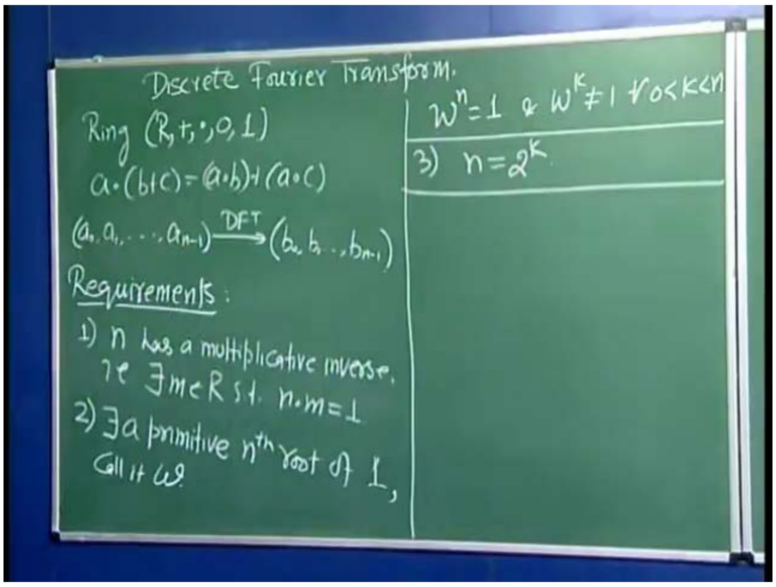
Computer Algorithms - 2
Prof. Dr. Shashank K. Metha
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 23
Discrete Fourier Transformations-I

Hello, so today we will discuss Discrete Fourier Transforms and an efficient algorithm to compute these. Many of you might have been might have seen Fourier transforms of functions, in case of Discrete Fourier Transforms, we will be computing a transform of a vector, which you can think of the function with n value.

(Refer Slide Time: 00:51)



Discrete Fourier Transform.

Ring $(R, +, \cdot, 0, 1)$

$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

$(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{DFT} (b_0, b_1, \ldots, b_{n-1})$

Requirements:

1) n has a multiplicative inverse. ie $\exists m \in R$ s.t. $n \cdot m = 1$

2) $\exists$ a primitive n^{th} root of 1, Call it ω

$\omega^n = 1$ & $\omega^k \neq 1 \ \forall 0 < k < n$

3) $n = 2^k$

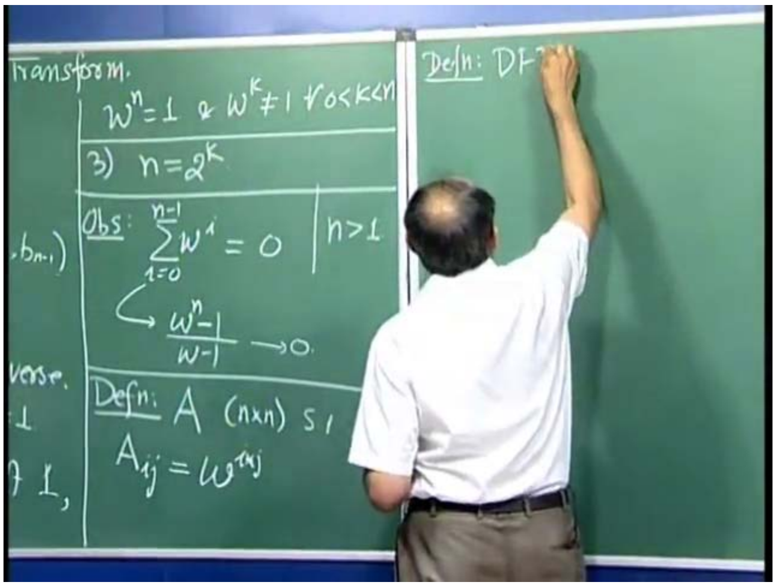
So, what we have here is a ring where R is a set, and which has the two binary operations, the plus and multiplication. It has two special elements in a in it so called 0 and 1. This is a ring in which multiplication is commutative. So, it is a commutative ring with a unit element, other properties we have earlier discussed are the same namely with plus R is a group, which means for every element that is in inverse. So, you can have a, there is in negative a says that their sum is 0, a plus 0 is 0 with multiplication one plays the role of identities.

So, a times 1 is a, 1 times a is a because it is commutative, by commutative we mean a times b is equal to b times a; and multiplication distributes over plus, which means a times b plus c is a times b plus a times c. This holds now given a vector on this ring, so components of the vector belong to this, so I have n vector a 0 a 1and a minus 1. Its discrete Fourier transforms is a vector b 0 b 1 b n minus 1. There are many useful properties this transform has and helps us compute certain quantities which are difficult to directly compute in terms of complexity.

So, we will first formally describe the Fourier transform and later on will deduce certain properties and describe an algorithm to compute the DFT which is discrete Fourier transform more efficiently. So, let us try to describe the properties, we need the ring. The requirements are first that n has a multiplicative inverse that is there exists m in the ring such that n times m is 1. This is generally not necessarily available in the ring such as this two, we also need that there exists, a primitive nth root of unity of 1, call it omega 1. This means that I will just right down w or omega power n is 1, which makes it nth root of unity and omega power k is not 1 for all k between 0 and n.

These two requirements are there in addition to the fact that this is a commutative ring with unity. We further insist on n to the power of 2 although this is not an essential property, but this makes the competition efficient and usually such requirements is not a burden because we can always extend the vector by padding with 0’s. So, we will also add a third requirement notice that this is only for efficiency in competition, so we will add the requirement that n is a power of 2. Now, we are ready to define the discrete Fourier transform of a vector with n components, so let us establish a few properties.

(Refer Slide Time: 06:32)



Transform.

$w^n = 1$ & $w^k \neq 1 \ \forall 0 < k < n$

3) $n = 2^k$

Obs: $\sum_{i=0}^{n-1} w^i = 0$ | $n > 1$

$\hookrightarrow \frac{w^n - 1}{w - 1} \rightarrow 0.$

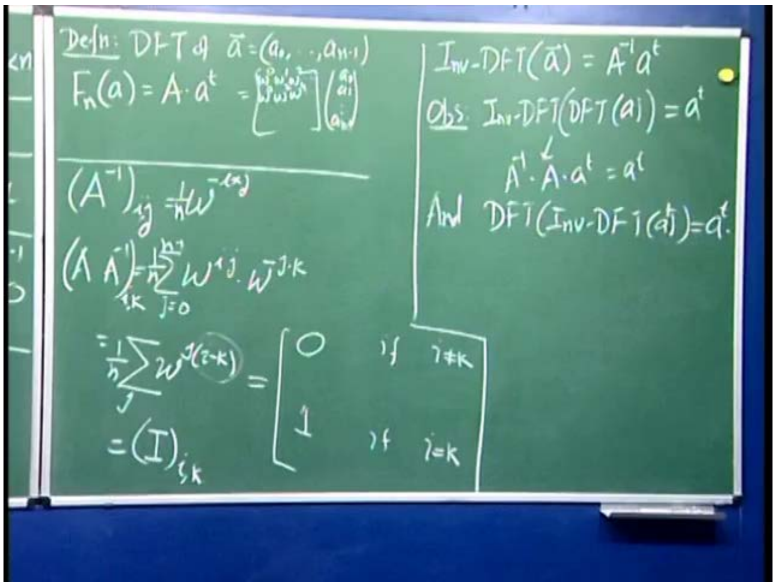
Defn: A $(n \times n)$ s.t.

$A_{ij} = w^{i \times j}$

Defn: DF

Observation says that some omega I, i going from 0 to n minus 1 is 0. This is trivial because this is geometric series and this is nothing but w power n minus 1 over w minus 1 this is 1 and this is 0. We know that w, we are assuming that n is greater than 1. So, w power 1 is not 1 and this makes it 0. Let us define a matrix A which is n cross n such that A i j the i j th element of this is w i times j.

(Refer Slide Time: 08:10)



Defn: DFT of $\vec{a} = (a_0, \ldots, a_{n-1})$

$F_n(a) = A \cdot a^t = [\ \ ] \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}$

$(A^{-1})_{ij} = \frac{1}{n} w^{-(i \times j)}$

$(A A^{-1})_{ik} = \frac{1}{n} \sum_{j=0}^{n-1} w^{i \cdot j} \cdot w^{-j \cdot k}$

$= \frac{1}{n} \sum_j w^{j(i-k)} = \begin{bmatrix} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{bmatrix}$

$= (I)_{ik}$

$\text{Inv-DFT}(\vec{a}) = A^{-1} a^t$

Obs: $\text{Inv-DFT}(\text{DFT}(a)) = a^t$

$A^{-1} \cdot A \cdot a^t = a^t$

And $\text{DFT}(\text{Inv-DFT}(a^t)) = a^t.$

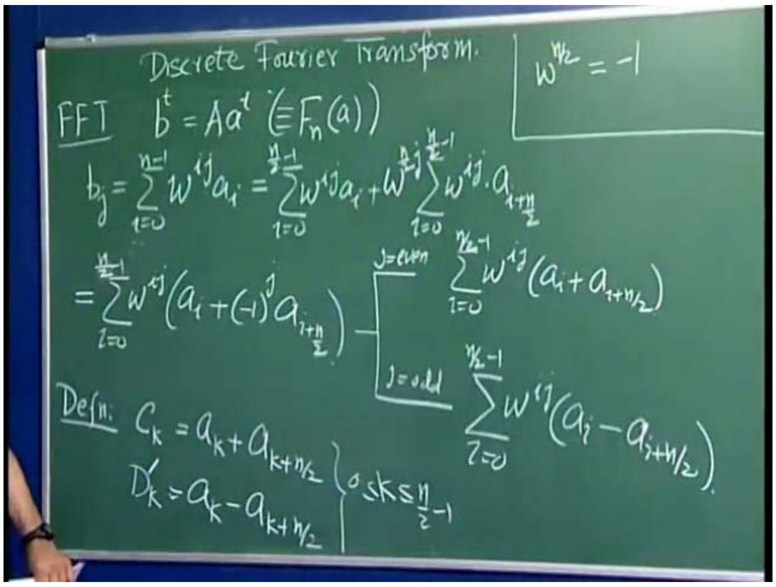
Then, the Fourier transform, the discrete Fourier transform of vector a which is a 0 to a n minus 1 is F n of a equal to a and these are the entries w power 0 w power 1 w power 2 then w power 0 w square w 4 0 1 2 0, w square w 4. Now, I probably need is strong result then this so I will say i times j will be seen as w j power n minus 1 by w j minus 1 and of course, here i and j are, note that are i and j are between 0 and n minus 1. Hence, this is w power n power j minus 1 over w power j minus 1, this is 1 power j is 1. This is 0, this is not 0 because w is a primitive n root hence, more generally this sum is 0.

Now, using this result we will show that there is a inverse transform. So, let us define A inverse as the i j th element of this, w i times j with a negative sign. So, this is the matrix, notice that this is inverse because time A inverse is sum w i, let us say this is i k with element of this i j into w minus j times k, j going from 0 to n minus 1, which makes it j. I would have to add coefficient 1 over here, so I will have 1 over here as well this is 1 over n w j, i minus k, i minus k is a constant for a fixed i and k as long as i and k are not equal. This quantity is other than 0. So, let us consider i naught equal to k, this is not equal for j, here we will exclude 0 value because if j is 0, then this quantity becomes 1 and this whole thing adds up to n.

So, we will help to refrain from making, so our j is between 1 and n minus 1. If this is not 0, then this sum is 0, so we get 0 values. If i is equal to k this is 0, this quantity is 1 and this whole sum is nothing but m divided by n. It becomes 1 if i is equal to k which means this is the identity matrix the i k th value of that which is establishing the factor indeed this is the inverse of it. Well in case, this is inverse then we define inverse DFT of a vector a to be A inverse a. Now, observe that DFT inverse or I would simply say inverse DFT of DFT of a should be a, because this is nothing but A inverse of A of a, which is just a transpose and DFT of inverse DFT of a is also transform.

Hence, we have a very simply notion DFT or discrete Fourier transforms of vector is nothing but the product a times similarly, the inverse discrete transform is A inverse applied on vector a. Now, next we are going to see how to compute this efficiently, in efficient algorithm, which takes the advantage of sati structure of this w's is called fast Fourier transform or just FFT.

(Refer Slide Time: 15:51)



Discrete Fourier Transform.

FFT $b^t = Aa^t \; (\equiv F_n(a))$

$w^{n/2} = -1$

$$b_j = \sum_{i=0}^{n-1} w^{ij} a_i = \sum_{i=0}^{\frac{n}{2}-1} w^{ij} a_i + w^{\frac{n}{2}j} \sum_{i=0}^{\frac{n}{2}-1} w^{ij} \cdot a_{i+\frac{n}{2}}$$

$$= \sum_{i=0}^{\frac{n}{2}-1} w^{ij}\left(a_i + (-1)^j a_{i+\frac{n}{2}}\right)$$

$j = \text{even}$: $\sum_{i=0}^{n/2-1} w^{ij}(a_i + a_{i+n/2})$

$j = \text{odd}$: $\sum_{i=0}^{n/2-1} w^{ij}(a_i - a_{i+n/2})$.

Defn: $C_k = a_k + a_{k+n/2}$, $D_k = a_k - a_{k+n/2}$ $\Big\}\; 0 \le k \le \frac{n}{2} - 1$

Let us suppose, we have b is a of a which is the Fourier transform so from now onwards we will just use this symbol to denote the discrete Fourier transforms of vectors of size in components. So, we have b j from here sum w i j ai here i goes from 0 to n minus 1, this is from this definition we directly get this sum, I am going to split in the middle. Now, you notice why we had assumed n to be a power of 2, I can always divided this into two parts, we will take from 0 to n by 2 minus 2 n by 1. So, this is w i j a i and i is from 0 to n by minus 1 plus sum w i j. Notice that in the second half of the sum the value of i is always greater than equal to n by 2.

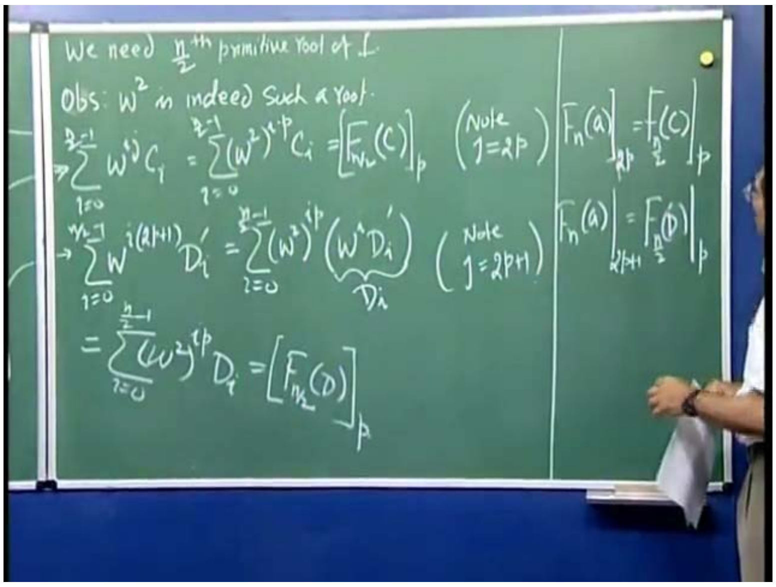
So, n w power n by 2 times j term can be the factor can be taken out this is n by 2 times j. Now, I can start from 0 to n by 2 minus 1 and here you have a i plus n by 2. So, I have just re rewritten the second half this way my indices, i instead of going from n by 2 to n minus 1. Now, will go from 0 to n by 2 now w power n by 2 w power n by 2 is minus 1 because the square of this is 1, this cannot be 1 because w being a primitive root of unity, so the only quantity it can be is minus 1. Hence, this quantity is nothing but minus 1 power j, so we are going to get sum i going 0 to n minus 1 w power i j a i plus minus 1 power j a i plus n by 2 this expression simplifies to this. Now, let us split this entire sum into two separate sequences, this is n by 2 minus 1 n by 2 minus 1 because the combining these.

So, let us now take the two cases separately one when j is even and the other j is odd. So, let us say here we take j to be even, I am going to have this sum to be i going from 0 to n by 2 minus 1 w i j ai plus this is w power n by 2. So, this is fine this makes it a i plus n by 2. In case when j is odd we will have this sum i going from 0 to n by 2 minus 1, so this quantity is minus 1 we are going to write down w i j ai minus ai plus n by 2. Now, the intent is that somehow we reduce this, both of these in to a Fourier transforms of vectors of only n by 2 components, notice that we have a sum running over n by 2 quantities each.

If we define new vectors, which is folded like this the higher component is higher component of the original vector plus n by 2 plus higher component of the original vector. If I take that similarly, we do something here and try to reduce this task of computing Fourier transforms of n component vector into two Fourier transform of half the size then I am going to get a an efficient algorithm. Hence, let me now define two vectors C and D as follows, so let us define for the sake of even j.

Let us define C k you should think of j as 2 k as a i k plus a k plus n by 2, sorry so this is just this k is nothing but this i here. Now, let us look at this vector, let us define our D k as a D prime k, I would say a k minus a k plus n by 2. So, this the initial impression that we would like to create to half size in both this cases by k runs from 0 to n by 2 minus 1, but when you talk about Fourier transform of n by 2 component vector then we have to a suitable primitive root of unity.

(Refer Slide Time: 24:50)



We need $\frac{n}{2}$th primitive root of 1.

Obs: w^2 is indeed such a root.

$$\sum_{i=0}^{\frac{n}{2}-1} w^{ij} C_i = \sum_{i=0}^{\frac{n}{2}-1} (w^2)^{i \cdot p} C_i = \left[F_{n/2}(C)\right]_p \quad \begin{pmatrix} \text{Note} \\ j = 2p \end{pmatrix}$$

$$\sum_{i=0}^{n/2-1} w^{i(2p+1)} D_i' = \sum_{i=0}^{\frac{n}{2}-1} (w^2)^{ip} \underbrace{(w^i D_i')}_{D_i} \quad \begin{pmatrix} \text{Note} \\ j = 2p+1 \end{pmatrix}$$

$$= \sum_{i=0}^{\frac{n}{2}-1} (w^2)^{ip} D_i = \left[F_{n/2}(D)\right]_p$$

$$F_n(a)\Big|_{2p} = F_{\frac{n}{2}}(C)\Big|_p$$

$$F_n(a)\Big|_{2p+1} = F_{\frac{n}{2}}(D)\Big|_p$$

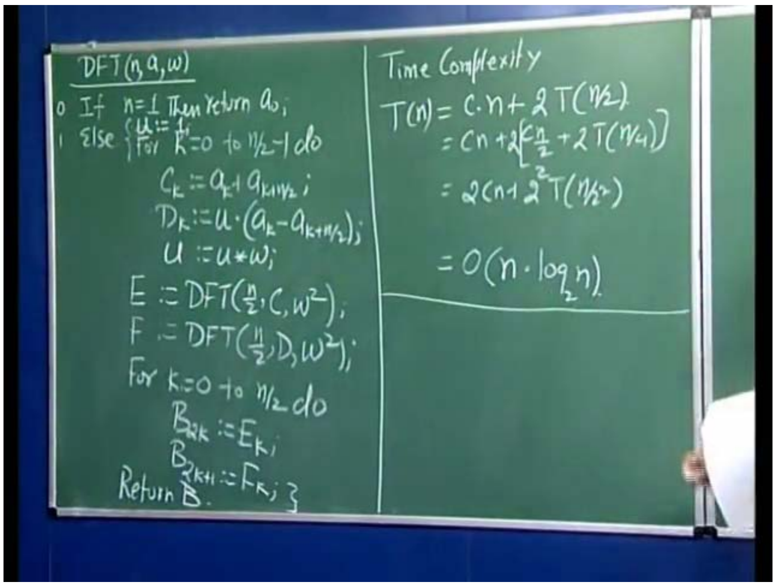
So, now we need n by 2 th primitive root of one in order to define transform of half size. So, notice that w square, so let us observe here that w square is indeed such a root there is an w square power n by 2 is w power n which is 1, any smaller root this one the power w square power j will be w power 2 j, if j is less, then n by 2 then 2 j is less than n. Hence, it will not be 1, so this is a primitive n, but with root of unity, so what we will have to do here is write the sum i going from 0 to n over 2 minus 1.

Now, let me suppose j is 2 p, the even case when j is 2 p, I can write down i j C j C i which is this quantity as sum i going from 0 to n over 2 minus 1. Replace j by 2 p you are going to get w square i times p C i and now this is the p th component of the Fourier transform of this vector. So, we are now successful in transforming this into Fourier transform problem of half size. Let us now look at the second this sum goes from, i from 0 to n by 2 minus 1, here I would like to keep in mind that j has been replaced by 2 p. Hence, we get this, so keep in mind that the p th element of this Fourier transform is actually 2 p th element of the original transform, so you have here w i 2 p plus 1.

Now, in this case j is suppose to be odd and replacing this by 2 p plus 1, this is D prime i this i can write down as sum w square i p w power i D prime i and lets call this quantity as D i. So, the sum now is i goes from 0 to n by 2 minus 1 w square power i p D i, which is the Fourier transform of half size of vector D and this is the p th notice that here j was replaced by 2 p plus 1. So, when p goes to 0, j is 1 which is the first odd index. Hence, for j going from 1 to 3 to 5 and so on or p will go from 0 to 1 to 2 and so on. So, we have again a Fourier transform of half size and we are computing the p th quantity in this.

So, finally, what we get is that F n of a p or rather I would say 2 p of this is the F n by 2 of C the p th value and F n of a 2 p plus 1. Component is same as F n by 2 C, rather D th component and this is the p th component of D th vector this is the p th component of this Fourier transform. So, all we have to do is compute the vectors C and D compute the half size Fourier transform and then match them together first from here, second from here, third from here, fourth from here and so on. Now, you have got the Fourier transform of a.

(Refer Slide Time: 32:04)



```
DFT(n, a, w)
0 If n=1 Then return $a_0$;
1 Else { $u := 1$;
         For k:=0 to n/2-1 do
             $C_k := a_k + a_{k+n/2}$;
             $D_k := u\cdot(a_k - a_{k+n/2})$;
             $u := u * w$;
         $E := DFT(\frac{n}{2}, C, w^2)$;
         $F := DFT(\frac{n}{2}, D, w^2)$;
         For k:=0 to n/2 do
             $B_{2k} := E_k$;
             $B_{2k+1} := F_k$; }
     Return B.
```

Time Complexity

$$T(n) = c\cdot n + 2T(n/2)$$
$$= cn + 2\left[c\frac{n}{2} + 2T(n/4)\right]$$
$$= 2cn + 2^2 T(n/2^2)$$
$$= O(n \cdot \log_2 n)$$

Now, let us write the algorithm. Discrete Fourier transforms of a size n of a vector a and the we also give the primitive root n th primitive root. So, step 0 is, if n is 1 then return is 0. Notice that Fourier transforms of a single component vector is itself, so you can verify that else. Now, we will have to prepare to split this vector. Generate those two smaller vectors to compute the Fourier transforms and then combine them to compute the Fourier transform of the big vector.

So, I will have for k from 0 to n by 2 minus 1, we will define these two vectors C k is a k plus a. So, I am going to have a, which I will show you why you need u equal to 1 here, a k plus n by 2, this was the first smaller transform, this is the vector for that and D k.

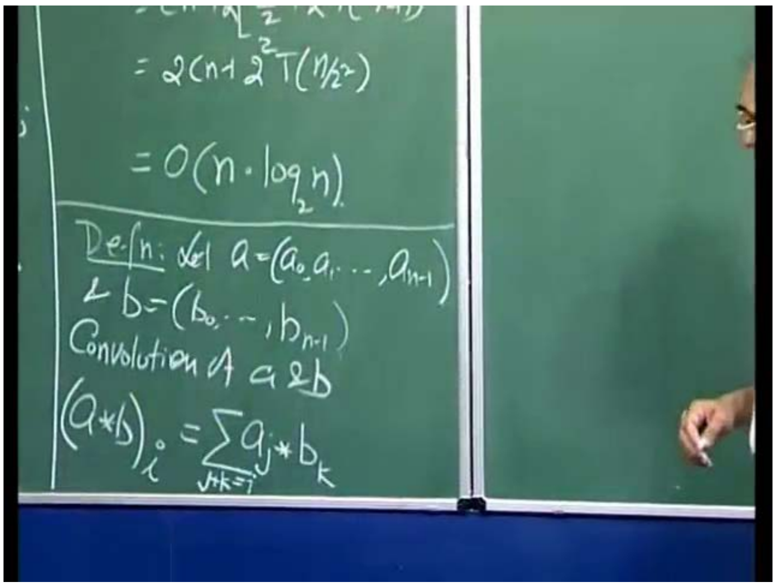
Recall that we had D prime and D prime was simply the difference. But in D k, we will have w power k times this. So, for each component we will have different power of w. That is what you will be, so we will have u times a k minus a k plus n by 2 and then will upgrade u, so u will be time w i.

Hence, u is always w power k when k was 0, u was 1, first time subsequently it keeps on growing. Now, we had prepared the two vectors you are going to compute the two Fourier transforms, so let us call them we have C and D. So, let us call it E this, so this is array containing the Fourier transform of vectors C, D, F, T of n by 2 C w square F. Similarly, so make a recursive call for this two smaller transforms. Now, combine the for, combining we need again for k going from 0 to n by 2, let us define B or rather we have been using thus the index, we will just write the 2 k is same as E k and 2 k plus 1 is F k.

Now, we have done, so we can know return our vector B, this is the Fourier transform of our input a, so the complexity I should probably put a closing. The main computation is generating these n components n by 2 components of C an n by 2 components of D. Then we make a recursive calls an again we generate this n components of the Fourier transform. So, what we have is time complexity of computing an n component of Fourier transform is some C times n plus 2 transforms of n by 2. The next time of course will have C n plus C n by 2 into 2 plus 2 t n by 4 which is same as 2 C n plus 2 power 2 t n by 2 power.

Hence, each time we are going to get C n term as we go along, but this is going to reduce to simple case in only log and steps because every time it is having hence this is order n times log n this is the time complexity of fast Fourier transform. Now, we are ready to show a very useful application of Fourier transform and that we will use in giving an efficient algorithm for computing product of two polynomials.

(Refer Slide Time: 39:50)

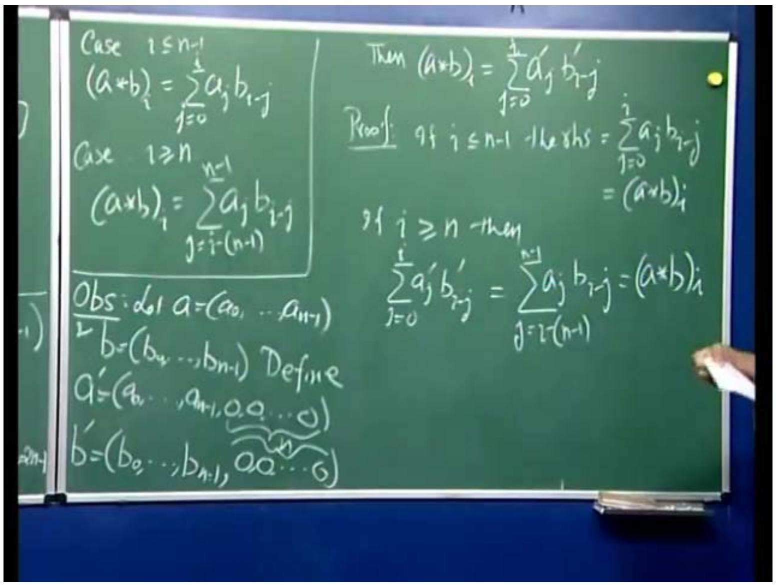


$= 2cn + 2^2 T(n/2^2)$
$= O(n \cdot \log_2 n)$
Defn: Let $a = (a_0, a_1, \ldots, a_{n-1})$
& $b = (b_0, \ldots, b_{n-1})$
Convolution of a & b
$(a*b)_i = \sum_{j+k=i} a_j * b_k$

So, let me first define something called convolution of two vectors definition. So, let a be a 0, a 1, a n minus and we have another vector b which is b 0 to b n minus 1. Then convolution which I am going to define convolution of a and b, which is a star b is also a vector, but the size of this vector is 2 n and the i th component of this is sum of a j times b k such that j plus k is i.

So, let us now expand this and see exactly, we are getting, notice that here my i is 0 1, all the way to 2 n minus 1. Although, you will notice that it will never have any case of 2 n minus 1, the maximum such a thing is possible is 2 n minus 2. But we will force this extra component just to keep the things of a multiple of 2 in sizes; in that case we are going to define the 2 n minus first component as 0. Let us just open this and see what exactly this is giving us as long is our i is less than is equal to n minus 1.

(Refer Slide Time: 42:03)



Case $i \leq n-1$

$(a*b)_i = \sum_{j=0}^{i} a_j b_{i-j}$

Case $i \geq n$

$(a*b)_i = \sum_{j=i-(n-1)}^{n-1} a_j b_{i-j}$

Obs: Let $a = (a_0, \dots a_{n-1})$ & $b = (b_0, \dots, b_{n-1})$ Define

$a' = (a_0, \dots, a_{n-1}, \underbrace{0, 0, \dots 0}_{n})$

$b' = (b_0, \dots, b_{n-1}, \overbrace{0, 0 \dots 0})$

Then $(a*b)_i = \sum_{j=0}^{i} a'_j b'_{i-j}$

Proof: If $i \leq n-1$ the rhs $= \sum_{j=0}^{i} a_j b_{i-j} = (a*b)_i$

If $i \geq n$ then

$\sum_{j=0}^{i} a'_j b'_{i-j} = \sum_{j=i-(n-1)}^{n-1} a_j b_{i-j} = (a*b)_i$

So, let us take a look at case one is i less than or equal to n minus 1, in that case a 0 b i a 1 b i minus 1 and so on, we will contribute. So, we will get a star b i as sum a j b i minus j, where j goes from 0 to i, this is what we will get. The other case is i is greater than equal to n. Now, let us take a case when say i is n plus 1 just for the sake of an example. Imagine i here n minus 1, n, n plus 1, so if it is n plus 1 then a 0 will not be able to contribute anything because this is 0, maximum this can go is n minus 1. They will not contribute anything, 1 n minus 1 add up to n still does not contribute, but it will start from 2 and go up to n minus 1.

So, when it is 2, it will be n minus 1, n minus 3, n minus 2, minus 1, so when we are looking at i th component then i minus n minus 1 is the first index that will contribute. So, we will have a star b i equals a j b i minus j i minus j and our j starts at i minus n minus 1 and goes up to n minus 1, this is how the two cases plate in this case the upper side does not contribute and the lower case lower side does not contribute. So, this is what we have, for convenience usually the splitting into two cases is a hard thing to deal with.

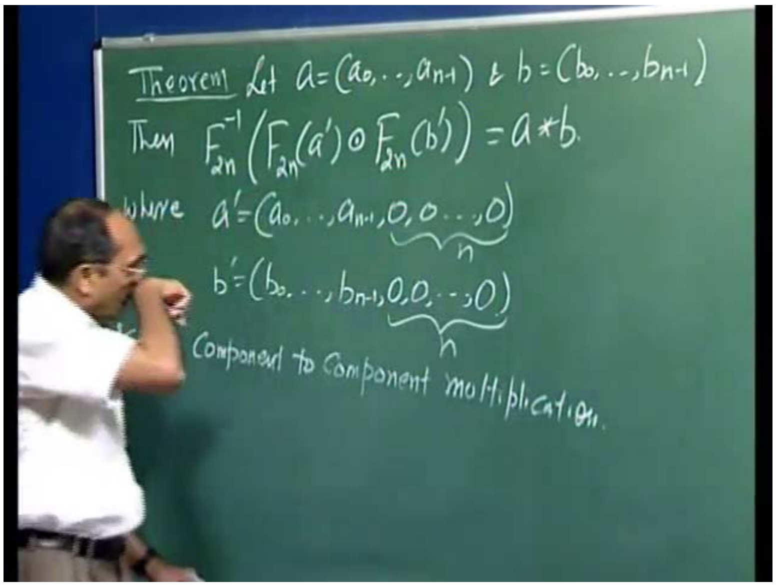
So, we are going to show that this can be reduced into a single expression. If we extend our vectors in the following fashion and what I am going to do is define new vectors as follows. So, let us say let again we have these n 0 to a n minus 1 and we have b which is b 0 through b n minus. Define you are a prime as a 0 to a n minus 1, and then padded with n 0’s, so these are n of them similarly, define b prime as b 0, b n minus 1, 0 again these are also n 0’s. So, we define this extended vectors of size two and then a star b the i th component is sum a j a prime j b prime i minus j and j goes from 0 to i.

Let us verify this again. We are going to just take the two cases and verify that they match with these, so first if my i is less than or equal to n minus 1, notice that the right hand side will be same as j equal to 0 to i a j b i minus j. The reason is that in this case all the indices are within n minus 1 because my i is at most n minus 1. My j runs only up to I, and my i minus j is also contained, because the lowest value this can have 0, highest is i. All these values are contained in the range 0 to n minus 1, in that range a prime is a b prime is b.

Hence, this is indeed equal to this, so this is actually a star b i from that case one expression and if i is greater than equal to n. Then what is going to happen is that this expression j from 0 to i a prime j b prime i minus j. Keep in mind, every index greater than n minus 1 is 0; so in case say we have i equal to 1, sorry n minus a, if i is n and j is 0 then we will have a prime 0 b prime n which is 0. It will not contribute anything and this will go on until both the indices are within n minus 1.

So, we will drop those terms which have contributed nothing, so which is the first value of j when we have a non zero contribution, well when this is n minus 1. If this is n minus 1 then j is i minus n minus 1 i minus n minus 1, the largest j can go in n minus 1 beyond that it is 0 anywhere. Now, of course both this terms are non-zero and when they are non zero then a prime is a and b prime is b. So, this becomes a j b i minus j and which is again as we see this is exactly the same, we are getting this so we have now a single expression for the convolutions of the two vectors. Now, I am ready to state a theorem and we will prove that in the next lecture.

(Refer Slide Time: 51:07)


Theorem Let $a = (a_0, \ldots, a_{n-1})$ & $b = (b_0, \ldots, b_{n-1})$
Then $F_{2n}^{-1}(F_{2n}(a') \odot F_{2n}(b')) = a * b$.
where $a' = (a_0, \ldots, a_{n-1}, \underbrace{0, 0 \ldots, 0}_{n})$
$b' = (b_0, \ldots, b_{n-1}, \underbrace{0, 0, \ldots, 0}_{n})$
Component to Component multiplication.

Now, finally, I am going to state a theorem which tells you how to compute the convolution of the two vectors efficiently which actually uses Fourier transform. The proof of the theorem, we will do it in the next lecture. So, the statement that suppose we have two vectors let a, b, a 0 to a n minus 1 and b be 0 to b n minus 1. Then inverse Fourier transform of 2 n vectors of Fourier transform of 2 n vector, a prime F 2 n b prime is a star b.

So, let me first define this is Fourier transform of 2 n, so now what is a prime as we had written earlier where a prime is a 0 through a n minus 1 padded with n 0's, b prime is similarly, so these are n 0's. This operation notice that, this is the vector of size 2 n, so this indicates that you multiply first component of this with the first component of this. Make that as the first component of the result. Second of this, second of this multiply them that will be the second component, so the encircled dot is component to component multiplication. Hence, this whole product will be a vector of size 2 n. Therefore, we can compute inverse Fourier transform of such an entity and this whole thing, we will show is nothing but the convolution of the two input vectors.