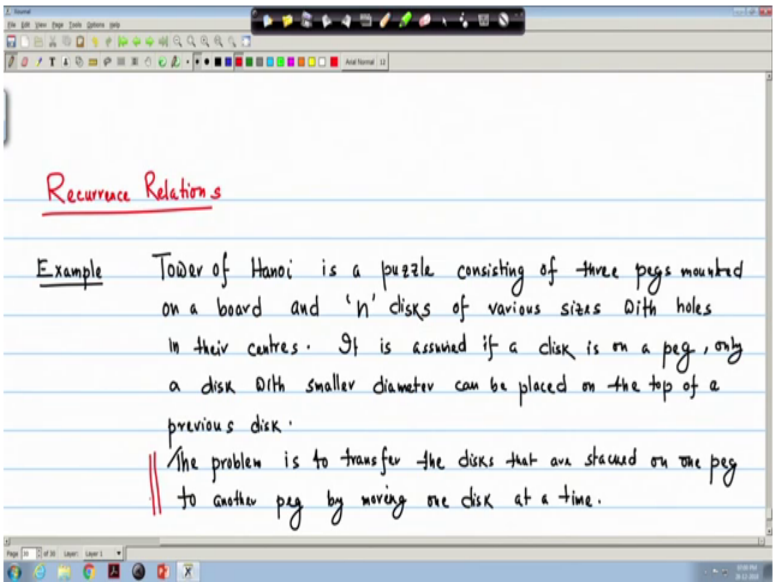
Discrete Structures
Prof. Dipanwita Roychoudhury
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 32
Recurrence Relations (Contd.)

So, we are discussing about the Recurrence Relation and mainly the how to design the recurrence relation. Now, if we remember that when we have read the recursive algorithm then some examples, we have seen mainly the Fibonacci sequence that if we write the recursive algorithm to compute the nth Fibonacci number, we have seen that the recursive algorithm is not efficient because, it took much more number of additions than the iterative algorithm.

But there were many examples even the recursive algorithm is not efficient, then also we use recursive algorithm or recursion. Since, there are many problems where we can very simple way, we can give a solution procedure; that means, we can give algorithm; that means, the recursive algorithm is much easier to solve the problem. And we will see today this type of problems. So, first we read our tower of Hanoi problems.

(Refer Slide Time: 01:59)


Recurrence Relations

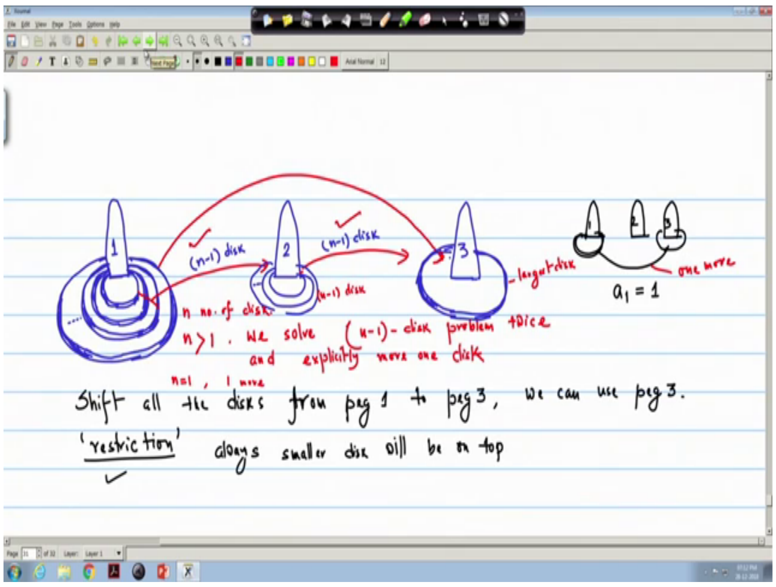
Example Tower of Hanoi is a puzzle consisting of three pegs mounted on a board and 'n' disks of various sizes with holes in their centres. It is assumed if a disk is on a peg, only a disk with smaller diameter can be placed on the top of a previous disk.

The problem is to transfer the disks that are stacked on one peg to another peg by moving one disk at a time.

So again, we continue the recurrence relations and first we see some of the problems where, how we can frame the recurrence relation then one such example is this is the tower of Hanoi. So, tower of Hanoi is a puzzle first, we explain the problem. So, it is a puzzle consisting of 3 pegs mounted on import. And n number of disks of different sizes of sizes with holes in the center.

Now, it is assumed there is some restriction of placing the disks on the peg, that what is that restriction that it is assumed that if a disk is on a peg, only a disk with smaller diameter can lie can be placed on it on the top of the previous disk; that means, always the smaller disk will be on the top. Now the problem is to transfer the disks that are stacked on one peg to another peg by moving one disk at a time and we can use the third disk for this purpose ok. So, this is my the problem is this. So, first we will frame a recurrence relation for this then say this is one puzzle and solving this puzzle is very difficult if when n is very large.

(Refer Slide Time: 07:33)



$n>1$. We solve $(n-1)$-disk problem twice and explicitly move one disk

$n=1$, 1 move

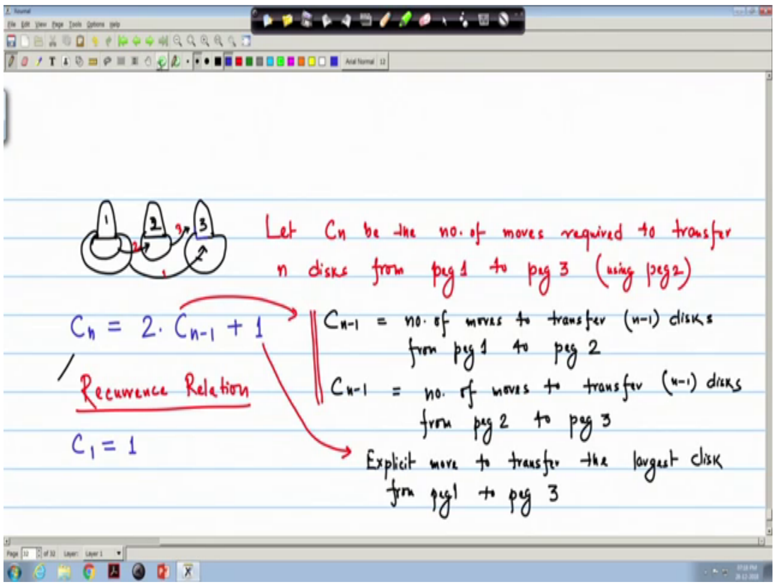
Shift all the disks from peg 1 to peg 3, we can use peg 3.

'restriction' always smaller disk will be on top

So, if I again explain properly, say I have 3 peg say, this is first peg second peg, third peg and in the first peg the n number of disks are there. So, in this way I can draw you say, I have to shift these all these disks from the problem is to shift or transfer all the disks to from peg 1 to peg 3. And, we can use peg 2 and the restriction is I should tell the restriction that always lower disks or smaller disk having disk with smaller diameter.

So, you can I can tell smaller disks, smaller disk will be on top even during the transfer, we must follow this restriction. We must follow this restriction you see if I have only 1 peg, if I have only 1 peg then what I will do that on simply, I will say, I have only 1 peg then simply, I will transfer this thing to peg 3. So, I need only 1 move ok. So, if I write then number of moves, it is 1, but if I have even if I have to then also I have to first I have to if.

(Refer Slide Time: 12:05)



Let C_n be the no. of moves required to transfer n disks from peg 1 to peg 3 (using peg 2)

$C_n = 2 \cdot C_{n-1} + 1$

Recurrence Relation

$C_1 = 1$

C_{n-1} = no. of moves to transfer $(n-1)$ disks from peg 1 to peg 2

C_{n-1} = no. of moves to transfer $(n-1)$ disks from peg 2 to peg 3

Explicit move to transfer the largest disk from peg 1 to peg 3

Now, say I have say I have 3 pegs. So, I have to then first I remove the larger one 2 3 then the smaller one was the larger one here then this smaller one into 2 then the smaller one from 2 to 3. So, I need 1 move then it is 2 move then it is 3. So, I have 1 move first here 2 here 3 move. So now, when the number of disks are small numbers say 1 disk, 2 disk, 3 disk then in this way, we can shift that thing, we can transfer all the disks from one peg to another. But, if n is a very large number then it is very difficult to write this type of probe and if we want to write a program then it is very difficult to write that thing then first thing is that how to solve or what will be the number of moves required, if all the disk I want to shift from 1 to 3.

So, what will be the output? So, to get a solution for this type of problem actually recursive approach is very easy to understand and to represent, what will be doing? So, our approach should be that if I have n number of n number of disks and already we have seen that form on disk, it is very easy when in greater than 1, we solve in 1 disk problem, what is that thing as if the we will transfer the largest disk here.

So, I need 1 1 move for this and these remaining n minus 1 disks remaining n minus 1 disk, I give this number to shift. So, this is my 4 n minus 1 disk, I will give and then I have for this, I have a larger disk. Here says, if I have a larger disk the and then I will ship this n minus 1 disk to here. So, again this is some n minus 1 disk moved to here then all n disk should be should are shifted from 1 to 3. So, here actually are we get giving yet as if these are my these are my n minus 1 disk and this is my the largest disk that was at the bottom of the peg 1. So, we solve n minus 1 disk towards here 1 and here 1.

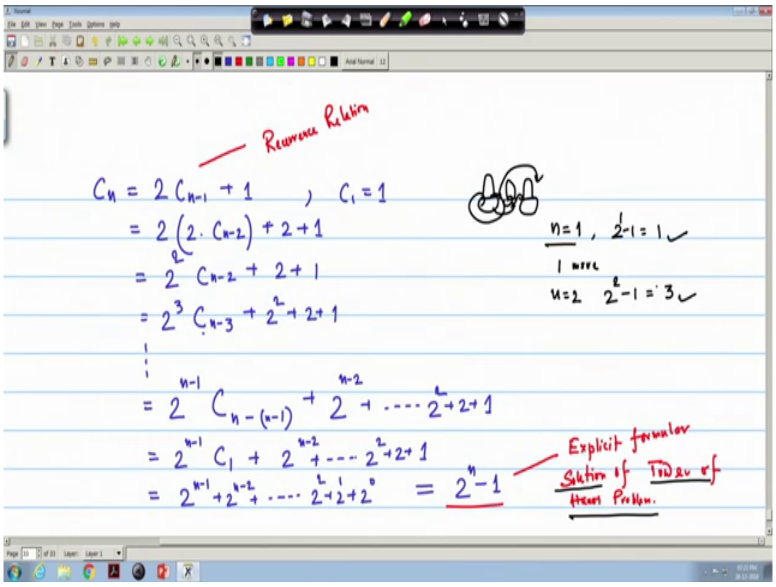
So, twice and explicitly move 1 disk that is the largest disk. So, this will give me the solution. So, what is a recursive approach that we know that if n equal to 1 for n equal n equal to 1, I need only 1 move that already I have explained here, I need only 1 move and our approach is it is a recursive way of solving or recursive algorithm that as if we solve n minus 1. This problem twice that means, in my the largest is first moved to or since at the top the first n minus 1 disks are moved from 1 to 2. Then the largest disk is moved to 1 to 3 and the n minus disks are moved from 1 to 2 and the n minus 1 disks are moved to 2 to 3 then we will be getting the solution that all n disks are moved from 1 to 3.

So, this is the our approach to solve the problem it is I think it is very much clear that if n is very large even, if it is n is 10 then it is very difficult to move this type of thing rather as if we assume that that when I have 10 number of disk as if the smaller number of this say for 9, it is already solved; that means, we can move in this way we can do that thing. So now, how we can give the recurrence relation for this problem? So, the recurrence relation that just now what I told that we have to solve n minus 1, this problem to us. So, the solution is if we represent let C n be the number of moves required to transfer n disks from peg 1 to peg 3 using peg 2.

So now, we can write the way we have given the solution that C n is twice, we solved the n minus 1 to C n minus 1; that means, and 1 explicit move, what is this 2 C n minus 1? This 1 C n minus 1, you can write 1 C n minus 1 is the number of moves to transfer n minus 1 disks from peg 1 to peg 2 and another C n minus 1 is number of moves to transfer n minus 1 disks from peg 2 to peg 3. So, I can tell this is my 2 C n minus 1, I can tell this is my 2 C n minus 1 and this 1, this 1 is for move this is my explicit move to transfer the largest disk from peg 1 to peg 3. So, this is one rick this is my recurrence relation for tower of Hanoi problem.

So, this is my recurrence relation. And it is very much clear that much easier way, we can represent this thing then once I get this thing and since I know the initial conditions also what are my initial conditions, what is my C 1, if I have only 1 peg? Number of moves required is 1. So, C 1 is 1, now I can even I can get a explicit formula for this.

(Refer Slide Time: 24:53)



Recurrence Relation

$$C_n = 2C_{n-1} + 1 \quad , \quad C_1 = 1$$
$$= 2(2 \cdot C_{n-2}) + 2 + 1$$
$$= 2^2 C_{n-2} + 2 + 1$$
$$= 2^3 C_{n-3} + 2^2 + 2 + 1$$
$$\vdots$$
$$= 2^{n-1} C_{n-(n-1)} + 2^{n-2} + \cdots 2^2 + 2 + 1$$
$$= 2^{n-1} C_1 + 2^{n-2} + \cdots 2^2 + 2 + 1$$
$$= 2^{n-1} + 2^{n-2} + \cdots 2^2 + 2^1 + 2^0 = \underline{2^n - 1}$$

$n = 1$, $2^1 - 1 = 1$ ✓

1 move

$n = 2$ $2^2 - 1 = 3$ ✓

Explicit formular

Solution of Tower of Hanoi Problem.

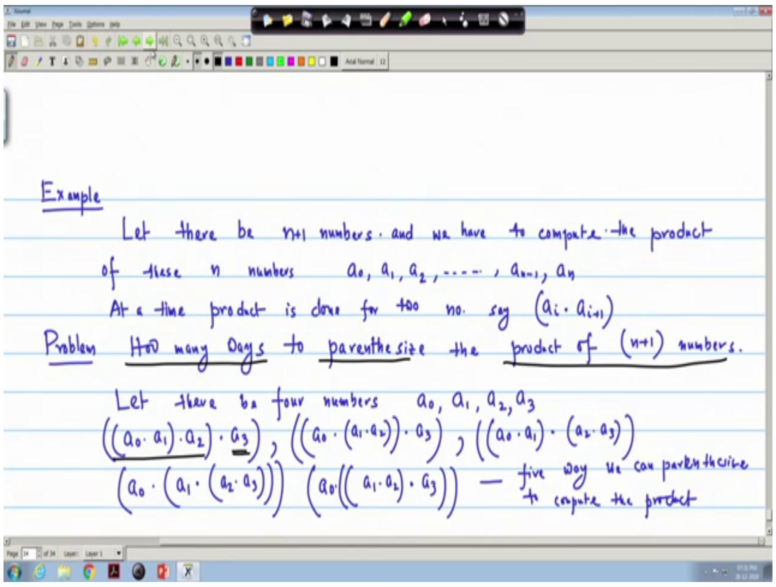
So, I know the recurrence relation C n is 2 C n minus 1 plus 1 with C 1 equal to 1 then I can write 2 to C n minus 2 plus 1. So, plus 2 plus 1 then again this is to square C n minus 2 plus 2 plus 1 then it is 2 q C n minus 3 plus 2 square plus 2 plus 1. So, in this way if we proceed then we will be getting 2 to the power n minus 1 C n minus n minus 1 plus 2 to the power n minus 2 plus 2 square plus 2 plus 1, because when we are getting 2 cube it is C n minus 3 and it is 2 square. So, it is I am putting as n minus 1. So, this becomes n minus n minus 1.

So, this becomes 2 to the power n minus 1 C 1 plus 2 to the power n minus 2 up to 2 square plus 2 plus 1. So, this becomes since this is C 1 equal to 1. So, this is 2 to the power n minus 1 plus 2 to the power n minus 2 plus 2 square plus 2 plus; that means, it is I am getting that n number of terms and this we know the answer it is 2 to the power n minus 1 by 2 minus 1. So, this is 2 to the power n minus 1, see we have started with the recurrence relation. So, this is my just now, we got the recurrence relation of solving tor of an eye problem, this is my recurrence relation and just if I continue iteratively then we get the solution also and this is my explicit formula for the number of moves to transfer n number of disks from one peg to another.

So, this is the solution for tower of Hanoi problem or you can write that this is my solution of tower of Hanoi problem and this is a solution. So, solution is one that it must satisfy the sequences; that means, if n equal to 1, if I get n equal to 1; that means, only 1 peg then 2 to the power n minus 1 becomes 2 to the power 1 minus 1 that is 1. So, we know that from peg 1 to peg 3 only 1 move is required for n equal to 1, it is only 1 move, say for n equal to 2 then 2 to the power 2 minus 1 is 3 move, we have already seen that if it is a 2, if we remember that so if I have 2 peg.

So 1, first the largest will go here 1 smallest will be go here and then again smallest will go here. So, I have I have 3 moves. So, it satisfies the relation for any number of pegs. So, these will give me the solution this is my solution of tower of Hanoi problem and what you see that it means if I now take the recursive approach or the recursive algorithm then actually, it is much easier to handle this type of problem.

(Refer Slide Time: 30:08)



Example

Let there be n+1 numbers. and we have to compute the product of these n numbers $a_0, a_1, a_2, \ldots, a_{n-1}, a_n$

At a time product is done for two no. say $(a_i \cdot a_{i+1})$

Problem How many ways to parenthesize the product of $(n+1)$ numbers.

Let there be four numbers a_0, a_1, a_2, a_3

$((a_0 \cdot a_1) \cdot a_2) \cdot a_3)$, $((a_0 \cdot (a_1 \cdot a_2)) \cdot a_3)$, $((a_0 \cdot a_1) \cdot (a_2 \cdot a_3))$

$(a_0 \cdot (a_1 \cdot (a_2 \cdot a_3)))$ $(a_0 \cdot ((a_1 \cdot a_2) \cdot a_3))$ — five way we can parenthesize to compute the product

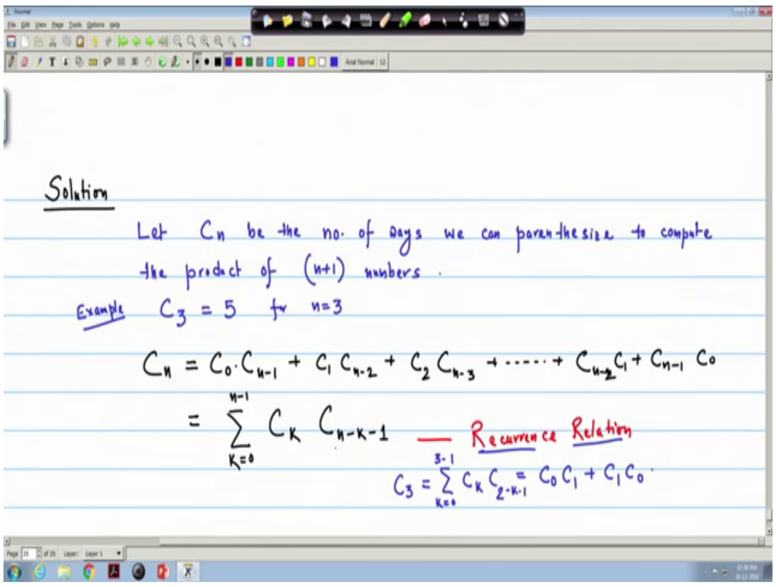
Now, we take another example. Let I have, let there be n numbers then how many ways the problem is that how many ways and we have to first write the problem we have n numbers. And, we have to compute the product of this n numbers see numbers are a 0, a 1, a 2 say n minus 1 and then I take I can take n plus 1 number and I can put here n. Now at the product at a time product is done for 2 numbers, say a i and a i plus 1.

Now the problem is problem that how many ways to parenthesize the product of n plus 1 numbers, we try to explain or with some example let there are that there will be 5 numbers, I will get phone numbers then it will be easier to give that thing numbers say a 0, a 1, a 2, a 3. Now, we can parenthesize to compute the product like I can parenthesize a 0, a 1, then a 2 then a 3 then I can do a 0, a 1, a 2 then a 3, I can take a 0 a 1 a 2 a 3 and then I take the product I can take a 0 then a 1 then a 2, a 3 or I can take a 0, a 1, a 2 then a 3 first and then.

So, these are the 5 ways, I can the 5 ways we can parenthesize to compute the product, when we have phone numbers a 0, a 1, a 2, a three. So, my problem is that how many ways to parenthesize the product of n plus 1 numbers. Now, if we observe that the product or the parenthesising the numbers, when it is only 4 then we see that as if we take that we are solving the problems with a lesser numbers then it is much easier to do their thing, say we take this thing is 0 a 1 a 2.

So, as if a 3 is the number 1 number left and this is that when there are 3 numbers is 0 a 1 into. So, we first complete the product of this 3; that means, it gives that if 3 numbers are there instead of 4 then it gives that how many ways, we can parenthesize and then we add 1.

(Refer Slide Time: 38:13)



Solution

Let C_n be the no. of ways we can parenthesize to compute the product of $(n+1)$ numbers.

Example $C_3 = 5$ for $n=3$

$$C_n = C_0 \cdot C_{n-1} + C_1 C_{n-2} + C_2 C_{n-3} + \cdots + C_{n-2} C_1 + C_{n-1} C_0$$

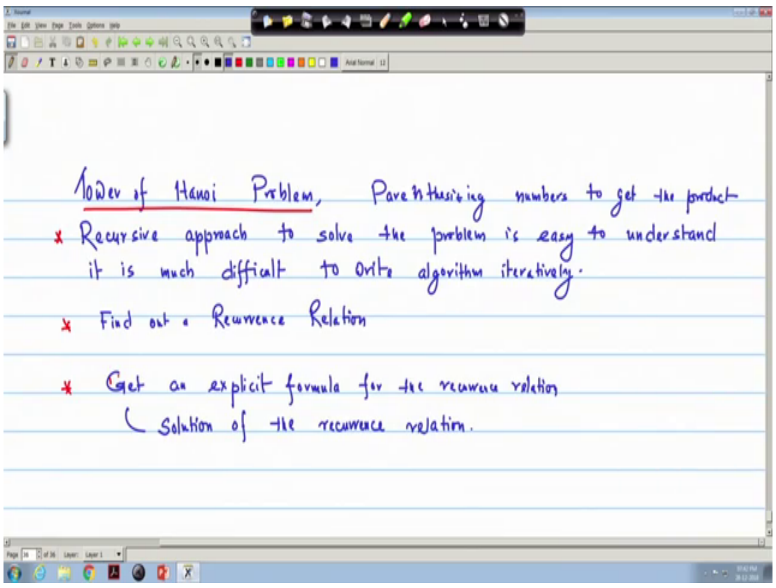
$$= \sum_{k=0}^{n-1} C_k C_{n-k-1} \text{ — Recurrence Relation}$$

$$C_3 = \sum_{k=0}^{3-1} C_k C_{2-k-1} = C_0 C_1 + C_1 C_0.$$

So, if in this way, we can we can do then my solution will be I can write my solution. So, let C n be the number of ways we can parenthesize to compute the product of n plus 1 numbers ok. And just now, we have seen that we have if it is C3 like one example, we have taken that if it is C3 equal to 5 for n equal to 3. So, if some recursive approach, we apply then we can write that we can write C n is C 0 C n minus 1 as if only one element, we have taken and then C 1 C n minus 2 C 2 C n minus 3 then the reverse order n minus 2 C 1 C n minus 1 C 0. Again, we just see the way we have done is 0 a 1, a 2, a 3 and this is the way we have taken it a 0, a 1, a 2, a 3 the reverse thing. Similarly, here it is reverse and this is only one that is 0 1 into a 3.

So, I can write this thing as sum of K equal to 0 to n minus 1 C K and C n minus k minus 1. So, this gives a recurrence relation. So, already we have seen that for n equal to 3 that C3 equal to 5. So, we can put here that n equal to 2. So, what is my C 3? So, my C3 is K equal to 0 to 3 minus 1 is C K C n minus 1 is C2. So, this gives me that is C 0 C n minus 1 and then keep save me, the 10 minus of n minus 1 n minus k minus 1 I write. So, first thing is that K equal to 0 and n equal to 2. So, this becomes 2 minus k minus 1 C 0 C 1 plus C 1 C 0, the C 2 C 0 C 1 plus C 1 C 2. So, we can get in this way, we can actually satisfy that we can that our solution must satisfy the recurrence relation.

(Refer Slide Time: 45:05)



Tower of Hanoi Problem, Parenthesizing numbers to get the product

* Recursive approach to solve the problem is easy to understand it is much difficult to write algorithm iteratively.

* Find out a Recurrence Relation

* Get an explicit formula for the recurrence relation
  (Solution of the recurrence relation.

So, what we have read in this lecture that we see that some of the problems like tower of Hanoi like tower of Hanoi problem or this parenthesizing numbers to get the product, that recursive approach is efficient. In that sense that, recursive approach to get or to solve the problem is easy to understand and even for it is much difficult to write algorithm iteratively.

So, we have seen the that some examples particularly tower of Hanoi problem we have seen. An approaches first will apply some recursive our approach, then we find out the recurrence relation, find out a recurrence relation. We have seen that how to get the recurrence relation and next we will be for some simple examples, we have shown that how to get the explicit formula, which is nothing, but the solution of the recurrence relation.

So, the next is to get the recurrence relation or get an explicit formula for the recurrence relation which is nothing, but the which is the actually the solution of the recurrence relation. So now, we learnt that how to solve the recurrence relation. So, in this class we have read how to frame or how to design the recurrence relation for a given problem. And to get the solution, we will we have to learn that the different techniques that is used to solve the recurrence relation.