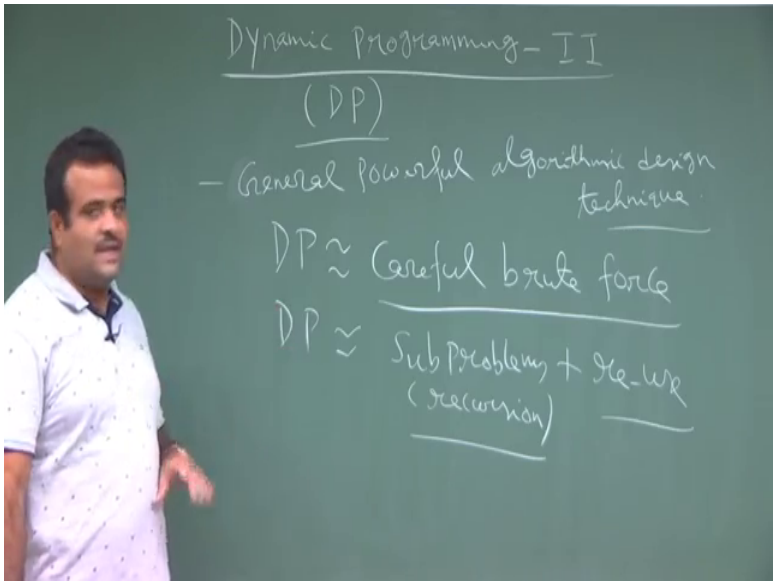
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture - 57
More On Dynamic Programming

So, we talk about dynamic programming which we have seen earlier also, you have started. So, we will talk about more on dynamic programming.

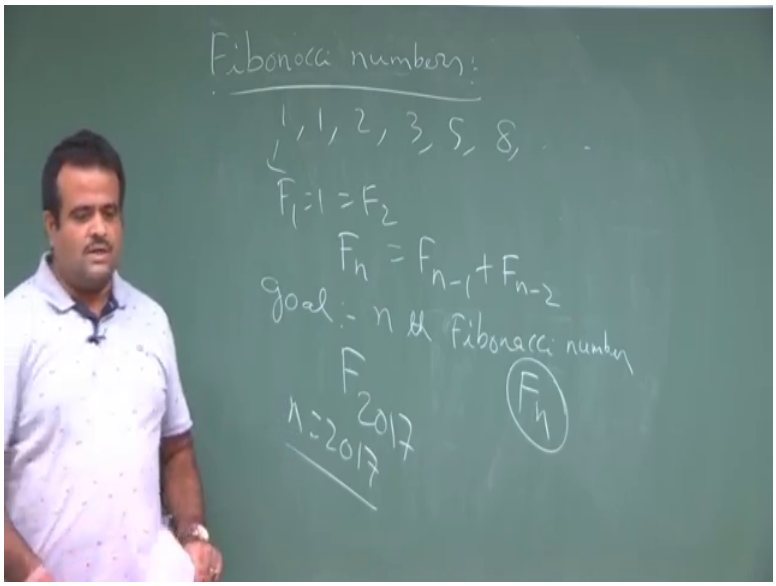
(Refer Slide Time: 00:35)



So, it is very exciting topic. So, dynamic programming in short it is called DP. So, it is basically a very powerful general powerful design technique. So, I mean. So, it is basically a powerful general powerful design techniques to solve the problems algorithmic design technique. So, basically it is a careful brute force. So, DP is basically careful brute force or exhaustive search. So, in exhaustive search the time is exponential. So, we look at the all possible solutions. So, that is called exhaustive search or brute force. So, that is the exponential algorithm, but we reduce that exponential space to the polynomial space carefully. So, that is the technique of dynamic programming. So, it is a basically; we reduce this exponential space to polynomial space. So, that is the idea.

So, it is specially good for kind of optimization algorithm like if there is a (Refer Time: 02:13). So, dynamic programming is very much useful for optimization algorithm. So, it is basically. So, we have a problem we reduce this problem in a sub problems. So, by recursively; so, this is basically sub problems we have to get the sub problems and then we reuse the sub problem. So, this is by recursively. So, this is the recursion. So, recursion and then we reuse this sub problems to get the; I mean we reuse the value of we reuse this value in the whole problem. So, this is the idea. So, basically we discuss this through some example like problem like finding the Fibonacci number and then the shortest path problem. So, let us just start with the Fibonacci number problem. So, finding the nth Fibonacci number; so, these we have discussed already in the beginning of few beginning of our; this course like when you start the divide and conquer approach.

(Refer Slide Time: 03:47)

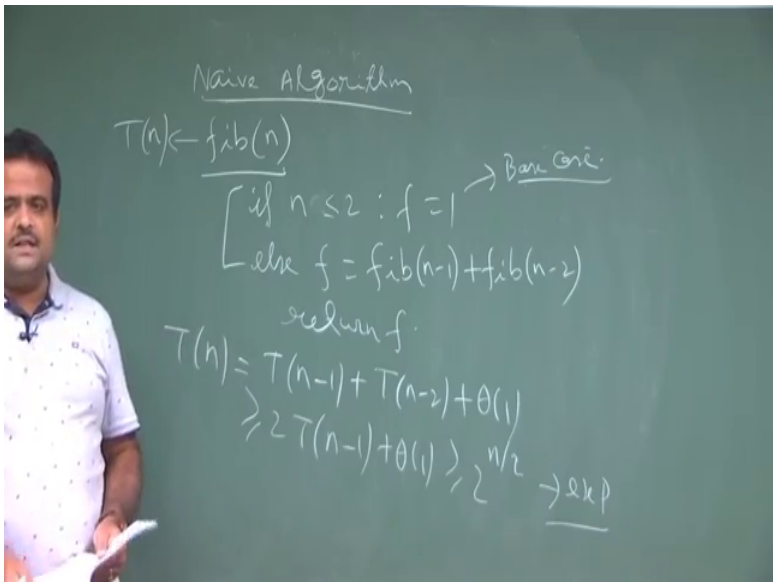


So, there we have discussed this. So, just to bring this problem and we will see how we can solve this using a dynamic programming technique.

So, here is the definition of the Fibonacci number problem, Fibonacci numbers. So, what are the Fibonacci numbers basically you start with one; 1, then 2, then 3, then 5, then 8. So, sum of previous 2. So, this is basically it start with say F one is one which is same as F 2 and then F n is nothing, but F n minus 1 plus F n minus two. So, this is the formula.

So, the our goal is to find the problem is to find the nth Fibonacci number nth Fibonacci number that is F n. So, this is the problem we have to find F n. So, n is an input say we have to find F 2 thousand seventeen. So, n is 2 thousand seventeen. So, this is the problem. So, finding the nth Fibonacci number. So, what is the naive of algorithm to solve this problem?

(Refer Slide Time: 05:22)

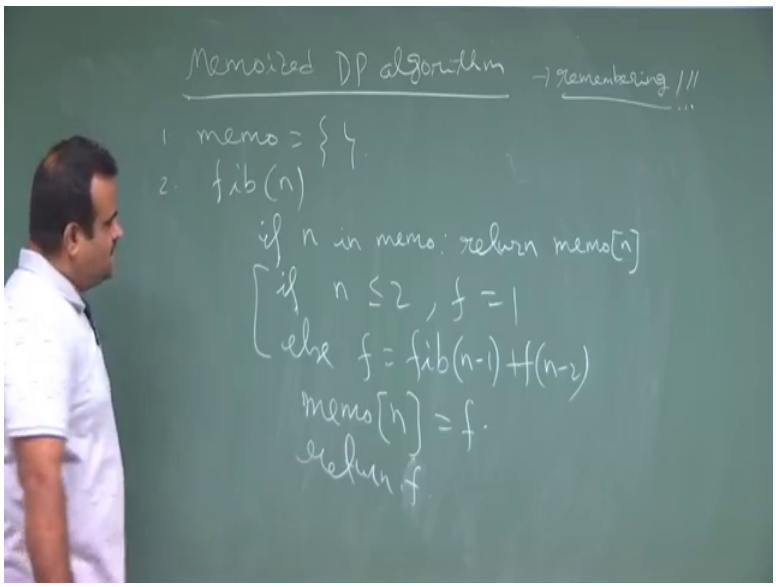


So, what is the correct algorithm to solve this problem? So, if you use this recurrence. So, you have to use this formula. So, this is the naive algorithm.

So, we have to follow the recursive formula like. So, suppose you want to find this is the call, now if n is less than 2 we return 1. So, this is the base case otherwise else. So, else what we do we compute f of fib of n minus 1 last fib of n minus 2 and we return f. So, this is the correct algorithm. So, to compute the f n nth Fibonacci number; so, this is the. So, this is the recurrence, this is the recurrence, right. So, this is coming from the definition of the f n. So, this is the recursive definition and this is what is called base case. So, now, what is the time complexity of this algorithm or of this code? So, time complexity this is exponential algorithm why because if T n is the time to time for this then T n is basically we have 2 call T n minus 1 plus T n minus 2 plus theta 1.

Now, the why this is exponential this is basically we can write as greater than equal to 2 T n minus 1 plus theta of 1. So, this will again give us greater than 2 to the power n by 2. So, this is basically exponential algorithm. So, now, this is a exponential time algorithm. Now you want to use the technique which is called dynamic programming technique to reduce this exponential algorithm to the polynomial time algorithm like linear algorithm. So, we will use 2 techniques to do that one is memorization; memoize DP algorithm another one is bottom of DP algorithm. So, first let us start with memoized DP l because this is exponential exponential is bad. So, you want to reduce this we want to use the DP technique to reduce this time complexity from exponential to polynomial.

(Refer Slide Time: 08:31)



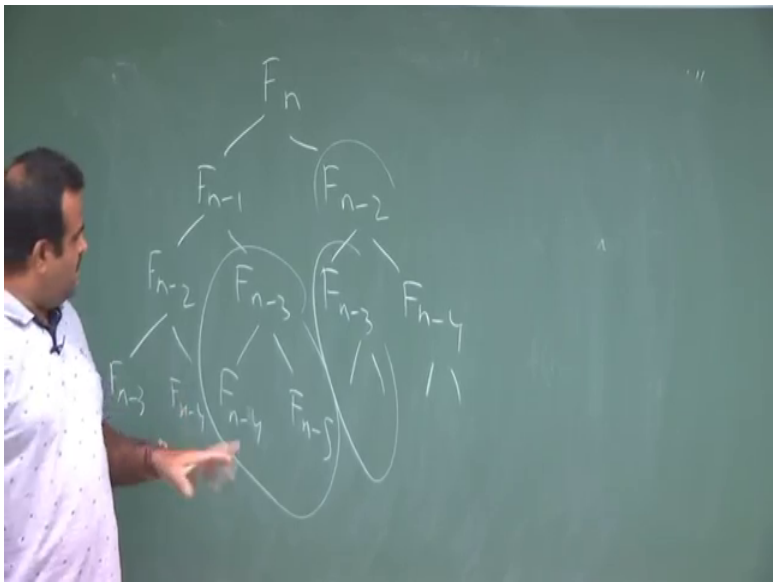
So, this is the first technique we will use memoized DP algorithm. So, memoize mean we will remember once we compute a value we remember. So, we will use a dictionary we will put the values over there, so that we do not need to compute it again. So, that is the memoization remembering. So, we remember the value. So, this is basically remembering. So, we remember the value. So, what is the how we remember we use a dictionary once we compute the value we put it into dictionary.

Now, when we now we look at the dictionary they if the value is there then we will get that otherwise we have to compute it and once we compute it we put it into the

dictionary. So, that is the memo memoize that is the remembering. So, suppose this is our dictionary which is empty initially and say this is the code and now we are computing fib of n now if n is not in the dictionary if n is in memo, then we return memo n. So, this is just a kind of table lookup we do what else if; then the code is similar to earlier is the recursive call else if n is less than 2 then F is equal to 1 else we compute we call fib of n minus 1 F of minus 2 and then we put this into the memo and the return F we return f. So, this is the recursive step same as earlier, but here we are just look at the table we had just look at the dictionary. So, if that is in the table then we are just we are just not computing that otherwise we are computing that.

So, this we can this type of technique we can use for any recursive algorithm this kind of. So, let us look at the recursive tree for this code. So, so suppose we want to compute F of n.

(Refer Slide Time: 11:41)

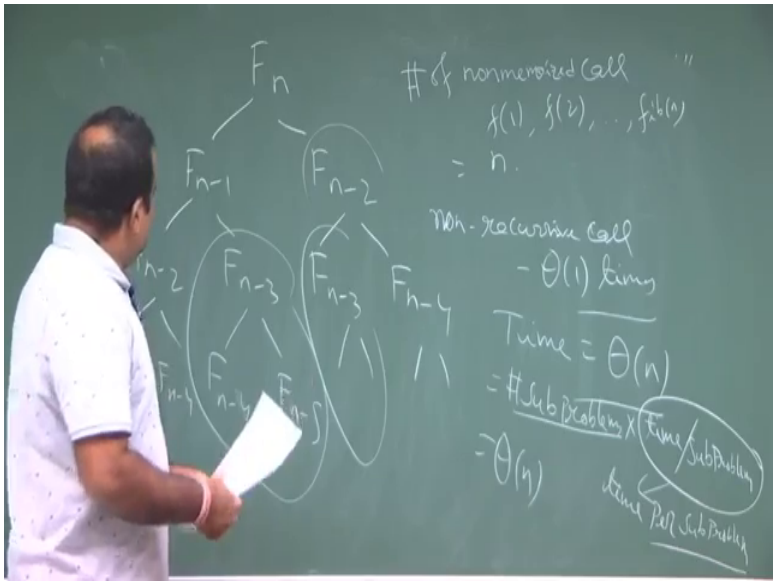


So, to compute F of n we need to compute F of n minus 1 F of n minus 2. So, in order to compute F of n minus 1 we need to compute F of n minus 2 F of n minus 3. So, here also we need to compute F of n minus 3 F of n minus 4. So, like this again we need to compute F of say F of n minus 2 means F of n minus 3 F of n minus 4 F of n minus 4 F of n minus F of n minus sorry F of n minus, minus 1. So, F of n minus 4 F of n minus 5

like this. So, this is the exponential time algorithm. Now here if we look at this tree and these trees are same and this tree and this tree are same. So, there are many sub problems are going on. So, this is one of the hallmark of dynamic programming technique dynamic programming problem. So, if you are many sub problems. So, by once you calculate this because here we are again calculating this once we calculate this why to calculate it again.

So, to avoid that; we are going to memorize that we are going to remembering that value. So, we are going to store into the table. So, that is the idea. So, this way we are saving the time otherwise this will be exponential. So, this is the technique. So, we are memoizing we are storing this once we calculate this we are storing into the dictionary now once we are going to calculate these.

(Refer Slide Time: 13:34)

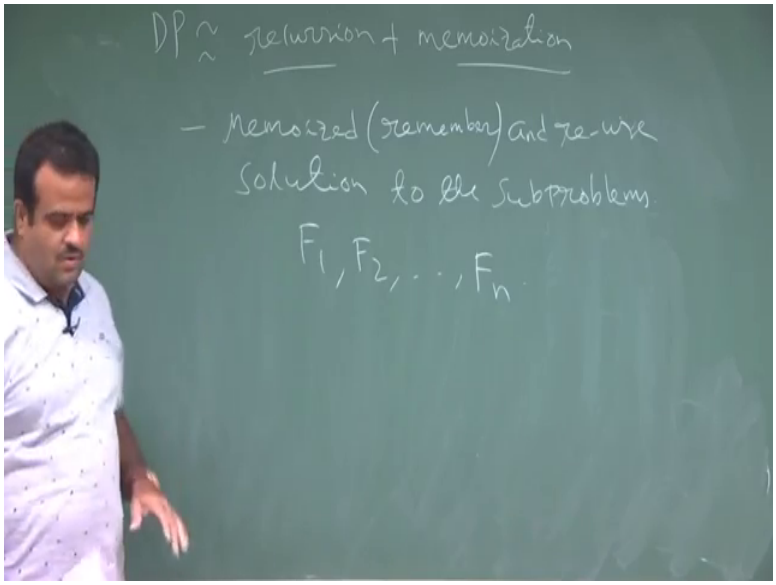


We look at the dictionary since we got it will not execute this calculation. So, that way we are saving the time. So, basically how many non memoized call you have to do. So, so number of non-memoize. So, number of non number of non memoized call is basically. So, we have to calculate this F one F 2 up to I mean fib n. So, n times the number of non memoize calculation this is n times. Now for memoize calculation. So, so that is the non recurrence. So, non recurrence; non recursive call non recursive call is

basically takes theta one times because we have we have already calculated this. So, now, what is the total number of what is the time complexity time complexity is basically theta of n. So, time complexity is basically theta of n. So, this is basically is telling us this is basically time is basically number of sub problems into the time to solve each sub problems this is the amortized analysis because few sub problems we are calculating that by adding, but few sub problems we are getting from the look of. So, that is the way. So, basically we are just adding this 2.

So, this is basically time divided by sub problems. So, this is this is basically time per sub problems and time per sub problems is theta one because we are just looking at the table to get the value and this is n. So, that is why it is coming to be theta of n n into theta one. So, this is basically theta of n. So, this is the theta of n times algorithm for solving the; to getting the Fibonacci number. So, now, we look at another approach which is instead of memori instead of putting everything into a dictionary we look up what is called bottom up DP algorithm.

(Refer Slide Time: 16:46)

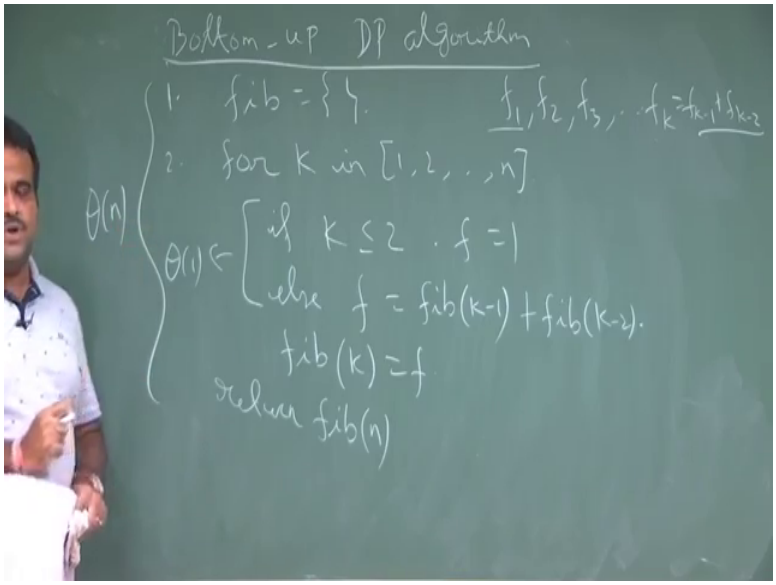


So, this is basically; this is basically bottom up DP algorithm. So, let us just before that let us. So, DP is nothing, but recursion plus memoization. So, basically idea is we memoized or remember or remember we remember the value which you have already

calculated and we reuse the and reuse the solution to the sub problems. So, that is the to the sub problem. So, basically, what are the sub problems in Fibonacci numbers? So, this is basically F one F 2 up to F n. So, these are basically our sub problems. So, there are n sub problems and to solve each sub problem it is taking constant time because we are just looking at the table and then getting the value we are adding. So, theta one time and there are n sub problems. So, that is why it is theta of n. So, this is the linear time algorithm.

So, now we look up the another approach another DP approach which is called bottom up DP algorithm where we just store the last 2 value instead of putting everything into dictionary because we 2 just need the F of n minus 1 and F of n minus 2, we do not need to store the whole dictionary.

(Refer Slide Time: 18:39)



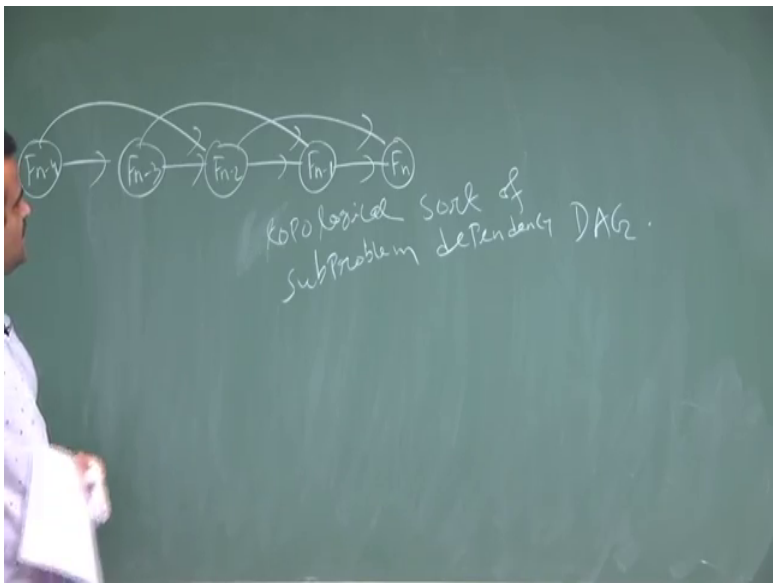
So, that is the idea. So, this is called bottom up DP algorithm. So, what we are doing here this is the code. So, we have this empty set now for k in the; this is the range of the k. So, range of the k is basically 1 to n, we are calculating F n. So, our range is 1 to n. So, now, we have the recursion that recursive formula less than equal to 2 F equal to 1 else F is f of fib of k minus 1 plus fib of k minus 2 and then fib of k is basically F and we return we return fib of n. So, this is basically. So, this part it is basically theta of one time. So, this

is this is basically we are just storing last 2 value. So, this is the bottom up way. So, we start calculating F (Refer Time: 20:28). So, we basically start calculating F one F 2 then F 3 F four. So, always we store the last 2 value like this.

So, we keep on calculating this way and we ultimately we return this. So, this is the; so, bottom up way. So, we start with F 1, then F 2, then F 3; when you call F 3, we forgot F 1. So, this way; so, F k; F k is basically F k minus 1 plus F of k minus 2 this way. So, every time we just store last 2 value of the Fibonacci number and then we add it up to get the next value; so, when you stop we stop at F n. So, this is the linear this is basically sorry this is theta of n times algorithm this is the same time complexity as the memoize DP algorithm, but this has the advantage that the space complexity is less because it has we are. So, for memoize we are putting everything into the dictionary. So, that size is that is less. So, exactly same competition, but less space; so, this is practically faster because there is no recurrence.

Now, we want to see some topological ordering for this computation of F n. So, like this is a F n.

(Refer Slide Time: 22:02)

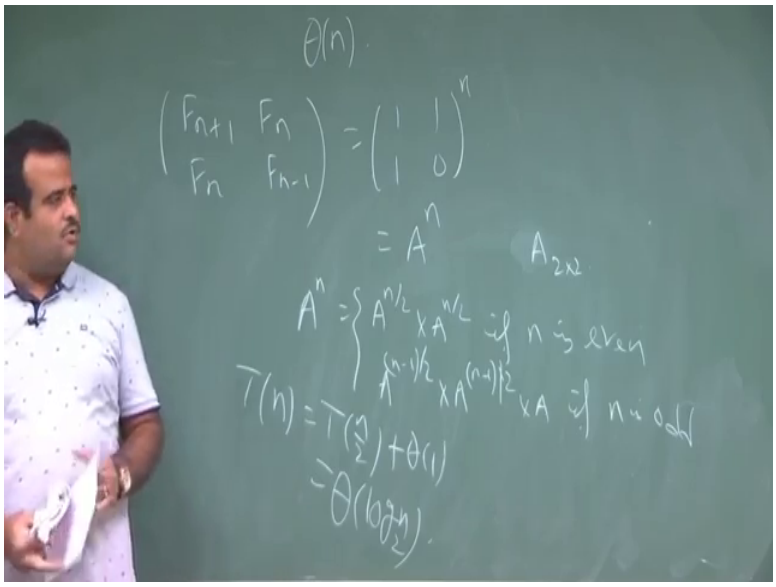


So, to calculate F n, we need to calculate F of n minus 1 F of n minus 2. So, F of n minus

1 F of n minus 2 now to calculate F of n minus 1 I need to get F of n minus 2 and F of n minus 3. So, and then to calculate F of n minus 2 we need to get F of n minus 3 and F of n minus four. So, this way dot dot dot. So, this is the; this is basically a topological sort sorry this is basically a topological sort of sub problems. So, these are the sub problems sub problems dependency graph dependency a I cyclic graph. So, so this is basically this is basically the DP approach for this problem. So, this is giving us this. So, our naive approach was exponential, but we reduce this into polynomial (Refer Time: 23:32) algorithm. So, this is basically linear term T turbine.

Now, we have seen this problem can be solved in log in who can remember this problem of finding the Fibonacci nth Fibonacci number; so, how to how to get nth Fibonacci number. So, we have seen this in the divide and conquer technique in the earlier beginning of the course.

(Refer Slide Time: 24:07)



So, we have seen this F of n last one F of n; F of n; F of n minus 1, this is this value 1 1 1 0 to the power n this kind of formula we have seen. So, this is basically kind of powering a number powering a matrix, but this matrix is a constant matrix. So, once you have a constant matrix. So, this is some sort of A to the power n, but A is a 2 by 2 matrix; just 2 by 2 matrix. So, it is kind of same as powering a number. So, it is basically we have the divide and conquer formula for this; this is basically A to the power n by 2 into A to the

power n by 2 if n is even otherwise A to the power n minus 1 by 2 into A to the power n by 2 into A; if n is odd.

So, this is the divide and conquer steps. So, we have a problem of size n we reduce this problem into sub problems of lesser size and then what is the recurrence; recurrence is basically T n is equal to. So, only sub problem T n by 2 last theta of 1. So, this will give us theta of log n. So, this we have seen in the divide and conquer technique class, but if we use the DP we can solve it using. So, DP is a general technique where we can reduce a exponential time algorithm to a polynomial time algorithm. So, that we will use for many, many, many problem many optimization problem can be reduced in the DP technique.

So, in the next class, we will discuss another problem which is called shortest path problem and we will see how we can use the dynamic programming approach to finding the single source shortest path.

Thank you.