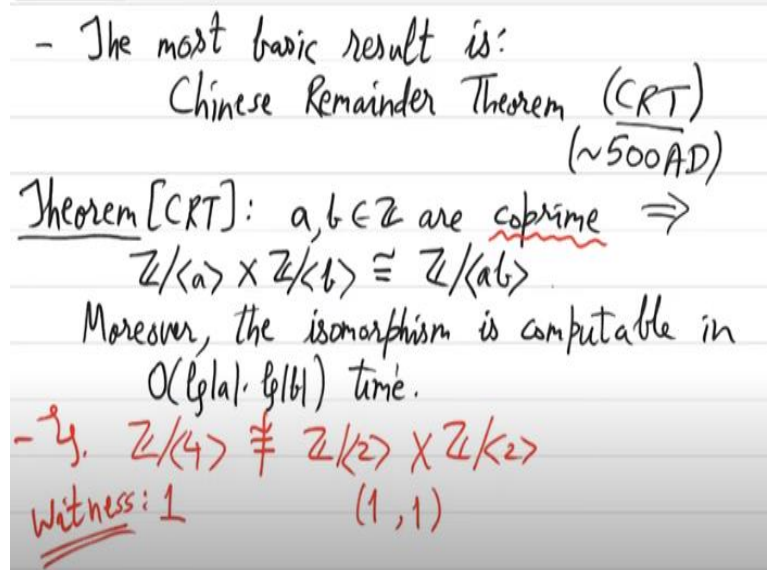


**Computational Number Theory and Algebra**  
**Prof. Nitin Saxena**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology-Kanpur**

**Lecture - 04**  
**Fast Polynomial Multiplication**

(Refer Slide Time: 00:15)



Okay, so last time we started Chinese Remaindering. The theorem said that if you take two integers that are coprime, then arithmetic modulo  $a$  and modulo  $b$  is the same as arithmetic modulo  $ab$ . And in terms of rings it meant that the factor ring  $\mathbb{Z} \bmod a$  cross the factor ring  $\mathbb{Z} \bmod b$ . This as a ring is isomorphic to the factor ring  $\mathbb{Z} \bmod ab$ . And the coprimality we saw was necessary condition.

So that was the mathematical part. There is actually an algorithm also which can achieve this. So this isomorphism map is very efficient. It can be computed in  $\log$  of the bit size of  $a$  times the bit size of  $b$ , right. So in the terms of input size bit size it is quadratic. Any questions?

(Refer Slide Time: 01:14)

$$\text{Pf: } \mathbb{Z}/\langle a \rangle \times \mathbb{Z}/\langle b \rangle \rightarrow \mathbb{Z}/\langle ab \rangle$$

$$\phi: (x_1, x_2) \mapsto x_1 u_1 b + x_2 u_2 a$$

$$\left. \begin{aligned} u_1 &:= b^{-1} \bmod a \\ u_2 &:= a^{-1} \bmod b \end{aligned} \right\}$$

$$\triangleright \phi(x_1, x_2) \equiv x_1 \bmod a$$

$$\equiv x_2 \bmod b$$

1)  $\phi$  is a homomorphism?  
 2)  $\phi$  is injective?  $[(x_1 u_1 b + x_2 u_2 a) \cdot (x'_1 u_1 b + x'_2 u_2 a)]$   

$$\Rightarrow \phi \text{ is isomorphism!} \quad \begin{aligned} &= ab \cdot x_1 x'_1 \cdot (u_1 b)^2 + x_2 x'_2 (u_2 a)^2 \\ &= \phi(x_1 x'_1, x_2 x'_2) \end{aligned}$$

□

So it was efficient because we actually gave a very explicit map. This map  $\phi$  was it just required computation of  $b$  inverse mod  $a$  and  $a$  inverse mod  $b$ .

(Refer Slide Time: 01:33)

CRT for polynomials: If  $f, g \in \mathbb{F}[x]$  are coprime polynomials, then
 
$$\mathbb{F}[x]/\langle f \rangle \times \mathbb{F}[x]/\langle g \rangle \cong \mathbb{F}[x]/\langle fg \rangle.$$

Moreover, the isomorphism is computable in  $O(\deg f \cdot \deg g)$  many  $\mathbb{F}$ -operations.

$\rightarrow$  CRT reduces rings to fields; in proofs & in algorithms.

So in the same spirit you can show Chinese remaindering theorem for polynomials. So if  $f, g$  over a field  $F$  in one variable  $x$  are, so these are univariate polynomials and they are coprime. Then like before, you have to look at the factor ring, right which will be the polynomial ring modulo  $f$  the ideal. So this is the first factor ring. The second factor ring is  $F[x] \bmod g$ .

So mod  $f$  arithmetic and mod  $g$  arithmetic this product is isomorphic to mod  $fg$ . Is that clear? So like numbers, polynomials have the same Chinese similar Chinese remaindering theorem, which is that as long as they are coprime mod  $fg$  arithmetic

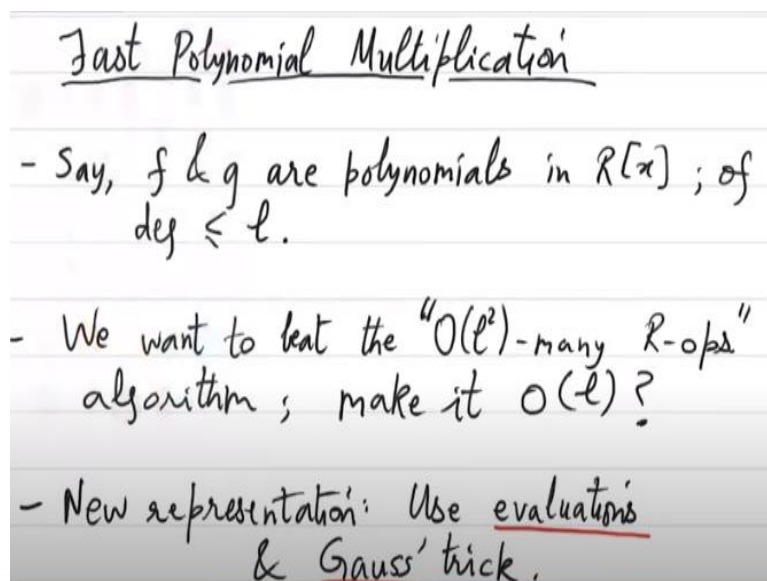
can be broken into simpler arithmetic mod  $f$  and mod  $g$ . The proof will be identical. You use the same map with  $f$  inverse mod  $g$  and  $g$  inverse mod  $f$ .

And the time complexity will be to compute that the isomorphism is computable in how many  $f$  operations? Right. So you have to recall how much time  $f$  inverse mod  $g$  computation requires. So you will get this in terms of degrees. So these many  $f$  operations which we are not going inside. So the ring operations in the  $f$  the field operations  $f$  we are counting as unit time.

On top of that it is degree  $f$  times degree  $g$  many such operations. And here also the coprimality is important. Okay if they are not coprime then again this statement is this isomorphism equation this will be false. That you can easily see. So if you take for example  $f$  to be  $x$  and  $g$  to be  $x$  then mod  $x$  arithmetic versus mod  $x$  square arithmetic, right. Those things are very different and unrelated.

Any questions? Right, so now whenever we invoke Chinese Remaindering Theorem, we will just invoke one of these two and the time complexity etc., will be implicit. Okay we have shown that these things are efficient. So the first major algorithm that we will see in this course is how to multiply polynomials.

(Refer Slide Time: 05:43)



Fast Polynomial Multiplication

- Say,  $f$  &  $g$  are polynomials in  $R[x]$ ; of  $\deg \leq l$ .
- We want to beat the " $O(l^2)$ -many  $R$ -ops" algorithm; make it  $O(l)$ ?
- New representation: Use evaluations & Gauss' trick.

So fast polynomial multiplication. Oh I wanted to make another remark before, why is Chinese Remaindering Theorem important in our study. So one thing is that always

the motivating case will be when you take  $f$  and  $g$  to be irreducible okay. So you want to do arithmetic modulo  $f$  times  $g$ , which is clearly a reducible modulus.

And you may not be able to study much mod  $f$   $g$ , but when you go mod  $f$  then since  $f$  is irreducible, this factor ring is a field. Okay so you reduce the question of rings to two fields. So the understanding over fields or in fields is far more than the understanding in rings. So this is why, in fact, that is the very reason CRT exists. That it gives you, it reduces rings to fields.

So this it does both in proofs and also in algorithm. So ideally you want to work over fields because there are more theorems over fields and over rings. More things are true over fields. Rings are harder to study. So this is one way to do that transfer okay. So back to this. Right so the problem here is clear. You are given two polynomials  $f$  and  $g$  in some polynomial ring.

So they are polynomials in one variable  $x$ .  $R$  is some base ring. And their degrees are let us say less than equal to  $l$ . So we are given  $f$  and  $g$  in the input and you want to compute  $f$  times  $g$  as fast as possible. Already, you know how to compute an  $l$  square operations. But now you want to do something closer to  $l$ . So before solving this problem, we should have studied actually how to multiply integers first, right.

That is a more basic question than multiplying polynomials. But the only reason why we are not doing that is because integer multiplication algorithms use polynomial multiplication. So somehow the machinery is better for polynomial multiplication. So we will first do this, then we will do the more basic thing which is integer multiplication. So actually integers are first written as polynomials.

And then they are multiplied in the fast algorithms. So for that strange reason, we start with polynomial multiplication first. So we want to beat the naive  $l$  square time algorithm. So we want to beat this algorithm order  $l$  square algorithm. So ideally you want to make it order  $l$ , right. That may be impossible. But at this point, we do not know whether it is impossible or possible, because the input size is order  $l$  to  $l$ .

So all we know is that we will need time at least  $l$ , but we do not know how much more. So the correct complexity is between order  $l$  and order  $l^2$ . So this fast algorithm will achieve something between and much closer to  $l$  than  $l^2$ . So I think some of you already know the idea how this is done. So how do you do this? FFT? Yeah, but what is why do you go to FFT?

Yes, so we change the representation of the polynomial and hence the algorithm also. So this basic algorithm is looking at the polynomials as sum of monomials and then it is multiplying one monomial with one more monomial one from  $f$  one from  $g$ . So that gives you  $l^2$ . Instead, you can look at a different representation, which is in terms of the values of  $f$ .

So you compute  $f$  at some points, so you get some evaluations and same with  $g$  and then you multiply these evaluations. So use the property that evaluation is a homomorphism. So when you multiply  $f(\alpha)$  with  $g(\alpha)$ , that is also the value of  $fg$  at  $\alpha$ , right. So you use that property. That evaluation is a homomorphism. So you can actually as well work with evaluations as far as multiplication is concerned.

So that is the trick that use evaluations and the thing with evaluation is that yeah, so now you can recall interpolation. So interpolation tells you that for a degree  $l$  polynomial, how many evaluations do you need to uniquely specify  $f$ ? Sorry  $l + 1$ . And if you assume  $f$  to be monic then only  $l$ , right. So polynomial is uniquely specified either by  $l$  coefficients or by  $l$  evaluations, okay.

So this is the second representation we use  $l$  evaluations. Then you can multiply them corresponding evaluations of  $f$  and  $g$ . And, but remember that in the output you have to output polynomial in the monomial representation, right. So you have to then do interpolation again, you have to basically reverse the interpolation to get the monomial representation.

So that is the sequence of steps, change the representation, multiply, and then again change the representation back. So in the middle of this Gauss' trick is used which will give the efficiency. So we will see these things. Yeah, so what do these keywords

mean? Let us now describe in more detail. So since somebody mentioned FFT, so this means that you will not take any arbitrary evaluation but some special evaluation.

So what are the points at which you evaluate? Sorry. Yeah, so you will use complex roots of unity. Yes. So let us move to that.

(Refer Slide Time: 14:48)

- Suppose  $R$  has a primitive  $l$ -th root of unity  $\omega$ .  $[l\text{-th root of unity} \in \mathbb{C}]$

Idea: \*1) Evaluate  $f$  &  $g$  at  $\{\omega^0, \omega, \omega^2, \dots, \omega^{l-1}\}$ .  
 2) Multiply  $f(\omega^i) \cdot g(\omega^i)$  in  $R$ . ( $0 \leq i < l$ ).  
 \*3) Interpolate to get  $h := f \cdot g(x)$ .

- Let  $f(x) =: \sum_{i=0}^{l-1} a_i x^i$ ,  $a_i$ 's in  $R$ . (linear operator)

- Discrete Fourier Transform  $DFT[\omega]: (a_0, \dots, a_{l-1}) \mapsto (f(\omega^0), \dots, f(\omega^{l-1}))$   
 where  $l := 2^n$ .

So first we make a major assumption that the base ring has those. That the base ring  $R$  already has complex roots of unity. So suppose  $R$  has a primitive  $l$ -th root of unity  $\omega$ , okay. So this may not be true because for example, if ring was ring of integers then it will have the second root of unity which is  $-1$ . But it does not have the third root of unity. So symbolically you are looking at this  $1$  raised to  $1$  over  $l$  or  $l$ -th root of unity.

So in general these are available in complex field but may not be available anywhere below complex field. So we are assuming that  $R$  has whatever we need, okay. So that is a major assumption. Usually this assumption is not worked out in other courses. But true to the name algebra in this course, we will work it out. So what do you do when  $R$  does not have  $l$ -th root of unity which is actually all the time.

In all practical applications,  $l$ -th root of unity will not be there in  $R$ , right. So there is a major construction that you have to do to get rid of this assumption. But let us first see the basic idea assuming this. So the idea is after this assumption so evaluate  $f$  and  $g$  at

these points. So points being  $\omega^0$  which is 1,  $\omega$ ,  $\omega^2$  and it goes up to  $\omega^{l-1}$ , right. So these are the  $l$  points.

So you get  $2l$  values in  $R$ . Then you multiply them. So  $\omega^i$  with  $\omega^i$ ,  $f$  and  $g$  respectively. This multiplication will happen in the ring  $R$ . This is happening for all  $i$ 's. So these are  $l$  multiplications you will do. These are  $l$   $R$ - operations. So we do not have to specify how this is done because this is in the base ring. So that we do not specify and then you have to do interpolation to get the product  $h$ , right.

So those are the three steps. So what is the complexity of this? If you just implement this, what is the complexity? Already in step one, right, because so it seems that we have not made any progress. We have just added a lot of keywords but no progress whatsoever, because just to evaluate  $f$  at  $\omega^i$ , in step one, it takes  $l$  steps because there are  $l$  monomials to be computed, and then added.

So  $l$  many  $R$ - operations and then you have to do this  $2l$  many times because  $f$  at  $l$  points  $g$  at  $l$  points, so this is  $l^2$ , right. So there is no progress it seems because the coefficients of  $f$  are arbitrary. So those arbitrary multiplications you have to do. **“Professor - student conversation starts”** Sir for interpolating  $h$  wouldn't we need  $2l$  coefficients? **“Professor - student conversation ends”**.

That is true, yeah. Yeah, those things are there, but say you assume that  $l$  is big enough. So we started with the assumption that  $f$  and  $g$  have degree at most  $l$ . So in practice you use  $l$  by 2. So  $f$  and  $g$  are degree  $l$  by 2 each. So each is of degree  $l$  and then we keep working with the same  $\omega$ . That just a readjustment. So the bigger problem is that this is not improving on  $l^2$ , right.

So this is why just changing the representation is not enough, you have to use something called the trick of Gauss, which is a clever halving step in a recursion, okay. So we will actually do a recursion. So it should be compared with the merge sort. So merge sort sorting algorithm requires for  $l$  elements  $l^2$ , I mean without clever recursion, it is  $l^2$ .

But with merge sort you halve it and then you get a recurrence formula, which gives you  $1 \log 1$ . So same thing we will do here, same effect. But the reason will be different. So yeah, let us basically we want to do step 1 and step 3 in a faster way in a recursive way. So those are the problems. 2 is trivial. 2 is not the problem the problem is 1 and 3. How to improve over  $1 \text{ square}$ ?

So say the polynomial is I said degree  $l$  but say this is  $l - 1$ . So there are  $l$  coefficients  $a_0$  to  $a_{l-1}$ . The  $a_i$ 's are in the ring  $R$ . So let us take this as the definition of  $a_i$ 's. So this evaluation step in the literature it is called Discrete Fourier Transform, okay. So the Discrete Fourier Transform is just a longer name for evaluation at roots of unity. So we will call it DFT  $\omega$ .

So DFT  $\omega$  is an operator, which has already taken an argument  $\omega$ . And then it will take these coefficients as an argument  $a_0$  to  $a_{l-1}$  and output the values of  $f$  at  $\omega$  to the  $i$ . So  $a_0$  dot, dot,  $a_{l-1}$ . It will map it to  $f(\omega_0)$ ,  $f(\omega_{l-1})$ . Okay so it takes this  $l$  coordinates and defines the polynomial  $f$  based on this and then  $f$  it evaluates at powers of  $\omega$ , which is also a parameter to the operator.

So this operator is classically called Discrete Fourier Transform. This is what we wanted to do in step 1 once for  $f$  once for  $g$ . Now we want to compute this in a very fast way, not in the trivial way, but faster way. So for that we need some assumptions. So we will assume that  $l$  is a power of 2. Why cannot we do that? Why can we assume  $l$  to be a power of 2?

Yes. So whatever is the degree upper bound for  $f$  and  $g$  how far do you think is a power of 2 from that degree bound? Sorry, twice yes. You just have to maximum you have to double that upper bound. So you double it and you use that power of 2 as the new upper bound. So that is why it is trivial. You can assume that  $l$  is a power of 2,  $2^n$ , okay. And then for this we will give a fast method to compute the operator.

**(Refer Slide Time: 25:06)**



Lemma 1:  $\text{DFT}[\omega^l] \circ \text{DFT}[\omega] = l \cdot \text{Id}_l$  [ $\omega^{-1} \cdot \text{DFT}[\omega^l]$ ]

Pf: •  $\text{DFT}[\omega]$  is the following matrix action:

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{l-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{l-1} & \dots & \omega^{(l-1)(l-1)} \end{bmatrix} \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{l-1} \end{pmatrix} = \begin{pmatrix} f(1) \\ f(\omega) \\ \vdots \\ f(\omega^{l-1}) \end{pmatrix}$$

$\Rightarrow$  The action of  $\text{DFT}[\omega^l] \circ \text{DFT}[\omega]$  is:

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega^l & \dots & \omega^{l(l-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{l(l-1)} & \dots & \omega^{(l-1)(l-1)} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{l-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{l-1} & \dots & \omega^{(l-1)(l-1)} \end{bmatrix} = \begin{bmatrix} l & 0 & \dots & 0 \\ 0 & l & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & l \end{bmatrix}$$

So well I think first I prove a nice property of the operator before moving to the algorithm. So let us first observe our relationship between so DFT omega is the operator you want to study. Actually in the algorithm idea already you would be interested in also finding the inverse of this operator, right because in step 2 you multiply the values. And then from the values you want to go back to the polynomial h.

So that is nothing but inverse of DFT omega. So what is the inverse of DFT omega? DFT, yeah nearly DFT omega inverse. So that is what we will show in this lemma that if you compose this with DFT omega inverse. So you apply the first operator DFT omega on a vector and then you apply DFT omega inverse. Omega inverse is simply 1 over omega. It is just this next primitive l-th root of unity.

Then the composition is actually just a scaling operator. So it is 1 times identity, okay. So this is just the identity transformation, linear transformation. So remember that DFT is a linear transformation. It is acting on a vector space. Is that clear from this? It should be clear from the definition in blue that this map that we have defined this is a linear operator.

So if you take this vector a bar, another vector a bar prime, right how is the action on a bar plus a bar prime? So a bar defines the polynomial f, a bar prime defines a polynomial f prime. When you take the sum you actually get f plus f prime, right. So

the image also can be decomposed as sum. So this is our linear operator with respect to what field. So that will depend on whether  $R$  is a field or not.

So but if  $R$  is a field then this is a linear operator over  $R$ , okay? Yeah, so identity is simply the, it sends a bar to a bar. And 1 times identity is that it is scaling a bar to 1 times a bar. Okay, that is the formula we want to show. This is an extremely fortunate formula because it tells you that inverse of DFT is again DFT. So it also has a simple similar form. So if you show that step 1 is fast, then step 3 is also fast.

Yeah, that is a good point. Why would  $I$  not be invertible,  $I$  is  $2$  raised to  $n$ . So what is the reason? The characteristic of  $R$  is  $2$ . So just rule that out. Say that the characteristic of  $R$  is either  $0$  or odd. And what to do with characteristic  $2$ ? That will be a simple exercise. We will use a slightly different, we will use as a simple trick to handle characteristic  $2$ . Yeah, but then we are digressing.

So those things are not important. Let us first prove this and go to the efficiency point. So DFT I write as a matrix product. So DFT is matrix action is you have first row is all  $1$ . Then you have  $1, \omega, \omega^2, \dots, \omega^{l-1}$  and so on times a  $0$  to a  $l-1$  okay. So once written this way, it is easy to see  $f_1$  is just some of the  $a_i$ 's. And  $f_\omega$  is basically a  $0$  to a  $l-1$  are the coefficients defining  $f$  right.

So when you look at the left hand side, you are actually evaluating  $f$  at  $1, \omega, \omega^2, \dots, \omega^{l-1}$ . So that is the matrix action. So what is the matrix action now of DFT  $\omega$  inverse on this? So you will multiply, on the left hand side you will multiply by the matrix where you replace  $\omega$  by  $\omega$  inverse. So this is the following product. So this is now  $\omega$  inverse.

So this matrix times the above matrix, right. And now you will with some practice, you will be able to see that the first row, all  $1$  row when you multiply it by the first column of the second matrix, right that will give you clearly  $1$ . But the inner product of the first row with the second column is the sum of  $1, \omega, \omega^2, \dots, \omega^{l-1}$ . And what is that? Why is that  $0$ ? Yeah, good. So then we have to prove it. So let us prove that first.

**(Refer Slide Time: 33:51)**

$$[ X^l - 1 = (X-1)(X-\omega_l) \cdots (X-\omega_l^{l-1})$$

$$\Rightarrow \text{coef}(X^{l-1})(\cdot) : 0 = 1 + \omega_l + \cdots + \omega_l^{l-1} \quad \checkmark$$

$$\sum_{i=0}^{l-1} \omega_l^{2i} = \sum_{i=0}^{l-1} \omega_{l/2}^i = 0 \quad \checkmark$$

$$\sum_{i=0}^{l-1} \omega_l^{3i} = \sum_{i=0}^{l-1} (\omega_l')^i = 0 \quad \checkmark$$

$$\vdots \text{ \& so on. } ]$$

Note: Assume  $l = 2^n \nmid \text{ord}(R)$ , i.e.  $2 \nmid \text{ch } R$   
 $[ \text{odd}(\text{ch } R) \Rightarrow l^{-1} \in R ]$

So  $x$  to the  $l-1$  this polynomial has factors or roots, right. This is the factorization over complex of  $x$  to the  $l-1$ . What is the coefficient of  $x$  to the  $l-1$  in the above? So in the left hand side it is 0 and in the RHS it is just with sign 1 plus omega dot dot okay. So this means that the sum of omega to the  $i$  vanishes. That is the first thing.

But in the matrix product you are also getting when you multiply first row with the third column, they take the inner product. Then you will be looking at sum of 1, omega square, omega 4, dot, dot, omega to the  $2l-1$ . So what is that? So let us shorten this. So what is this equal to, omega to the  $2i$ ? What is omega square? Is it a root of unity? But it is then  $l$  by 2-th, right.

So you can replace it by omega prime or maybe we can use we can parameterize this. So this was omega 1. This is now, this can be written as  $l$  by 2, okay. This is the  $l$  by 2th because  $l$  is 2 raised to  $n$ . So this is 2 raised to  $n-1$ . Omega square is 2 raised to  $n-1$  th primitive root of unity in complex. So when you look at this sum this will give you all these  $l$  by 2th roots of unity possibly with I mean with repetition, you get 2 basically.

Each of these are doubled. So just like you did argument before, you can do now, you can look at  $x$  to the  $l$  by 2 -1 and this also will be 0 okay. Then next will be 3 i. So what is that? What is omega 1 cube? What is the order of this? Yes omega 1 cube is just like omega 1. So omega 1 powers will give you all the  $l$ -th roots of unity. And omega 1 cube also has the same property.

So just the first equation can be used. So you can think of it as it is  $\omega^l$  some other  $\omega^l$  prime, some other  $l$ -th root of unity and then you are raising it to  $i$ , which by the above argument, top argument is 0. So it is just this argument appropriately changed for the power okay. So for odd the argument is the top argument. For even, like for 2 it was you bring it down to  $l$  by 2. For 4 it will be bring it down to  $l$  by 4 and so on.

Okay, so this you can complete. Okay, so once you complete that at least the first row has been handled. Right. So first row you saw is 1 followed by zeros. What about the second row? Second row is 1,  $\omega$  inverse,  $\omega$ , inverse  $l-1$ . Sum it up. So that sum will be and  $-1$  you can think of it as  $l-1$ , right because it is on top of  $\omega$ .  $\omega$  is order  $l$ . So  $-1$  and  $l-1$  are the same.

So an  $l-1$  is odd. So by the green calculation, this will still be 0. So this will be 0. In fact, with all the columns you will get 0 except the second column. Second column will cancel  $-1$  with 1. So there you will get 1. And this pattern then continues and gives you the identity matrix. Is that clear? So DFT  $\omega$ ,  $\omega$  inverse when you compose them, then you get scaling effect. Scaling is exactly by  $l$ , okay.

So one note is assume  $l$  to be not a 0 divisor of  $R$  that is characteristic of  $R$  is not 2, okay. So 2 raised to  $n$  will be 0 divisor if and only if it is actually 0. In all other cases 2 raised to  $n$  will be invertible. So for now you can assume characteristic of  $R$  to be not 2. When it is 2 we will see a different argument. It is a simple correction. So the reason why you need  $l$  to be invertible so characteristic of  $R$  not equal to 2 implies that  $l$  inverse exists.

So you want  $l$  inverse to exist, so that this formula lemma 1 you can say that DFT  $\omega$  inverse divided by  $l$  is the inverse. You wanted that to make sense. So we want to look at  $l$  inverse times DFT  $\omega$  inverse. Yes, what was the question? **“Professor - student conversation starts”** Sir, characteristic of  $\mathbb{Z}_6$  so that is only 6 but 2 is still a zero divisor, so it will not be even. **“Professor - student conversation ends”**.

Oh I guess what I mean here is that it is a field. Yeah, but there are now too many implicit assumptions. Okay, let me change this. So 2 raised to n is not a zero divisor if and only if the characteristic is odd. And when the characteristic is odd then 1 inverse will exist. Let us live with this. Yes 2 does not divide or characteristic is 0. Right. So now we can see the Gauss' trick how do you compute DFT faster than 1 square.

(Refer Slide Time: 43:19)

- Naively,  $\text{DFT}[\omega]$  takes  $O(l^2)$ -time.  
But, Gauss' had a better idea:

Lemma 2:  $\text{DFT}[\omega]$  can be computed in  $O(l \cdot \log l)$   $R$ -operations.

Pf: •  $f(x) =: f_0(x^2) + x \cdot f_1(x^2)$  & use divide-conquer paradigm:

1) Compute  $\text{DFT}[\omega^2]$ :  $f_0 \mapsto (e'_0, \dots, e'_{l/2-1})$   
& " :  $f_1 \mapsto (e''_0, \dots, e''_{l/2-1})$ .

2) Compute 
$$\left. \begin{aligned} e_i &:= e'_i + \omega^i \cdot e''_i \\ e_{i+l/2} &:= e'_i - \omega^i \cdot e''_i \end{aligned} \right\} 0 \leq i < l/2$$

But Gauss had a better idea. So that idea is by as I said recursion, but how do you recurse? Surely you have seen it. But generally you do not understand it. Do you remember it? So this  $f(x)$  monomials you divide into kind of even monomials odd monomials better than faster than 1 square, okay. So you will be able to reduce actually both the roots of unity and the degree of the polynomial.

So that is how that is what is meant by recursion. And the algorithm you have seen. So what you get after that is this statement that DFT omega can be computed in order  $l \log l$   $R$ -operations, okay. So instead of 1 square you will get a  $l \log l$ . So let us just implement what we discussed now. So you will write  $f(x)$  as even monomials. So  $1, x^2, x^4, x^6$  monomials that part and the remaining part that is the odd part.

So even part is a function of  $x^2$  and odd part is a function of  $x$  if you remove  $x$ , then it is also a function of  $x^2$ . So it is this. So uniquely  $f(x)$  will give you  $f_0$  and  $f_1$ . And then you use the divide and conquer trick paradigm. So remember that you wanted to evaluate  $f$  at  $\omega^i$ . Instead you will evaluate  $f_0$  at  $\omega^i$  to

the  $2^i$  or equivalently you can think of as evaluating  $f_0$  at  $\omega^2$  as  $1$  by  $2^i$ th primitive root of unity, right.

So the points have been halved and what is the degree of  $f_0$ ? That is also  $1$  by  $2$ . So the degree of  $f_0$  has been halved and the points have been halved. So truly the DFT  $\omega$  has been halved, okay. The problem has been halved. So this is the first half and  $f_1$  is the second half. And then you can combine it by just taking a linear combination of  $f_0, f_1$ . So that divide and conquer is now straightforward.

So first you compute DFT  $\omega^2$  on  $f_0$ . So let us say this gives you, by  $f_0$  I mean look at the coefficients of  $f_0$  that vector. So on that vector you compute apply DFT  $\omega^2$ . And suppose you get  $e_0$  prime to  $e_{1/2-1}$ . So you get now  $1$  by  $2$  ring elements. And the same with the  $f_1$ . So  $e_0$  prime, prime okay. And now in step 2, you want to combine these two.

That will give you DFT  $\omega$  on  $f$ . So this was an important thing, which you should note that this is DFT  $\omega^2$ , okay. So the primitive root of unity order has been halved here. And also the dimension on the vector space has been halved to this. So here you just merge, merge the two solutions. So how will you merge? So you want to say  $e_i$  prime plus is this all for all  $i$ ?

Yeah, so this is actually only for half the  $i$ 's. So up to  $i$  less than  $1$  by  $2$  this is okay. And for the other  $i$ 's so the ones that come after  $1$  by  $2$  and beyond. So these ones will give you minus sign, okay. Any questions? So you get all from all the  $1$  coordinates. That is  $e_0$  to  $e_{1-1}$ . So you have this divide step. Now you have the merge step and that is it. So you will just output this.

**(Refer Slide Time: 51:03)**

3) Output  $(e_0, e_1, \dots, e_{l-1})$ .

- Let it take  $T(l)$  R-ops. for  $\text{DFT}[\omega_l]$ .  
Then, we've the recurrence:  

$$T(l) \leq 2 \cdot T(l/2) + O(l)$$

$$\Rightarrow T(l) = O(l \cdot \lg l). \quad \square$$

Theorem:  $h = f \cdot g$  computable in  $O(l \cdot \lg l)$  R-ops.

Pf:  $f \xrightarrow{\text{DFT}[\omega_l]} (f(1), \dots, f(\omega_l^{l-1})) \xrightarrow{\text{Mult}} (h(1), \dots, h(\omega_l^{l-1}))$   
 $g \xrightarrow{\text{DFT}[\omega_l]} (g(1), \dots, g(\omega_l^{l-1})) \xrightarrow{\text{Mult}} (h(1), \dots, h(\omega_l^{l-1}))$   
 $\downarrow \omega_l^{-1} \cdot \text{DFT}[\omega_l^{-1}]$   
 $h(x) \quad \square$

So the correctness of this should be fine, right. The way this is written, it is already a proof that we just have to analyze the time complexity, right how many R-operations does this take? So let it take  $T(l)$  time,  $T(l)$  R-operations for DFT  $\omega_l$ . Then we have the following recurrence. So the recurrence  $T(l)$  is less than equal to what is the relationship with  $T(l)$  by 2?

Yeah, so it is twice because there were two instances  $f$ ,  $g$ . Those many R-operations. And the merge was around  $l$ . So that many R-operations. So which if you solve like the merge sort you will get exactly a  $\log l$  okay. So the time complexity is  $l \log l$ , many R-operations. This is for our idea number 1, where you do the you wanted to evaluate  $f$  and  $g$ . So all the evaluations are basically computing that is the same as applying DFT twice.

So that can be implemented in  $l \log l$ , right. And then you had idea number 2 and 3. So in 2 you wanted to multiply the values and then in 3 you wanted to interpolate back. So let us see what happens there. So actually at this point, it should already be clear that all those steps will take similar time. So ultimately you would have multiplied polynomials  $f$  times  $g$  is computable in  $l \log l$  R-operations.

So the proof is just thus the sequence. So you take  $f$  and you do DFT. You do DFT so you will get  $f(1), \omega_l, \dots, \omega_l^{l-1}$ . And on  $g$  you will get so this one is DFT  $\omega_l$ . So this step can be done in  $l \log l$  by the previous lemma. Once you have these  $2l$  values, you

will just multiply them coordinate wise. So that takes  $R$ - operations. So multiplication gives you the values of  $h$ . So you get  $h$  at  $1, \omega, \omega^2, \dots, \omega^{l-1}$ .

So that is the multiplication in the ring. So that time will be how much in this step? This is only order  $l$  right. This is the cheap step. And now you want to get back  $h$  in monomial representation which is inverse DFT which is again a DFT right. So on this you apply DFT  $\omega$  inverse and that will give you  $h$  polynomial  $h(x)$ . So you apply DFT  $\omega$  inverse and then you scale down by  $l$ , that is a trivial step.

So overall this will take  $l \log l$  right all the steps together. So all the components now are in terms of this single operator called DFT right. So this is the fundamental operator. Any questions? Okay, so now we have to unravel the big assumptions we made, right? Because they are nearly always false.

In practice, if you take  $R$  to be complex then you have  $\omega$  present, but then saying that your algorithm takes  $l \log l$  complex operations is not of much help because computers cannot do complex arithmetic, right. Complex numbers themselves have to be represented by approximated by integers, ratio of integers. And then integer arithmetic has to be done. So ultimately everything has to be in bits.

So saying that you have a fast algorithm in terms of complex operations is actually practically useless. So you have to get an algorithm, given algorithm that is  $l \log l$  or similar complexity in bit operations or at least in  $\mathbb{Z}$  operations. So this is still far from that. So we have to actually create  $\omega$  when it is not there, right. It needs some creation.

**(Refer Slide Time: 58:32)**



---

- What if  $R$  doesn't have  $\omega$ ?  
We'll create  $\omega$  out of thin air!

- Consider  $E := R[y] / \langle y^{1/2} + 1 \rangle$  &  
 $\omega := y$  in  $E$  is irreducible over  $\mathbb{Q}$

---

34/34

So what if  $R$  does not have  $\omega$ . So what we will do here is that we will create  $\omega$  out of thin air. So what is the meaning of this? How do you create something in algebra out of thin air that does not exist? Yeah, so you take a ring extension. So this  $l$ -th root of unity is not there in  $R$ . So you take an extension such that this gives you a virtual  $\omega$ , not a complex  $\omega$ , but just a virtual representation of similar properties.

So consider an extension  $E$  of  $R$ . It is a ring extension. So  $R[y]$  modulo. So what should you quotient out? So we have gone to the polynomial ring of  $R[y]$ . You have to quotient out by something so that this  $\omega$  gets created. So first choice could have been  $y$  to the  $l - 1$ . Is that correct? It is not the correct choice because it does not mean so all these things here are vague.

But in a way, modulo  $y$  to the  $l - 1$ ,  $y$  is not does not have order it does not behave in a primitive way. So for it to be primitive, you want to say that  $y$  to the  $l$  by  $2$  should be  $-1$  okay. So you will mod out by that.  $y$  to the  $l$  by  $2 + 1$  okay. So if you look at the order of  $y$ , multiplicative order of  $y$  element  $y$  in this ring extension  $E$ ,  $y$  to the  $l$  by  $2$  is  $-1$ ;  $l$  is a power of  $2$ . So hence the order of  $y$  is actually  $l$ .

I think you will also be fine with  $y$  to the  $l - 1$ . But at this point I want to avoid that. Let us continue with this. But you could take this or you could take  $y$  to the  $l - 1$ . So that will provide you with this virtualized version of  $\omega$  and you can talk about

now  $y$  to the 0,  $y$ ,  $y$  square dot, dot  $y$  to the  $l - 1$ . So you can think of that set as the set of points. And you try to do everything like you did before.

Just syntactically replace  $\omega$  by  $y$ , okay. But then you have to, the only issue will be in the time complexity. So you because the time complexity you want to say how many operations in  $R$ . So you have to carefully analyze because you are now working with the polynomials over  $R$ . So you have to do polynomial arithmetic there above  $R$  just to do computations in  $E$ .

Okay, so we have to be careful with the time complexity analysis. So we consider  $E$  to be this and we take  $\omega$  to be now this  $y$  okay. That is the solution to work with  $\omega$  when  $\omega$  is not there. Oh I remember. So this one property of this polynomial so is  $y$  to the  $l - 1$  irreducible? It is not. Is this polynomial irreducible over  $Q$ ? Yeah, so this is irreducible, that is one good thing.

So I want to work with this because it is it has this irreducibility property over rationals. Generally  $R$  will be  $Q$  or  $Z$ . So in that setting this will be an irreducible polynomial. This also I leave as an exercise. So we will build on this in the next class. Yeah, but  $R$  is our ring. So there we have to, probably that also you can show as long as characteristic of  $R$  is not I mean is odd. It should be true.

Yeah, those are side observations, you can prove them as exercise. So it is very important in what we are going to do in the next class.