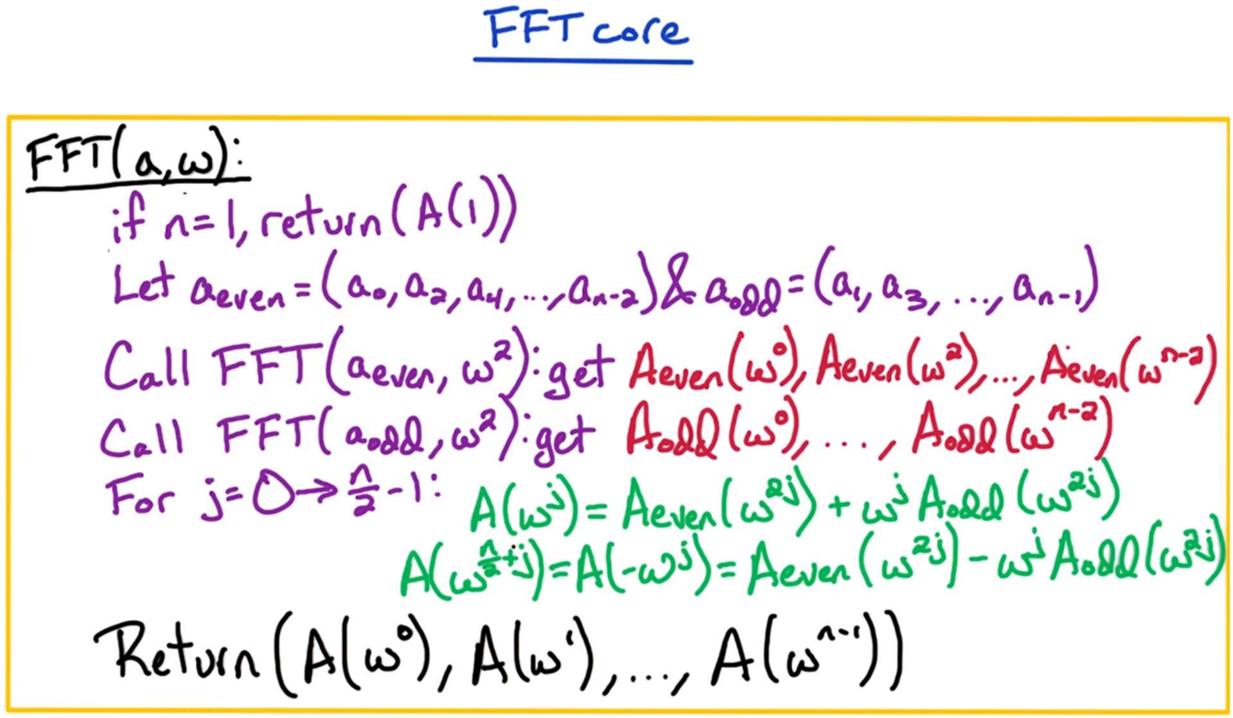
Now, the first input is vector a = (a0,a1,…,an-1), which are the coefficients for this polynomial A(x), and we're assuming that n is a power of two.

What is this second input? This ω? ω (omega) is an nth root of unity. For now, just think of ω as ω_n. In polar coordinates, this is (1, 2π/n) - this is $e^{2\pi i/n}$. For now, you can view ω as ω_n.

But, later, we're going to use this exact same algorithm - this is identical pseudocode with a different ω. We're going to use ω as ω_n^{n-1}. And that's going to allow us to do the inverse FFT. In inverse FFT, we're going from the value of the polynomial at n points to the coefficients.

Now, what's the output of the FFT algorithm? Well, what's its value of the polynomial at the nth roots of unity? If ω is ω_n, what are the nth roots of unity? Well, it's just the powers of this. So, the output is $A(\omega^0)$, A(ω), $A(\omega^2)$, and so on, up to $A(\omega^{n-1})$. When ω = ω_n, this gives A evaluated at the nth roots of unity.

(Slide 3) FFT: Core
***



Let's dive into the FFT algorithm. It's a Divide & Conquer algorithm.

So let's start with the base case. The base case is when n=1. What are the roots of unity in this case? Well, it's just 1. So we can simply return A(1).

Now, we have to partition this vector A into a_even and a_odd. These correspond to the polynomials Aeven and Aodd. So, let a_even, the vector a_even being the even terms in the vector a. So a0, a2, a4,…,an-2, and a_odd are the odd terms: So, a1, a3,…,an-1. The input vector a was a vector of size n. These two vectors a_even and a_odd that we just defined are vectors of size n/2.

Now, we have our two recursive steps. We call FFT, the same algorithm, with the vector a_even, and instead of ω, we use ω^2. And, we also call FFT with a_odd and also ω^2.

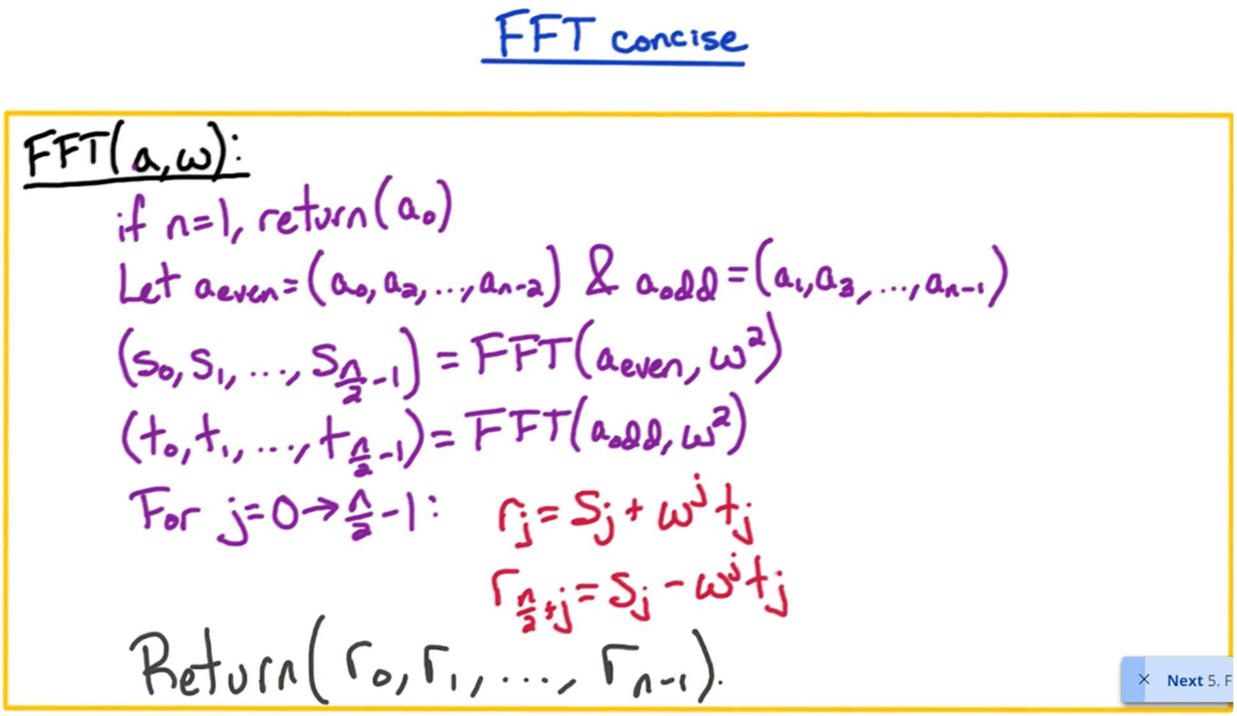
What do we get back from this call? What we get back is Aeven at the square of these n points, which are these n/2 points. ω0, ω^2, … up to ω^{n-2}. So, if ω is the nth root of unity, then we get Aeven at the n/2nd roots of unity. And similarly, we get Aodd at the n/2nd roots of unity. Notice that if ω = ω_n, then the jth of these points squared is the jth point in this sequence or actually the j+1st. This is (ω_n/2)^j So this is the jth or j+1st of the n/2nd roots.

Now using these values for Aeven and Aodd, we can get A at the nth roots of unity. Now we use our formula for A(x) in terms of Aeven and Aodd. So $A(\omega^j)$ = Aeven(ω^{2j}) + ω^j Aodd(ω^{2j}). And similarly, if we look at the point $\omega^{n/2+j}$. This is the opposite of ω^j . So using the same formula, this requires Aeven and Aodd at exactly the same points. The only difference is that we subtract these two terms instead of adding them together. This takes O(1) for each j. So, it takes O(n) total time.

Finally, we have A evaluated at these n points and that's our output that we return from the algorithm.

Now, notice this algorithm works for any omega which is an nth root of unity. We only require that Omega to the jth power is opposite Omega to the n/2 + j. That's true for any root of unity except when $\omega = 1$, because then this would be 1 and this would also be 1. So they're not opposites of each other. But, for any other root of unity, the jth power is opposite the n/2 + jth power.

(Slide 4) FFT: Concise
***



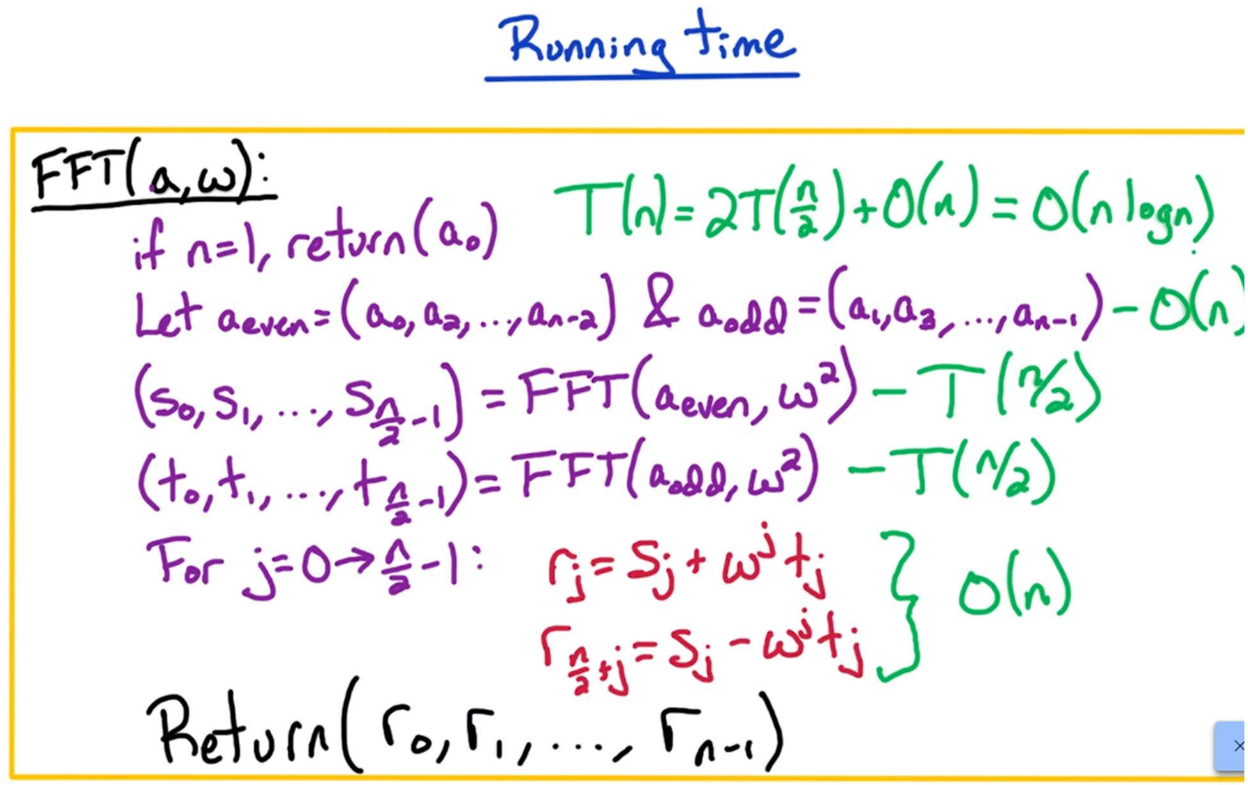
Part of the appeal of FFT is that the algorithm is quite concise. The algorithm is very simple. So let's re-express FFT in a more compact manner.

First off, we have the base case, n=1. This is a polynomial of degree 0. So in this case we simply return the constant term a0. Once again we define a_even, the vector, as the even terms in the vector a, and a_odd as the odd terms in the vector a. Then we recursively run the FFT(a_even, ω2). The output we get back we record as s0 through $s_{n/2-1}$.

Similarly, we will run FFT(a_odd, ω^2) and we record the output in t0 through $t_{n/2-1}$. Then, we combine the solutions for these subproblems to get the solution to the original problem. So rj (which is going to be A(x) at the point ω^j) = sj (which is Aeven at the point ω^{2j}) + ω^j tj. And similarly $r_{n/2+j}$ = sj - ω^j tj.

Finally we return these n numbers r0 through rn-1.

(Slide 5) FFT: Running Time
***



Now looking into running time of this algorithm, notice this step partitioning the vector a into a_even and a_odd that takes O(n) runtime.

FFT(a_even,ω^2) is a recursive call which is of size n/2 -> T(n/2)

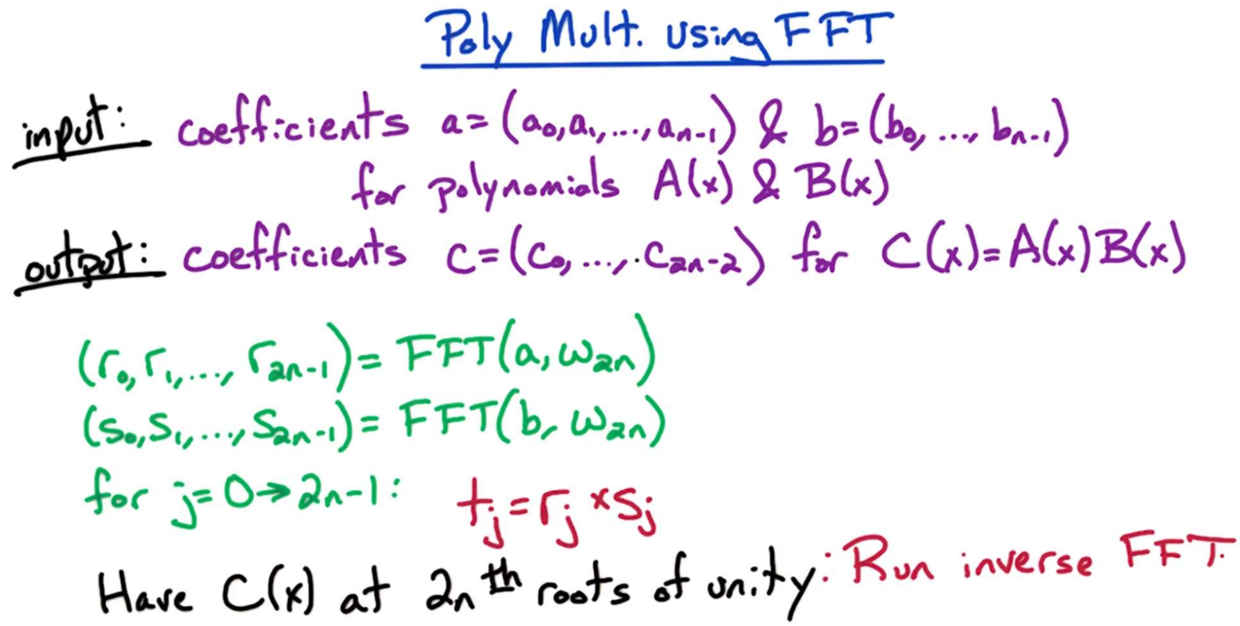
Similarly, for FFT(a_odd,ω^2), it's a recursive call of size n/2 -> T(n/2)

This computation of the r's takes O(1) for each pair. So it takes O(n) total runtime.

Therefore, this running time satisfies the recurrence T(n) = 2T(n/2) + O(n). And, of course, this solves to O(n log n).

That completes the FFT algorithm.

(Slide 6) Poly Mult. Using FFT
***



Now that we've completed the FFT algorithm, let's go back and look at our original motivation which was polynomial multiplication or equivalently, computing the convolution of a pair of vectors.

The input is a pair of vectors a and b of length n, corresponding to the coefficients for a pair of polynomials A(x) and B(x). The output is the vector c, which are the coefficients for the polynomial C(x), which is A(x) times B(x).

Equivalently, c is a convolution of A and B. In order to multiply these polynomials A(x) and B(x), we want to convert from the coefficients of A and B to the values of these polynomials A(x) and B(x).

How many points do we need these polynomials at? Well C is of degree 2n-2. So we want these polynomials … actually C(x) …at least 2n-1 points.

Notice that, in order to maintain that n is a power of 2, we'll evaluate A(x) and B(x) at 2n points. In order to do that, we'll run FFT. We will consider A(x) and B(x) as polynomials of degree 2n-1.

So, we'll just pad this vector with zeros. So, we run FFT with this vector a and ω_2nth root of unity. And this is going to give us a polynomial A(x) at the 2nth roots of unity. Similarly, we want to run FFT with this vector b and the 2nth root of unity and we get the polynomial B(x) at the 2nth roots of unity.

So now we have these polynomials A(x) and B(x) at the same 2n points. Now given A(x) and B(x) at the 2nth roots of unity, we can compute C(x) at the 2nth roots of unity.

We have a **for loop** on j = 0 -> 2n-1. So goes over all these 2n points and we multiply these pair of numbers. Even though these are complex numbers, t takes us O(1) runtime to compute the product of these pair of numbers. So it takes us O(1) to compute C(x) at the jth of the 2nth root of unity. So it takes us O(1) time to compute T(j) and therefore takes this O(n) runtime to compute C(x) at the 2nth roots of unity.

Now we have C(x) at the 2nth roots of unity. Now we have to go back from the value of this polynomial at these 2n points and figure out the coefficients. This is opposite of what we were doing before. Before, we were going from the coefficients to the values. Now we want to go from the values back to the coefficients. How are we going to do this? What we're going to do, is run an inverse FFT. And amazingly enough, the inverse FFT is almost the same as the original FFT algorithm.

(Slide 7) Linear Algebra View
***

Linear algebra view

For point x_j: $A(x_j) = a_0 + a_1 x_j + a_2 x_j^2 + \cdots + a_{n-1} x_j^{n-1}$

$$= (1, x_j, x_j^2, \ldots, x_j^{n-1}) \cdot (a_0, a_1, \ldots, a_{n-1})$$

For points $x_0, x_1, \ldots, x_{n-1}$:

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & & \vdots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Before we explore inverse FFT, it will be useful to explore the linear algebraic view of FFT. In this way, we can look at FFT as multiplication of matrices and vectors.

So consider a point xj. The polynomial A(xj) = a0 + a1 xj + a2 $(xj)^2$ + … + a_n-1 $(x\_j)^{n-1}$.

Notice that this quantity can be rewritten as the inner product of two vectors. The first vector are the powers of xj. And the second vector are the coefficients for this polynomial A(x).

Now FFT is evaluating this polynomial A(x) at n points. So let's look at this linear algebra view for the n points.

Now we're evaluating this polynomial A(x) at these n points. So we're computing A(x0), A(x1) and so on up to A(xn-1). The rows of this matrix would correspond to the powers of these points x0 through xn-1. We'll fill that in a second. But first what is this vector? This vector are the coefficients of the polynomial A(x).

Now, filling in the rows of this matrix … The first row are the powers of x0, and then the powers of x1 … and, finally, we have the powers of xn-1.

(Slide 8) Linear Algebra View of FFT
***

LA view of FFT

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & & \vdots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Let $x_j = \omega_n^j$

$$\begin{bmatrix} A(1) \\ A(\omega_n) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ & & \vdots & & \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

We just saw this linear algebra view of the evaluation of this polynomial of A(x) at these n points x0 through xn-1. Now let's look at it from the perspective of FFT.

For FFT, these n points correspond to the nth roots of unity. So let xj be the jth of the nth roots of unity. So it's $(\omega\_n)^j$. Now, let's rewrite these vectors and matrices.

Replacing these n points by the nth root of unity we have now A(1), A(ω_n), and so on up to $A((\omega\_n)^{n-1})$. So, these are the nth roots of unity.

This column vector is going to stay the same - it's still going to be the coefficients of this polynomial A(x).

Now let's look at the rows of this matrix:

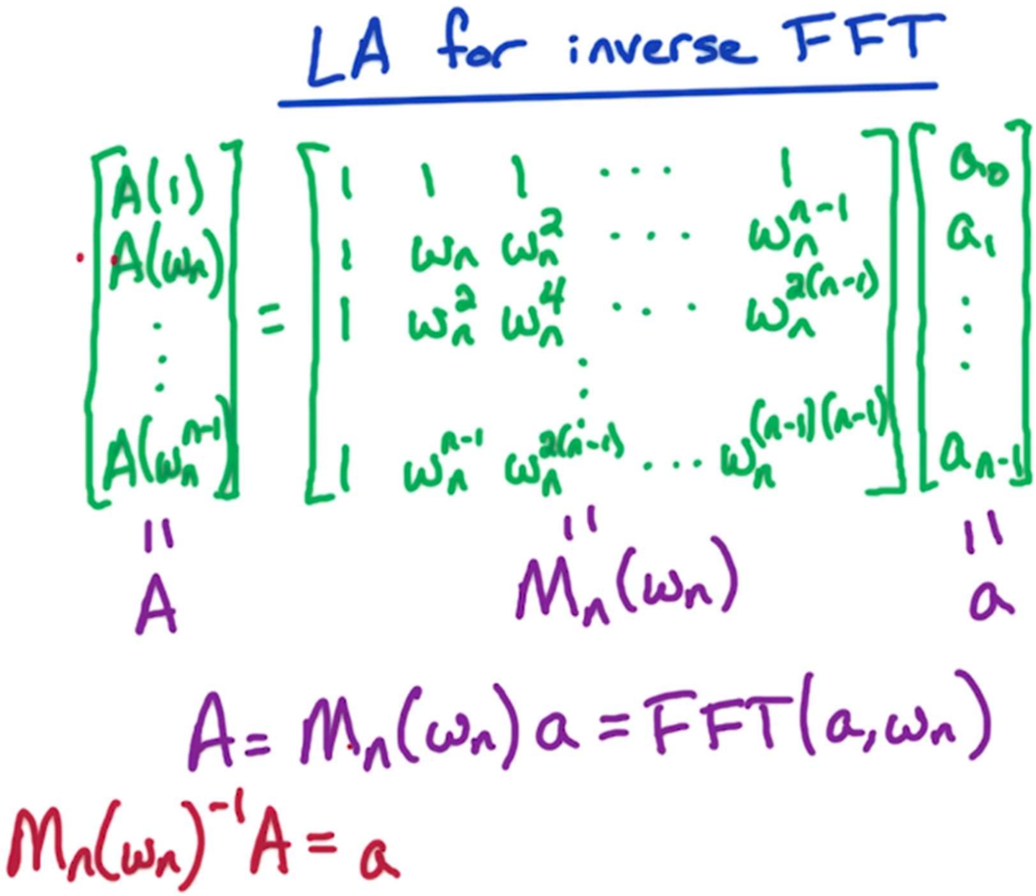
- Now, the first of the roots of unity is 1, so the first row are the powers of it - so it's just going to be one.

- The second row is going to be powers of ω_n. This, in fact, are just the nth roots of unity.
- The third row is going to be 1, $(\omega\_n)^2$, $(\omega\_n)^4$, … and so on up to $(\omega\_n)^{2(n-1)}$. It's just the powers of $(\omega\_n)^2$.
- The last row are going to be the powers of this last root of unity. So it's going to be 1, $(\omega\_n)^{(n-1)}$ and so on. The last term is $(\omega\_n)^{(n-1)(n-1)}$.

Now what's important thing to notice about this matrix? Well, first off, it's symmetric, the entry (i,j) is the same as the (j,i), so it's probably going to have some nice properties.

The other thing to notice is that it's just a function of ω_n. The entries of this matrix are just powers of ω_n.

(Slide 9) LA for inverse FFT
***



We have this linear algebraic view of FFT. Now let's simplify it a little bit.

This column vector is just the vector a. Let's denote this column vector by A. And let's denote this matrix by M, let's use subscript n to denote the size of M. And, as we observed, it's simply a function of ω_n. So given this variable, and this size, then we know it's Mn, and it contains powers of this variable ω_n. Therefore, this expression can be rewritten in the following manner: A = M_n(ω_n) a; and this product is exactly what FFT is computing.

When we run FFT(a, ω_n), it computes the product of this matrix M_n and this vector a and outputs the vector capital A, which is the value of this polynomial at the nth root of unity. A = M_n(ω_n) a = FFT(a, ω_n).

Now what do we want to do for inverse FFT? Now we want to take this value of this polynomial at these nth roots of unity and compute the coefficients.

Well, suppose this matrix has an inverse and we multiply both sides by this inverse. Well, then we have that the inverse of this matrix times this vector, A, equals this vector a. So FFT computes this product, this matrix M times this vector a. For inverse FFT, we want to compute the inverse of this matrix times this vector A.

How does this inverse of this matrix relate to the original matrix? Well, actually they're very similar to each other.

(Slide 10) Inverse FFT

***

Inverse FFT

$$A = M_n(\omega_n)\, a = FFT(a, \omega_n)$$

$$M_n(\omega_n)^{-1} A = a$$

Lemma: $M_n(\omega_n)^{-1} = \frac{1}{n} M_n(\omega_n^{-1}) = \frac{1}{n} M_n(\omega_n^{n-1})$

What is ω_n^{-1}? $\omega_n^{-1} = \omega_n^{n-1}$

$$\omega_n \times \omega_n^{-1} = 1$$

$$\omega_n \times \omega_n^{n-1} = \omega_n^{n} = \omega_n^{0} = 1$$

Once again, FFT when we run it with input a and ω_n (FFT(a, ω_n) - a has the coefficients for this polynomial A(x) and ω_n is the nth root of unity), it outputs A (which are the values of this polynomial at the nth roots of unity), and this corresponds to the product of this matrix M times a: A = Mn(ω_n) a = FFT(a, ω_n)

Now for the inverse FFT, we want to take these values of this polynomial and multiply by the inverse of them. And that will give us vector a, the coefficients.

What does the inverse of M look like? Well what we show is that the inverse of M = 1/n (just a scaling factor) x Mn(ω_n^{-1}). So we take the same matrix Mn, and instead of plugging in the nth root of unity, we plug in the inverse of the nth root of unity.

Now what exactly is the inverse of the nth root of unity? What is ω_n^{-1}? Well, this is the number when multiplied by ω_n equals one: ω_n ω_n^{-1} = 1. You multiply a number by its inverse you get one. So what is the inverse? It's ω_n^{n-1}. It's the last of the nth roots of unity.

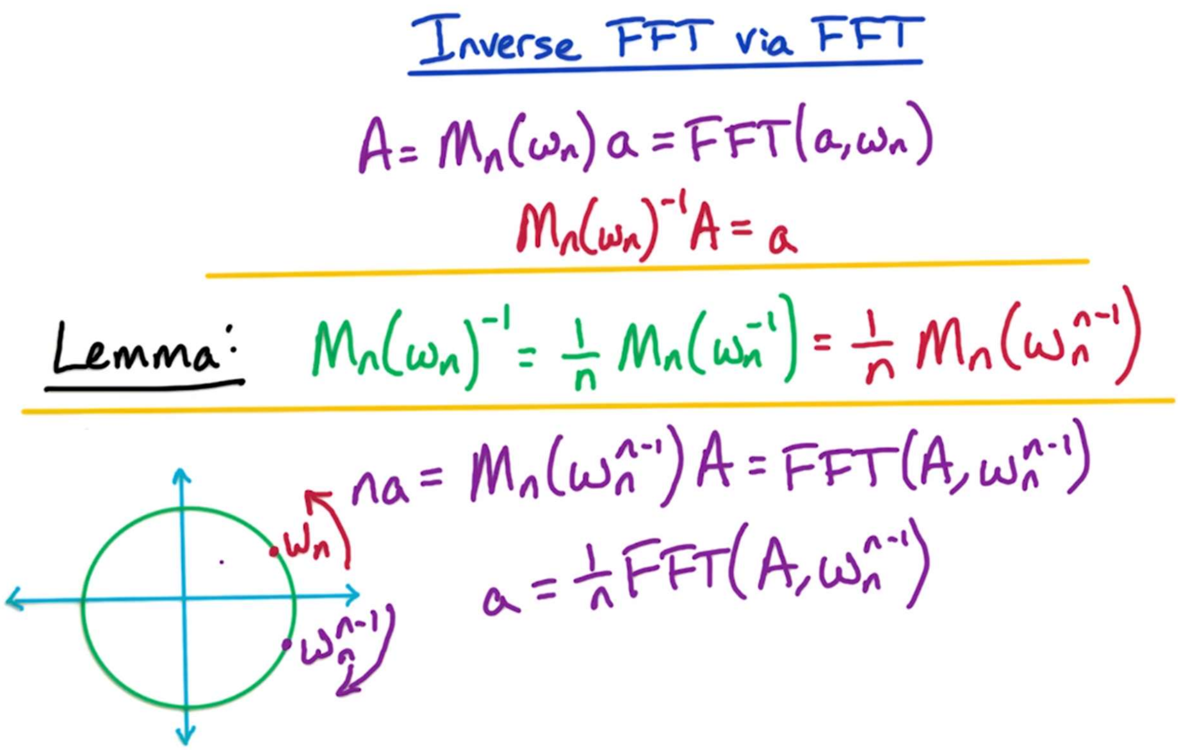
Notice that if you multiply ω_n x ω_n^{n-1}, what do you get? You get ω_n^{n} which is the same as ω_n^{0}, which is one. So, the inverse of ω_n is ω_n^{n-1}.

Now this is a basic fact, so you should make sure it is clear for you. If it's not intuitively clear, I would either convince yourself by plugging in these points in polar coordinates and also you can look at it geometrically.

So draw the picture of the complex plane and look at these points on the unit circle. Now we can plug this in and simplify. So, the inverse of this matrix M, with parameter ω_n, is 1/n times this matrix with ω_n^{-1}: $(Mn(\omega\_n))^{-1} = 1/n\ Mn(\omega\_n^{-1})$.

(Slide 11) Inverse FFT via FFT

***



Now let's recap what we have.

FFT is computing the product of this matrix capital M times a and it's outputting capital A, which is the value of this polynomial at these nth roots of unity. For inverse FFT, we want to do the reverse – so, we want to compute the product of M inverse times A and get back the coefficient vector a: $Mn(\omega\_n)^{-1} A = a$.

Now we claim the following lemma, which we'll prove momentarily:

$$Mn(\omega\_n)^{-1} = 1/n\ Mn(\omega\_n^{n-1}).$$

So we take the same matrix, and instead of parameterizing by ω_n, we parameterize it by $(\omega\_n)^{n-1}$ which is just a different root of unity. So let's take this expression multiply both sides by n and then substitute in n inverse with this quantity.

$$n\ Mn(\omega\_n)^{-1} = Mn(\omega\_n^{n-1}).$$
$$n\ a = Mn(\omega\_n^{n-1})\ A$$

So we have n times a, and then for M inverse, we replace it by this quantity $Mn(\omega\_n^{n-1})$ A.

Notice that this corresponds to an FFT computation. In particular, we want FFT with instead of using input little a, we use in put capital A and instead of using ω_n as the nth root unity, we use ω_n^{n-1} which is also an nth root unity.

Now it's quite intriguing what's happening here, we're taking the value of this polynomial A inverse at the nth root of unity and we're treating these values as coefficients for a new polynomial. Now we run FFT for this new polynomial and instead of using the nth root of unity ω_n, we're using this nth root of unity. It's still an nth root unity - so we can again run FFT. So we run FFT with these two inputs, we get back a vector which we scale by 1/n and this gives us the coefficients for our polynomial A(x). So we can go from the values at the nth root unity to the coefficients.

One more intriguing fact before we move on to the proof of this Lemma: Now, FFT, normally we run it with ω_n. It is this point right here. Then the n points we consider are ω_n to the powers which corresponds to the nth roots of unity going from one to ω_n and so on … around the unit circle in this manner. So we go counter-clockwise around the nth root unity.

Now what happens when we run FFT with ω_n^{n-1}? That's this point here. Now the only difference is we're going over the same points but we're going over them in a different order - Now we go over the nth root unity in clockwise order.

So inverse FFT is the same as FFT - we just go over the nth roots of unity in the opposite order … that's the amazing fact. And we can use the same algorithm as we detailed before because when we detail the FFT algorithm we allowed any nth root unity there.

Now I will proof the Lemma and that'll complete our polynomial multiplication algorithm and our convolution algorithm.

(1st Slide 12) Quiz: Inverses
***

Quiz: inverses

What is $(\omega_n^2)^{-1}$?

Before we dive into the proof of the Lemma, let's take a quick quiz on some basic properties of roots of unity. We saw just before about ω_n^{-1} … the inverse of ω_n. To make sure you understand that, let's look at ω_n^2. What's the inverse of this number? And don't simply write it with -2 in the exponent …write in some manner so that you have a positive exponent.

See DC5: FFT – Part 2: Quiz: Inverses

(2nd Slide 12) Quiz: Inverses (Answer)
***

Solution: inverses

What is $(\omega_n^2)^{-1}$? ω_n^{n-2}

$$\omega_n^2 \times \omega_n^{n-2} = \omega_n^n = 1$$

$$\begin{aligned}\left(1, \tfrac{2\pi}{n}\times 2\right)\times\left(1, \tfrac{2\pi}{n}(n-2)\right) &= \left(1, \tfrac{2\pi}{n}\times n\right)\\ &= (1, 2\pi)\\ &= 1\end{aligned}$$

This solution is ω_n^{n-2}. If I multiply ω_n^2 x ω_n^{n-2}, we get ω_n^n which is one. Similarly, in polar coordinates, this number in polar coordinates is (1,2π/n x 2). And this number (1, 2π/n x (n-2)). When we multiply these, we get 1 in the radius, and we add up the angles so we get 2π/n times n. This is the same as (1,2π) which is 1. So this verifies that the inverse of this number ω_n^2 is ω_n^{n-2}.

(1st Slide 13) Quiz: Sum of Roots

***

Quiz: Sum of roots

For even n,

$$1+\omega_n+\omega_n^2+\cdots+\omega_n^{n-1} = ?$$

Let's take another quiz on some basic properties of the roots of unity. Let's consider even n. And let's look at the sum of the nth roots of unity. So let's take $\omega\_n^0 + \omega\_n^1 + \omega\_n^2 + \ldots + \omega\_n^{n-1}$. What does this sum equal?

(2nd Slide 13) Quiz: Sum of Roots (Answer)

***

Solution: Sum of roots

For even n,

$$1+\omega_n+\omega_n^2+\cdots+\omega_n^{n-1} = 0$$

$$\cancel{\omega_n^0} \qquad \cancel{\omega_n^{n/2}} = -1$$

$$\omega_n^j = -\omega_n^{\frac{n}{2}+j}$$

The answer is zero. This sum is zero. The sum of the nth roots of unity is zero. Why is that true?

Well, that follows just from the +/- property which was the key to our Divide and Conquer algorithm. $1 = \omega\_n^0$. What is $\omega\_n^{n/2}$? Well, if you think of the complex plane and the roots of unity, we're going halfway around the unit circle. This is -1. They're opposites of each other. So, when we add them up they're going to cancel each other out.

Similarly, the jth of the nth roots of unity is opposite of the n/2 + jth. So, the first n/2 are opposite the last n/2. They are going to cancel each other out and we're left with 0. This is true for any even n.

(Slide 14) Proof of Claim
***

Proof of claim

Claim: For any n^{th} root of unity ω where $\omega \neq 1$:

$$1+\omega+\omega^2+\cdots+\omega^{n-1}=0$$

Proof: For any number z

$$\cancel{(z-1)}\left(1+z+z^2+\cdots+z^{n-1}\right)=0$$

0

$$=\left(\cancel{z}+\cancel{z^2}+\cdots+z^n\right)-\left(1+\cancel{z}+\cdots+\cancel{z^{n-1}}\right)$$

$$=z^n-1=0$$

Let $z=\omega$: $z^n=1$

Now in a proof of the lemma about the inverse of the matrix M, we're going to need the following claim which is a generalization of the quiz you just took. The claim says that if we take any omega which is nth root of unity … So ω^n is one … and we assume that ω != 1 but it is any other root of unity, and if we look at the sum of the powers of ω ... $1 + \omega + \omega^2 + \ldots + \omega^{n-1}$, then this sum equals zero. $1 + \omega + \omega^2 + \ldots + \omega^{n-1} = 0$.

Now we just saw that this is true when omega $\omega = \omega_n$ and n is even. We're going to need this more general claim so let's go ahead and prove it.

Now first off though. Notice why it's not true when ω=1, when ω=1 then this is 1 + 1 + 1 + - all the terms are 1. So this is going to be n. it certainly does not equal to zero.

Now let's forget about this claim for a second. For any number z, the following holds, look at $(z – 1) (1 + z + z^2 + \ldots + z^{n-1})$. So… powers of z. So looks a little bit similar to the claim.

Multiply this out, what do you get? Well first, multiple z times the series. So you get $z + z^2 +$ and so on up to z^n. And you've got -1 times this series. So you have minus the same quantity. Notice that most of the terms cancel each other out, z - z, z^2 - there's the z^2 there … $z^{n-1} - z^{n-1}$. What's left? The only term left here is the last term z^n and the only term left here is the first term -1.

Now let $z = \omega$ from the hypothesis of the claim. What do we know? We know that ω is a nth root of unity. So if we take ω or z^n, we get 1. So $z^{n-1} = 0$, which means either this quantity equals zero or this quantity equals zero or both. But we assumed that ω != 1. So that means z != 1. So z-1 != 0. So that can't be the case. So this quantity must be equal to zero. That's what we're trying to prove … when $z = \omega$, we were trying to prove that this sum equals zero.

So that completes the proof.

(Slide 15) Proof of Lemma
***

Proof of lemma

Need to show: $M_n(\omega_n)^{-1} = \frac{1}{n} M_n(\omega_n^{-1})$

$$\frac{1}{n}\underbrace{M_n(\omega_n^{-1})M_n(\omega_n)} = I = \begin{bmatrix} 1 & & 0 \\ & 1 & \\ 0 & & \ddots \\ & & & 1 \end{bmatrix}$$

For $M_n(\omega_n^{-1})M_n(\omega_n)$:
show entries (k,k) are n
& for $k \neq j$ (k,j) are 0.

Now let's go back to proving the lemma. We're trying to show that $Mn(\omega\_n)^{-1} = 1/n\ Mn(\omega\_n^{-1})$. So the inverse of this nth root of unity, which we saw before, is ω_n^{n-1}; but, it will be more convenient to treat it as ω_n^{-1}.

Now rewriting this, so multiply both sides by the matrix M. This becomes $1/n\ Mn(\omega\_n^{-1})\ Mn(\omega\_n) = I$, the identity matrix.

Now, what is the identity matrix? Well, this has 1s on the diagonal and 0s off the diagonal. So let's look at the product of these two matrices and we're going to look at the diagonal entries of this product and show that those are n and the off diagonals we're going to show are zeros.

So to recap, we have to show that the product of these two matrices, the diagonal entries, are n because 1/n times these products should be 1s, and the off diagonal entries (so for k!= j, the entry (k, j)) - these should be 0s. So we'll have two cases, the diagonal entries and the off diagonal entries.

(Slide 16) Diagonal Entries

***

Diagonal entries

For $M_n(\omega_n^{-1})M_n(\omega_n)$: show entry $(k,k) = n$

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ & & \vdots & & \\ 1 & \omega_n^{-(n-1)} & & \cdots & \omega_n^{-(n-1)^2} \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ & & \vdots & & \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix}$$

$$(1, \omega_n^{-k}, \omega_n^{-2k}, \ldots, \omega_n^{-(n-1)k}) \cdot (1, \omega_n^{k}, \omega_n^{2k}, \ldots, \omega_n^{(n-1)k})$$

$$= 1 + 1 + 1 + \cdots + 1 = n.$$

Let's first look at the proof for the diagonal entries. So let's look at the product of these two matrices: Mn(ω_n^{-1}) x Mn(ω_n). And we want to show that the diagonal entries (so, the entries (k, k)) = n.

First off, let's recall what this matrix Mn is. So Mn(ω_n) is this matrix - this is what we saw before when we analyzed FFT. Now this matrix Mn(ω_n^{-1}) is just this same matrix with ω_n replaced by ω_n^{-1}. So, we get this matrix here.

Now we're looking at the entry (k, k), so when you take the kth row and the kth column, the kth row of this matrix is the vector (1, ω_n^{-k}, ω_n-^{2k}, …, ω_n-$^{(n-1)k}$); the kth column of this matrix is this vector - (1, ω_n^{k}, ω_n^{2k}, …, ω_n$^{(n-1)k}$.

The entry (k,k) in the product matrix is the dot product of these two vectors:

- First term is one,
- and then we do ω_n^{-k} x ω_n^{k} … this is the same as ω_n^{0}, which is 1. So this term is 1.
- And similarly the third term is also 1. All the terms are 1. How many terms are there? There's n terms. So we get n.

So, we proved what we want for the diagonal entries.

(Slide 17) Off-Diagonal Entries
***

Off-diagonal entries

For $M_n(\omega_n^{-1})M_n(\omega_n)$: show entry $(k,j) = 0$
for $k \neq j$

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ & & \vdots & & \\ 1 & \omega_n^{-(n-1)} & & \cdots & \omega_n^{-(n-1)^2} \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ & & \vdots & & \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \qquad \omega = \omega_n^{j-k} \neq 1$$

$$(1, \omega_n^{-k}, \omega_n^{-2k}, \ldots, \omega_n^{-(n-1)k}) \cdot (1, \omega_n^{j}, \omega_n^{2j}, \ldots, \omega_n^{(n-1)j})$$
$$= 1 + \omega + \omega^2 + \cdots + \omega^{n-1} = 0.$$

Now let's look at the off-diagonal entries of this product matrix.

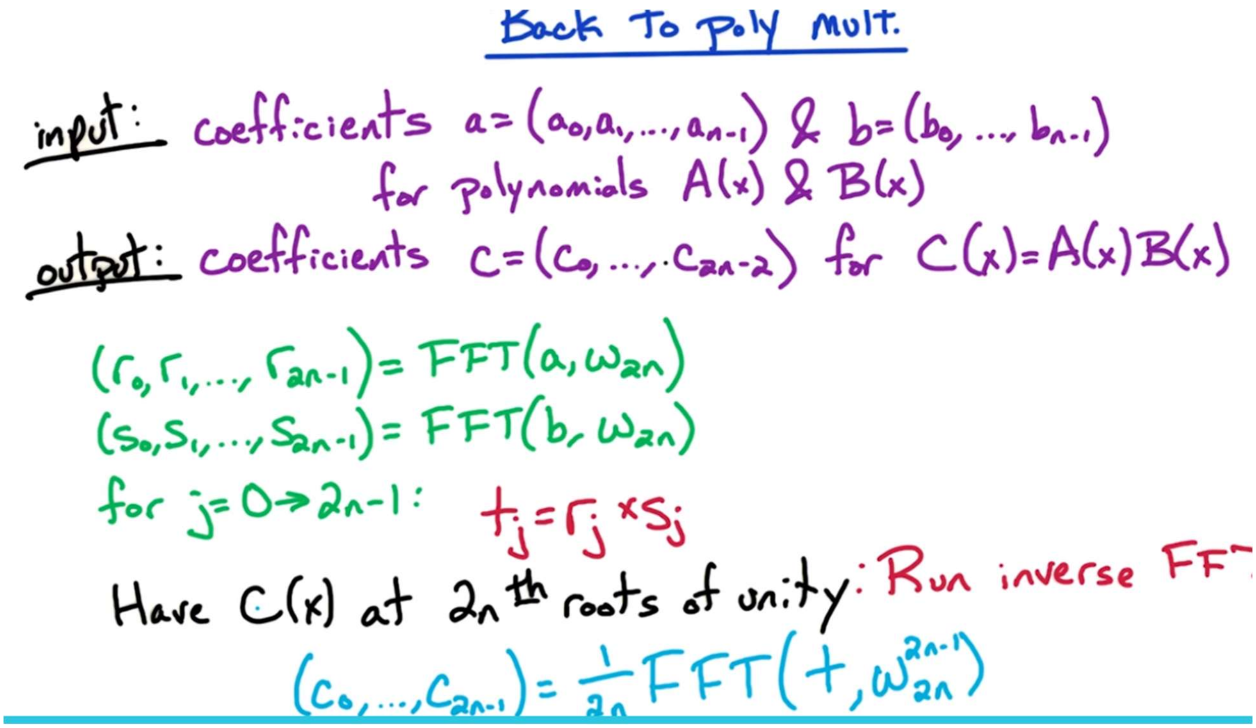
So we want to show that entries (k, j) = 0 equals zero when k!=j. Because if k=j, that's the diagonal entry. And we just showed that equals n. But, if they're not equal, we'll show it equals zero.

Here are the pair of matrices once again. We're again going to take row k and now we're going to take column j over here. The kth row of this matrix is the same as before and the jth column of this matrix is this vector. When we take the dot product of these two vectors we get the following one plus ω_n^{j-k}. And then we get powers of ω_n^{j-k}. Well, let's simplify this.

Let's do a change of variables to simplify it. Let's let $\omega = \omega\_n^{j-k}$.   Then the dot product of these vectors can be simplified as $1 + \omega + \omega^2 + \ldots + \omega^{n-1}$.   Now, what do we know about ω? Well, it's the nth root of unity raised to some power. So it's still an nth root of unity. And, we know that the exponent is not zero. Since it's not zero, then this thing is not 1. So ω is an nth root of unity and it's not one. So, we can apply our claim ... we're just doing powers of the nth root of unity ... we know for any nth root of unity, which is not 1, if we take powers of it, what do we get? - we get 0. Which proves this off-diagonal entries are 0 as we desire.

And that completes the proof of the lemma.

(Slide 18) Back to Poly Mult.
***



Let's go back to this earlier slide with our polynomial multiplication algorithm. Now we know how to do this last step. We know how to do inverse FFT which goes from the values of this polynomial C(x) to the coefficients of this polynomial. So let's add in the details of this last step which will complete our polynomial multiplication algorithm.

Now, in this last step, we're going to run FFT using these values t. t are the values of C(x) at the (2n)th roots of unity. Now, we treat these values t as the coefficients for a polynomial. So this vector t is the first parameter in our input.

The second parameter is a root of unity. What root of unity do we choose? We want to use the inverse of the 2nth root of unity, which is the last of the 2nth roots of unity, namely its $(\omega\_2n)^{2n-1}$.

Now when we run FFT on this input - FFT(t, ω_2n^{2n-1}) – we are going to get 2n points returned. Let's use this vector c as the return value, so c0 through c_2n-1. But recall, we have to scale this output, so we have to take the vector that's returned by FFT, scale it by 1/2n, and that gives us the coefficients of this polynomial C(x), and that's it.

That completes our polynomial multiplication algorithm and similarly, our convolution algorithm.