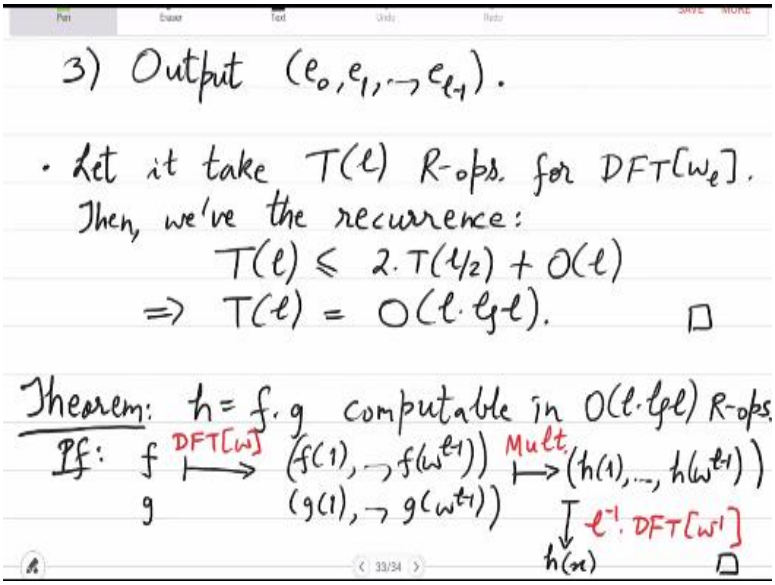
Computational Number Theory and Algebra
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture - 05
Fast Polynomial Multiplication (contd.)

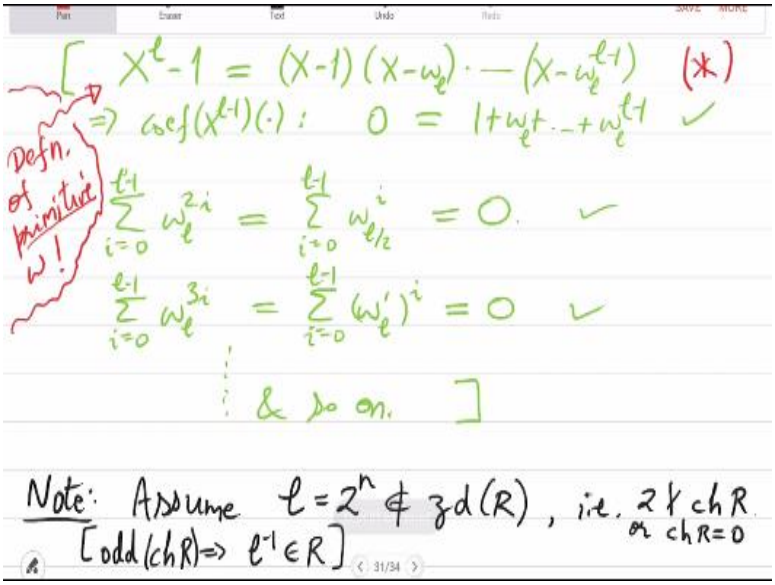
(Refer Slide Time: 00:16)



Okay, so last time we finished this theorem that you can multiply two polynomials each of degree l, in fact each of the degree l by 2. So the product will have degree at most l. So this multiplication can be done and the coefficients are in a ring R. So for this multiplication you will only need l times log l many R-operations, okay R additions and R multiplications.

So this was ultimately broken down into the single DFT description. So DFT omega is what we described, which is nothing but just evaluating f at l-th roots of unity, which you have to assume is present in R, okay. So you do that, then you multiply the values of f and g and then you apply DFT inverse. Any questions? So one point here was to do with the definition of primitive root. What is a primitive l-th root of unity.

(Refer Slide Time: 01:23)

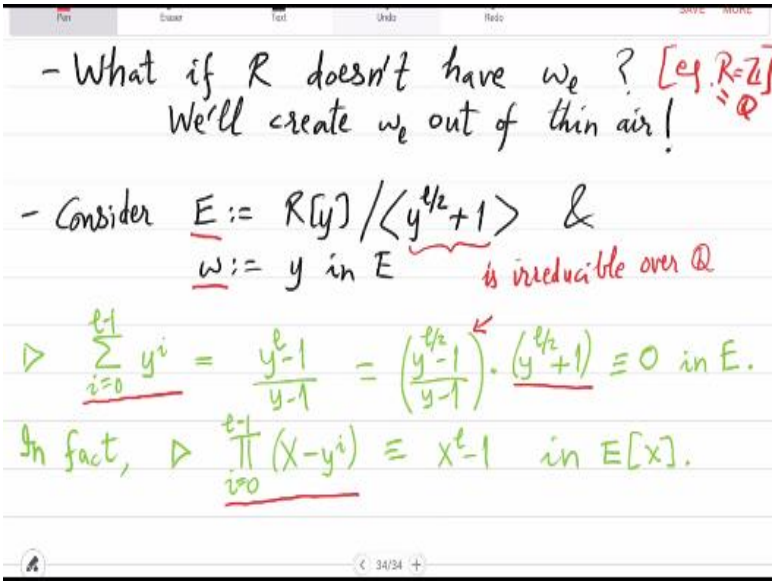


So remember that what we need from omega are these equations that sigma omega to the i should vanish and so on. So I would say that if your ring is not complex field, in that case you should take the definition of primitive omega is this. Omega is such that this identity holds okay. So this identity you can take as a definition. This identity star is what you want to call omega a primitive l-th root of unity.

Because this is really what you need, okay. Anything else we do not care. So this is mainly motivated from the complex fields. So their x to the l - 1 indeed factorizes this way uniquely. In other rings, it may not be so. There may be many factorizations and other problems and issues. So those definitions do not easily generalize. So we will just call star as the definition.

Right and there was this issue about characteristic also that for this to work, we have assumed characteristic zero or odd. That we will correct in the end of the class.

(Refer Slide Time: 02:48)



So let us now move to the general case which is a ring R which does not have omega, which will usually be the case. You can take R to be Z for example. So in this case except l = 2, you do not have any other primitive root of unity. So you keep in mind R = Z for the subsequent discussion, because that really is your motivating case or you can also take R to be Q, which has the same problems, okay.

Q also does not have any omega. And these are both of practical consequence. All your practical questions are in Z or Q. So in that case we will work with an extension called E which is where you have basically introduced this virtual y element. Formally you have modded out by the ideal y to the l by 2 + 1. So the motivation for this is that this is an irreducible polynomial if you are working over Q.

So this is a field extension. A more concrete reason is this calculation in green. So if you look at sigma y to the i, what is that? So sigma y to the i in E, you can do the geometric sum. So you will get y to the l - 1 over y - 1. Now the question is, is y - 1 inverse defined? So is the first equality even allowed in the ring E? Can y - 1 be a zero divisor? Sorry. No, I am talking about general R now.

We want to give a general algorithm. So it may have some characteristic, characteristic 3 or whatever. It may be a finite field. Finite field are also practical examples. So it could be a finite field, but it will not matter. So as long as it is characteristic odd or 0, y - 1 will always be coprime to the modulus, okay. So the coprimality means that y - 1 will be invertible. So that gives you the first equality.

And once you have the first equality then you can simplify it. There is no argument, you it is an exercise. So show that the polynomials y - 1 and y to the l by 2 + 1 they are coprime. It is a one line argument. I mean you if they were they are not coprime then y to the l by 2 + 1 has to vanish at 1. But that value is 2. Characteristic is odd or zero. So it cannot vanish. So they are coprime.

So but anyways you formalize it and then you see that by Euclid GCD, y - 1 is invertible. So the first thing is fine. Then you can simplify it. So you can factorize y to the l – 1 like this. You express it as, so it is you see it as sum of a difference of squares, right? So you get the difference and times the sum. And note that y - 1 divides the first expression. So we can separate.

So you can see that this is actually a polynomial, right. So you have a polynomial factor times another factor, which is in E it is zero because this y to the l by 2 + 1 is 0. So that is the beauty of this modulus. That sigma y to the i actually vanishes, okay. That is one of the key identities you need for polynomial multiplication. This would this be true if you went mod y to the l – 1.

No right because then the first equality itself will be a problem. So this is the major reason why we have picked the modulus like this. And in fact you can even work this out. What is product of x - y to the i. I am not sure whether I need all the power of this but this also you can see, this product will be x to the l - 1. So I would not discuss the proof of this but you can do this as an exercise if you want.

So in that sense, y is truly a primitive root of unity, l-th primitive root of unity in the ring E. Okay, so let us now proceed with, what is the question? **“Professor - student conversation starts”** From the roots of y to the l by 2 + 1 can we generate all the roots of y to the l – 1? **“Professor - student conversation ends”.** x to the you mean x to the l – 1? Use a different variable.

So all the roots of x to the l – 1? Yeah so those questions are not what you would expect when the ring R is complicated. Because in a ring R there may be many more than l roots. The degree of x to the l - 1 is only l, but over non field rings you do not

know how many roots a polynomial can have. So this intuitive feeling that you have that every l degree polynomial has at most l roots is completely false.

So I do not want to go into those issues. Let us stay close to a field. **"Professor - student conversation starts"** Why we are trying to, why we are taking modulus with this particular equality y to the l by 2? **"Professor - student conversation ends".** Yeah, so the first equality is the major reason sigma y to the i is zero. For other modulus it is not true.

And another reason is or another motivation is that if R is your usual field Q, then this is an irreducible polynomial. So you go to a field extension. Those are the two reasons. Okay, so. **"Professor - student conversation starts"** Sir, what will be the problem if we would have taken modulus y to the l -1? **"Professor - student conversation ends".** The first equality fields, you cannot sum up sigma y to the i.

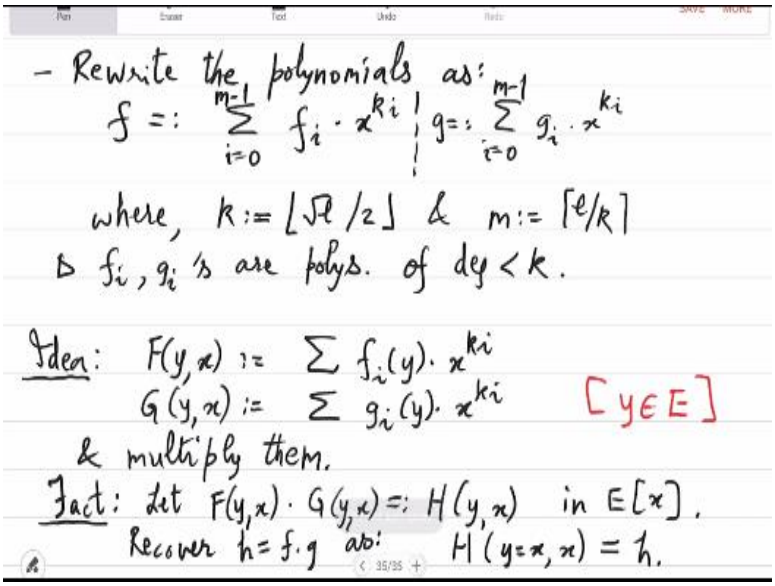
In particular, it is nonzero. And you wanted zero in your calculations that you did in this matrix product. This green part you will not get, right. So the idea is that in the case when R does not have omega, we introduce this y and we then work in the extension. But when you work in the extension, what will happen is this polynomial evaluation f (y) this so when you have f (y) times g (y) operation to be done, that is no more a R- operation.

It is an E- operation, right. And E is a ring that you have created above R that itself has dimension l by 2 so around l. So now you have to actually take care of this multiplication since you defined this ring extension ring, you have to give a multiplication algorithm and analyze the time complexity. You cannot see that this is for free. And that messes up our calculation.

So this again gives you l square, okay. This is too expensive because you are multiplying things l many times and each step kind of takes even if it takes l time, it is l time say so at least l square, right. So this idea then seems to be useless to go to an extension. Because now omega has become expensive. So this will need new ideas. So the new idea is you rewrite the input in a way so that the effective degree of your

polynomials becomes around square root l instead of l, okay. So the way you can do it is, have you seen that method?

(Refer Slide Time: 12:32)



How do you rewrite the polynomial of degree l? So instead of the coefficient being an R you collect them in clusters. So this you are actually looking at monomials which are powers of x to the k where k is around square root l and then what will be f i? f i's will now will be polynomials in x themselves not constants of degree at most smaller than k, k -1; at most k – 1, right.

So the monomials you have kind of written in one can say b is x to the k, okay. And then yeah the effect of this actually you will see very soon. Let us first fix the representation. So f is this and g is written like this where you take k to be square root l. So to be precise, let us take square root l by 2 floor. And whatever k is m will be around l over k. Basically k times m has to be l.

So you take m to be l by k and ceiling, okay. So effect is that both k and m are think of them as around square root l, right. So you have kind of you have written f in square root l blocks each of size square root l. This is how you have divided the monomials in an ordered way. And so these f i, g i's are polynomials of degree less than k. And so the idea is essentially that you are thinking of f as a bivariate polynomial.

So you think of f i's in one variable let us say y and x to the k i is in x, okay. You think of f as bivariate and individual degree of each variable is around square root. Well of y is square root l and of x it is still l but point is that all those monomials are just powers of x to the k. So the k is appearing everywhere. So if you remove x to the k, suppress x to the k then the degree of x to the k is also at most square root l, right.

So you have this balancing act. So let us define the bivariate. Actually, this y I am using intentionally. So this y is the same y as before. The y which is present in E, okay? It is not it is not a mistake. So let us consider these polynomials. Big F and big G. So define these and then multiply them. So and let me remind that y is in E, it is that element, the omega element.

So since f i and g i to begin with had degree around square root l you can think of the coefficients as having very small degree right. So in y also they have very small degree and what is the order of y or what was the degree of the moduli of E? l by 2. So in terms of l by 2 this square root l is very small, okay. So these coefficients of big F and big G we have made them small degree.

So when we will evaluate big F and big G at x equal to, let us say y to the j the hope is that multiplication can be done efficiently, okay. It is a low degree multiplication, not a high degree, not of degree l multiplication. So that is the vague hope but we have to obviously implement this carefully to check whether it actually works. So at this point one fact is that there is no information loss.

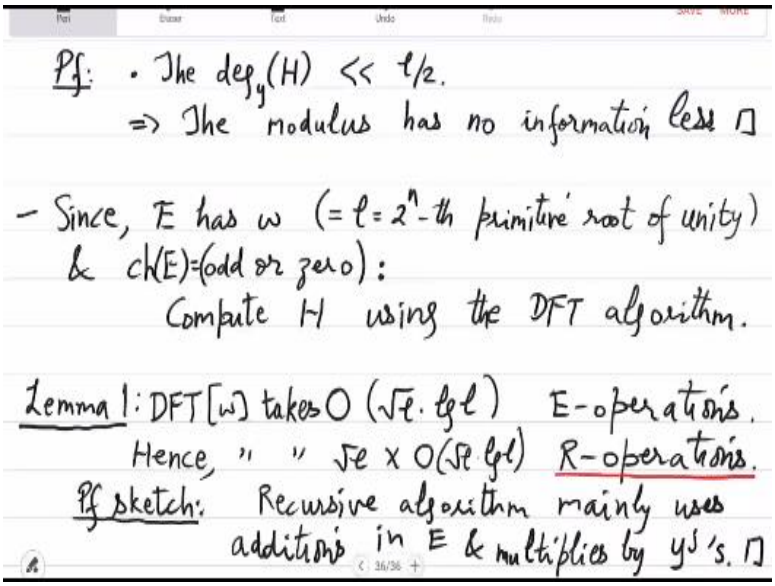
So if you take the product over E you multiply them and call it big H in this is happening in E x. So from big H can you recover small h, the actual product f x times g x? How, how do you recover small h from big H? Yes. So yeah, why does it work? So by substituting y = x in big H, why will you get the product f times g correctly? Exactly. So the argument is actually using the degree.

It is a degree argument. So in big F and in big G well x is anyways free variable. So there is no problem with x. The only problem is with small y because small y is tied with the modulus. So modulus may lose some information. But in this case it actually

does not lose any information. The reason is that small y has degree, individual degree only less than k which is around square root l.

So when you multiply two square root l degree polynomials you only get two square root l which is still far smaller than l by 2. So because of that the arithmetic which is happening in small y, it is absolute. Modulus actually does not change anything. So big H is absolute. So since it is an absolute you can go back and forth between y = x. Okay that is all. So there is no information loss. It is just by degree.

(Refer Slide Time: 22:03)



The degree of H with respect to y is much smaller than l. That is the only thing to note. So and maybe one more thing. The modulus has no information loss. Modulus is not making anything smaller. It keeps it as it is. Okay right. So what you can do now is you do the DFT three step idea on this big F and big G, okay. So big F and big G should be thought of as univariate polynomials in x with coefficients in y.

But that is fine because y is anyways a constant in the ring E okay. So big F and big G are around square root l degree, univariate polynomials over E. You do the three step DFT multiplication and everything will be in terms of square root l, right. That is the first algorithmic sequence of steps you do. So let us implement that. Note that E has omega. So you can do the three step DFT multiplication.

You can do the FFT. So since E has omega, which is l or 2 raised to n-th primitive root of unity FFT is allowed. And obviously, you are also assuming that characteristic

of E is odd or zero. So because of these assumptions you can do compute H, big H using the DFT algorithm that you saw in the last class. So that will immediately give you complexity as let us call it a lemma.

So time taken is how many E- operations? Let us first talk in terms of E- operations, because it is easier. You can directly invoke the previous theorem. So this will be in degree, effective degree here is square root l. So you will get square root l log of that. So that is very fast but it is only a myth because these are E operations. E operations over R is already there is a gap of l right.

So you can multiply this by l. And that will give you l to the 1.5, which is still very good progress over the high school algorithm, right. So this ultimately, I mean at this point already you have a great improvement over l square algorithm for integral polynomial multiplication. So two integral polynomials if you want to multiply on a computer, this is a true algorithm that will take, which will truly take l to the 1.5 times log l.

So maximum it is l to the 1.51, right. It is much smaller than l square. But we will not stop here, we will actually improve this. So let us bring it down to R- operations now. So let us see what are these E- operations that you require. So basically you are multiplying big F with big G, right the coefficients of those. But the coefficients are not maximum degree, they are limited degree. How much is the degree?

It is around root l. So you only have to multiply root l degree coefficients or add them. So that is what E- operations actually means. So how many R- operations? Yeah, so I think we can just multiply this by root l. This is what I want to do. Or let me break it up more. Let me first talk about only DFT omega. So DFT omega takes, let us do it in two steps, not in everything together.

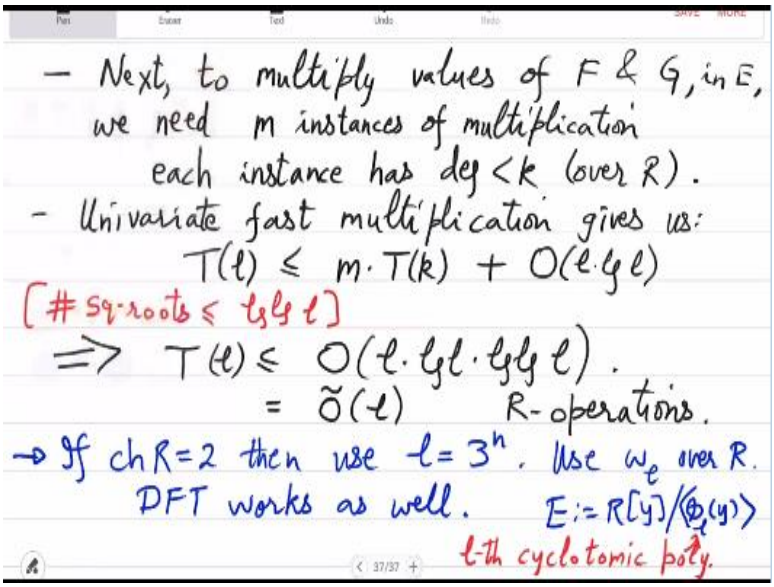
So DFT omega takes these many E- operations. And remember the DFT algorithm was this recursive algorithm, which basically divides the polynomial into two halves and then solve each half and then combines it right. So what are the E- operations in that algorithm? So the claim is that those operations only take square root l more. Do you agree with this?

So it is actually just l log l R- operations. Only talking about DFT omega. The reason is that there was actually no multiplication happening in that. Multiplication was of a very simple type. You were just multiplying by powers of omega, that is all and then you were taking linear combinations. So if your these basic coefficients if they are of degree at most square root l, then going from E- operation to R- operation, you will only multiply by square root l, okay.

So your time complexity is only l log l in terms of R- operations. That is the first claim. So mainly uses E additions and multiplies by omega which is actually y. So multiplications are actually very simple and everything else is addition that is happening in the recursive algorithm. So you just have to multiply by a linear this factor square root l, okay. So that is the DFT omega part.

Once we have done DFT omega on big F big G then you will have to multiply the values. So that is actual multiplication. So we have to now add that. That is a bit more complicated and expensive. So let us look at that.

(Refer Slide Time: 31:21)



Now next to multiply values of big F and big G in E. So the number of values is how many, so the DFT gave you m values, right the effective degree of big F in terms of x. This i here goes from 0 to m - 1. So it is basically m values. So m instances of multiplication, m instances of multiplication and these things that you are multiplying what is the degree of these polynomials in y?

At most k right, m instances of multiplication each instance has degree less than k over R. Yeah so this is what it takes to analyze the multiplication step, which is the second step of DFT idea which we had, right. There were three steps. First is you compute DFT then you multiply then you compute DFT inverse. DFT and DFT inverse kind of we have taken care of by lemma 1.

It is only the multiplication thing, which will be, we have to still analyze and it will be expensive. So in multiplication, what you are doing is you have m instances of multiplication in the ring E which is basically polynomials in y over R and each of these m instances has degree at most k. So what will you do here? This is univariate multiplication. So you use your first theorem, okay.

You use that algorithm you developed again and again m times. So let us do that. So univariate fast multiplication gives us the recurrence if the final time complexity is T l then T l is less than equal to this m times T k. And what is the overhead? Overhead is the DFT part that is l log l. Yeah, so this is the recurrence for the final time complexity of multiplying two polynomials over R of degree l, okay.

So essentially T l is square root l times t square root l. So you have broken into square root l many chunks, each chunk of size square root l plus an overhead of a l log l. So who will solve this? What is the solution of this recurrence as a function of l? So the result you wanted is l log l. Will that be a solution of this, l log l?

"Professor - student conversation starts" m times k is l right? M times k is l yes. In fact, you can think of both of them as square root l for simplicity. Yes. Like if you just substitute with l log l **"Professor - student conversation ends".** Yeah, so l log l seems close to a solution, but you can never prove it. I mean, it is not actually because the RHS will always be slightly more than c l log l.

Whatever c l log l you will fix, RHS will be slightly more than that. So you will never be able to prove, get c l log l. It will be slightly more. So how much more? Yeah, log square l is definitely true. That you can see by fitting it in. Even something better than

that, l log l times log log l, log log because in every iteration you are taking square root. So how many times can you take square root.

So number of square roots possible is bounded by. So l is 2 raised to log l. Every time you are halving log l, right? So that is log log l. So this is a good example of a simple problem where you get a function log log l, right. So you get T l to be l times log l times log log l. That is the algorithm you have. Everything can be suppressed. And this is really soft 2 of l, right?

It is nearly linear time, but a log factor and a log log factor is there. So and remember, these are R- operations. It is not, there is nothing extra being done. These are really R level operation. So this is the actual time complexity if you are trying to multiply two integral polynomials, okay. It will be proportional to l times log l times log log l. Any questions? Okay one unresolved issue is characteristic 2.

What do you do with characteristic 2 or characteristic even? What correction do you use? It is a very simple trick that anyone here can do. The problem was that this l inverse did not exist. You cannot change the ring that was given in the input, but you can change l. So how do you change l such that l inverse exists. Yeah, so if characteristic is 2 you take l to be 3 to the n, right.

I mean there was no reason to get stuck with the 2 raised to n. You use it a power of 3. So then everything works as before. Then use l to be 3 to the n and accordingly use omega l. But this 2 raised to n 3 raised to n has actually no, it has no consequence no bearing on the basic structure of your algorithm. So whatever you did with the previous l the same thing you do with this l with the added advantage that 1 over l exists in E, okay.

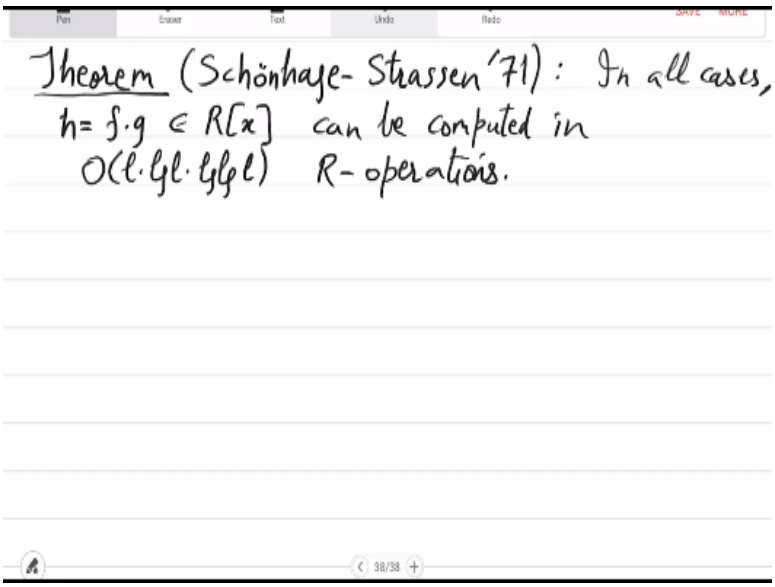
So DFT algorithm works. Is that clear? Yes. No, l by 3. So yeah so that is all. Yeah so but what will you write? You cannot say -1, you cannot say +1; y to the l by 3 + 1 is not good. Yes. So you have to do something else. You do this. Have you seen assignment number 1. So in assignment 1 there is this function defined called or these family of polynomials called cyclotomic polynomials.

So this phi a sub l, in fact phi sub n for any n is defined there. Even for composite n. So you use that polynomial. Yeah. So l here is no whatever is l does not matter whether it is power of 3 or power of 3 times power of 5 whatever. You take phi l function. That was phi l of y. How can that be? No. What is phi l y when l is 2 to the n? So you work it out, it will be that.

So the abstraction of that construction of E is this. You use l-th cyclotomic polynomial. So this is a polynomial that is simply an integral polynomial. Its main motivation comes from complex field. And once you know that how to define it over integers, we just plug it in also in arbitrary rings, okay. So here we will use phi l y. It has all the nice properties that we want.

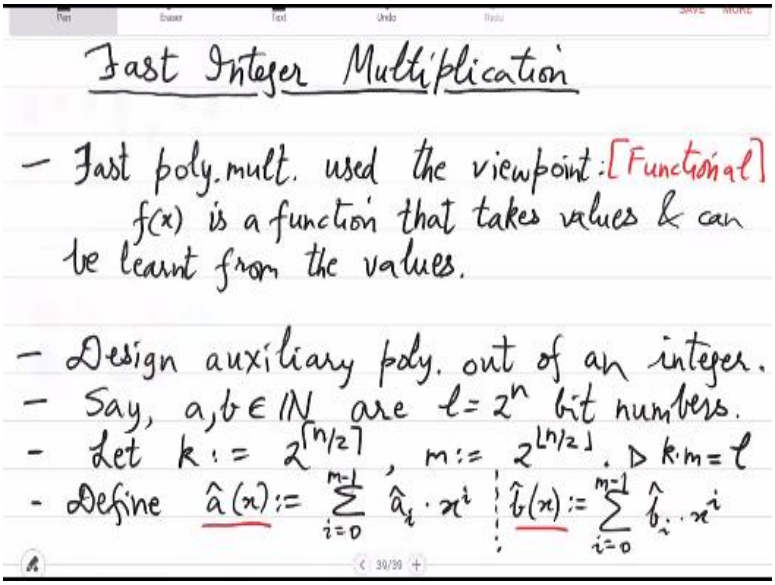
Sigma y to the i will be 0 and so on so forth. But that is not a very difficult modification. This is, once you do the exercise, this is easy to see.

(Refer Slide Time: 42:42)



So this finishes our major theorem that Schonhage and Strassen showed in 1971 that in all cases h equal to f times g in the ring R x polynomial ring over R can be computed in order l times log l times log log l R- operations. Okay this finishes the fastest polynomial multiplication algorithm. Okay, so any questions? If not then we will start a new topic, which is how to use this machinery. And this is a pretty big machinery to multiply numbers.

(Refer Slide Time: 43:58)



Now this fast integer multiplication is a, is an evergreen area. So the algorithm that we will see is probably three decades old. But even now papers come every few years improving it. So we would not be able to cover the fastest algorithms. We will just cover one of the fastest. It already gives you lot of ideas how it is done. But this would not be the state of the art.

So for two l bit integers, right you already know an algorithm to multiply it in multiply them in l square bit operations, l square actual time. How do you make it faster? With all the algebra that you have learned till now how do you bring it below l square? So you define a polynomial out of an integer right. So basically you will do what we did in just the previous slide that you break your you have a sequence of bits defining integer a.

So you break it into let us say square root l chunks, square root l blocks each of size square root l and then define a square root l degree polynomial using these as coefficients right. So you will have now two polynomials of low degree. You will multiply the polynomials using Schonhage-Strassen and then, from that you will deduce the integer. So this is basically what we will try to implement.

So fast polynomial multiplication. Use the viewpoint of evaluate the polynomial and then multiply the DFT; f x is a function that takes values and can be learnt from the values which is interpolation or DFT inverse. So you can call it the functional viewpoint. So instead of thinking of a polynomial as just a sum of monomials, it is

actually a function. So function, when you substitute for an argument, you will get a value.

And these values is what were used to then multiply in the base ring hopefully, and then learn the polynomial product polynomial h. So similar thing you want to do with integers, but obviously integers are not functions. They are already constants, they are already values. So what is done is that we have to artificially create a polynomial, a function out of an integer and then work with that function.

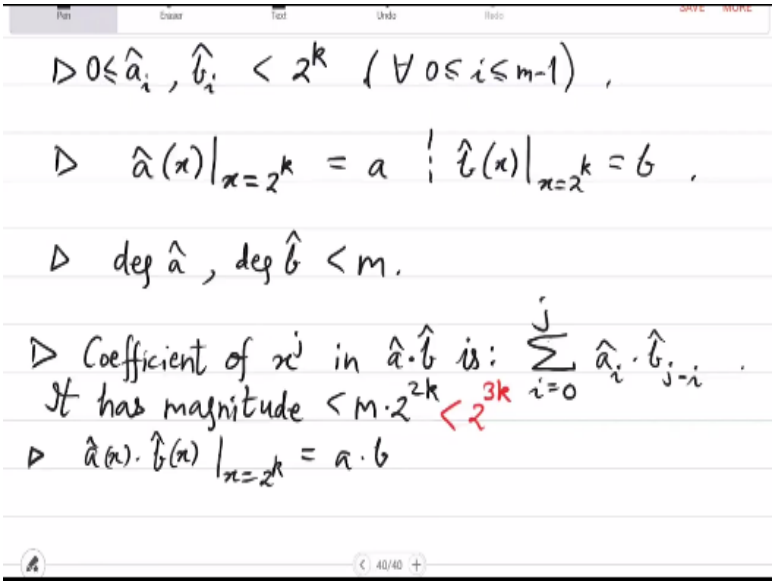
So we will define or design auxiliary polynomial out of an integer. Once you have a polynomial you have a function and then multiplication of functions use this older philosophy. Okay. So how do you define this auxiliary polynomial? So let us say a, b are your numbers, positive numbers, l bits l the same as before 2 raised to n bit numbers. So as promised we will break into square root l blocks equisized.

So let k be around square root l. This is around 2 raised to n by 2, which is same as square root l and m is the m is a lower k. So m is 2 raised to. So basically, km is l, okay. So in the simplified case this is the same as square root l will be the degree of the polynomial and the coefficients will have how many bits? Coefficients will be integers with how many bits? Around square root l, right.

So you will, this integer a you will be looking as a polynomial of square root l degree and the coefficients are integers which are small. So they are square root l bits. So does define that. So a hat x polynomial, out of a, this auxiliary polynomial is a i hat is the ith coefficient, x to the i, i goes from 0 to m - 1. And similarly b hat. So degree of a hat b hat is m or m – 1. They are m monomials.

And a i hat b i hat are small integers. How many bits? k bits. In a hat you have m many k bit integers, right. This is the polynomial corresponding to a. And then similar analogous thing for b. Okay, so remember this definition. So let us move. Let us recall these properties.

(Refer Slide Time: 51:37)



So first is that a i hat and b i hat are numbers at most 2 raised to k, k bit. Okay, this is the range of a i hat b i hat for all i. And how do you go back from a hat to a. So from a to a hat sorry, 2 power k. So if you substitute x equal to 2 raised to k then you get back a, right. So which tells you that there is no information loss whatsoever. In fact, these polynomials in a way know more, they give you more structure not less.

So once you have a hat, a single value of it is a and same with b hat. So what you will do is you will now use the use fast polynomial multiplication to get a hat times b hat and let us call it probably c hat. And in that c hat you substitute x to be this or some, basically some value of a hat times b hat will give you a times b, right. That is the plan. But we have to check whether this actually happens.

Degree also I can say about the degree. Degree of a hat, degree of b hat is less than m. So let us look at the coefficient in the product, okay. So coefficient of x to the j in a hat times b hat. What is that? So that is the convolution how many ways can you get j, right. So this will be sigma i from 0 to j, a i hat times j – i, right. So a i hat and b j - i hat these are the numbers, which are the coefficients of a hat b hat.

And these products you have taken and sum it up to get the coefficient of x to the j in the product. So how big is this number, sum of the these products? So j + 1 will be at most yes m. So it is m times a i hat and this b hat b j – i hat we know is 2 to raised to k. So 2 raised to 2 k. m was yeah, so I want to claim that this is smaller than 2 raised to 3k. Is that right? Basically, m is at most 2 raised to k.

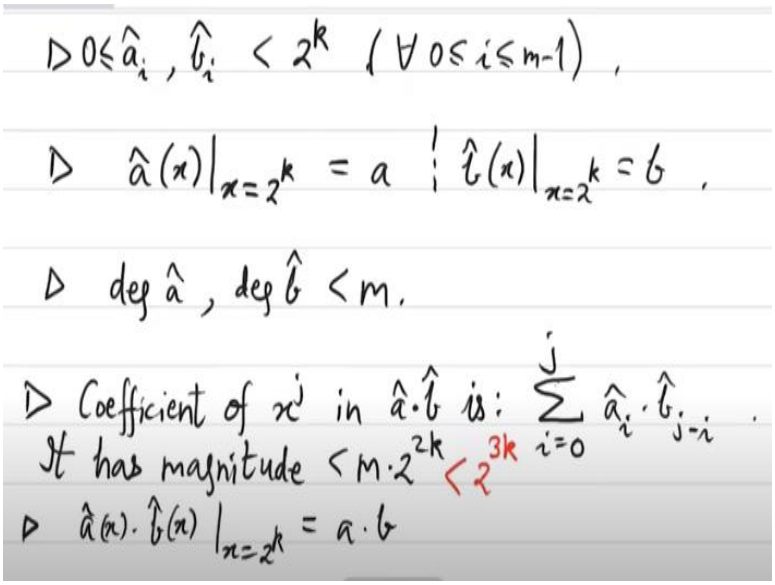
I mean m is much smaller than 2 raised to k. So m is around square root l and k is also around square root l. So this is actually much smaller than 2 raised to 3k. So point is that when you multiply these two polynomials, the coefficients do get bigger, but not by too much, right. So from 2 raised to k, they have only grown by a cube, at most a cube, more like a square actually, right.

So they super quadratically grow. So once you have computed a hat times b hat polynomial, how do you get a times b? Yeah, so actually, no I should rephrase my question. I want yeah maybe at least what we know we should write down. So you know that a x times a hat x and b hat x at x equal to 2 raised to k. Individually, they are both a times b. So you do get a times b that is true.

Okay, so the question is, how will you do this multiplication a hat times b hat? So to use the fast polynomial multiplication you have to go to a ring extension, right. Because these polynomials are an integer. So in integers you do not have a degree square root l. So you do not have square root l-th primitive roots of unity. So you have to go to an extension and do all that arithmetic.

To save on time and to get a algorithm as fast as possible we will not use that approach. We will use a more sophisticated approach, tailor made to this case to get a faster algorithm. So what we will do is we will actually go modulo 2 raised to 3k + 1, because that gives us a 2 kth primitive root of unity. So let me write that idea down. That is a diversion from fast polynomial multiplication.

(Refer Slide Time: 58:22)



I will tell you why we need this. Maybe we should discuss that first. So if you use fast polynomial multiplication at this point to compute a hat times b hat what will you get? You will get that O tilde l many Z operations, right. It is still integer operation. So but the integers here are how many bits? To do this polynomial multiplication you are doing you are working with square root l bit integers.

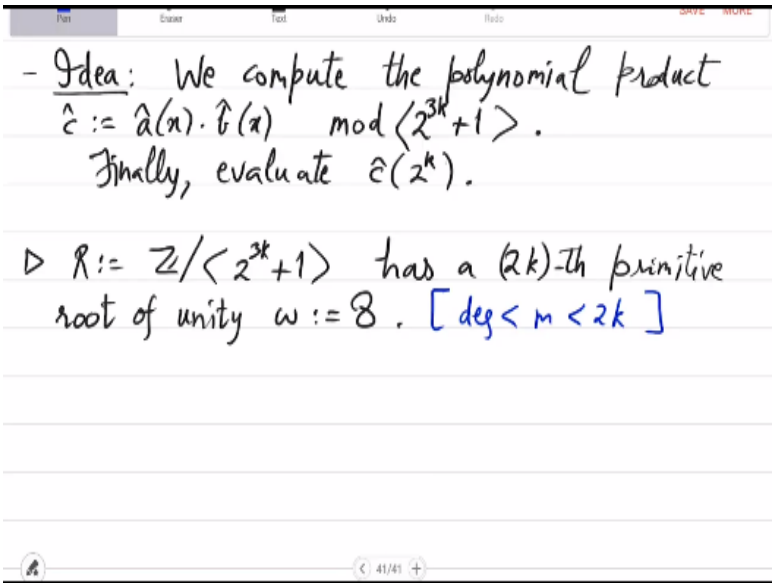
So but your original goal was to multiply l bit integers. So l bit integer multiplication has been kind of reduced to many instances of square root l bit integer multiplication. But not just one instance, many instances because when you will do this fast polynomial multiplication, there will be DFT and there will be multiplication and all that right.

So if you do that analysis, which will take hours, ultimately you will not gain that much. It will be a faster algorithm, but it will not be O tilde l time algorithm. So time means you have to reduce to bit operations, not just stop at integer operations. Because well, two integers can be multiplied in one integer operation, right. So that was anyways trivial to begin with. So that is not actual time.

"Professor - student conversation starts" Even this evaluation gets down to multiplying root l bits when you are evaluating at x equal to 0. **"Professor - student conversation ends".** Right. So root l many root l bit multiplication of integers, yes. Yeah. So just talking about Z operations is not enough. We really need actual time which is bit operations.

So which is why we have to actually optimize on this extension business which we did. I mean, in general we have to optimize the whole algorithm, but our starting point will be slightly different.

(Refer Slide Time: 1:00:53)



And that is why I introduced 2 raised to 3k. So we will actually compute the polynomial product a hat x times b hat x modulo small numbers. So 2 raised to 3k + 1 okay. So we will not we basically want to save on the integer arithmetic as much as possible. And so to do that we actually go modulo this 2 raised to 3k + 1 and then finally so this a hat b hat is c hat. Finally, evaluate c hat at 2 raised to k.

Okay that is the idea. Yeah, it will help because, so we will not just call the fast polynomial multiplication algorithm as a black box, because that does not help us. So we will actually be inspired by that algorithm and give a different algorithm. So in that algorithm, this key problem was going to an extension. So that part actually we will optimize. We will give a full algorithm here, we will not invoke that algorithm.

Formally we will not invoke. But you will see that it is very similar to that algorithm. So the ring where we are working in is the ring of integers modulo 2 raised to 3k + 1. Now this has a 2k-th root of unity which is. So what is this omega? Well 2 cube. So 2 cube is 8 and in this ring a to the k + 1 is 0. So 8 has order 2k, right. So omega is 8. So that is perfect because you do not need a complicated construction to get omega, okay.

Omega is just as simple as going modulo that number. So that will save us a significant amount of this ring extension business that we did in fast polynomial multiplication and 2k should be enough because your polynomials have degree less than m. And m is less than 2k, right. So because of this 2k is more than what we needed, okay. So this is the right kind of omega, looks like.

I mean the details you will see next time. But it will be a specialized analysis of what you saw for polynomial multiplication, okay. So I think we should stop now.