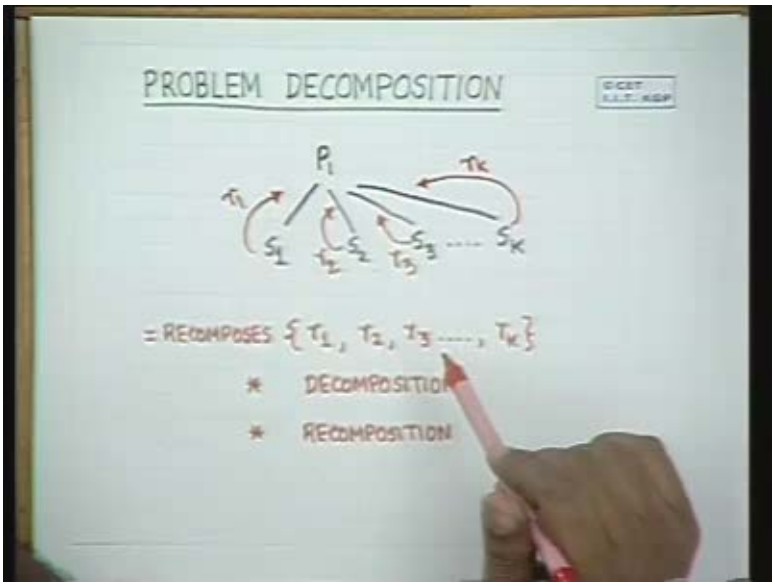
Programming and Data Structure
Lecture # 08
Problem Decomposition by Recursion - I
Dr. P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

The topic of today's lecture is problem decomposition and recursion. As we had mentioned in the first class that for every problem, we need to have a mechanism to solve the problem. To solve the problem means that given an abstract definition of the problem in a language like English, we would somehow have to decompose or convert the problem into a sequence of steps because unless we convert it into a sequence of steps, we cannot solve it on the computer. Secondly, we also saw that when we solve the problem there are many ways to solve a given problem and before going into one algorithm directly we need to look at alternative ways to solve the problem and have and examine each of these alternatives to find out the best possible choice.

Today we will discuss the issue of problem decomposition by induction which is one of the most well studied mechanisms and possibly one of the most well used mechanisms to solve problems at the first level. That is given a problem; problem decomposition means that given a problem we break up the problem into sub problems. So we are given a problem P_1 and we break it up into sub problems S_1 S_2 S_3 and so on up to S_k and solve these sub problems. Now each of these sub problems when solved will give a solution back, result 1, result 2, result 3 and result 4 k's. So given a problem, we break it up into sub problems and these sub problems provide us with results r_1 r_2 r_3 and so on. And the main problem then recomposes or reconstitutes the actual solution to the problem based on the solutions obtained from its sub problems. That is you have two phases, one is the phase of decomposition of a problem into simpler sub problems and second is recomposition of the solution of the simpler sub problems to obtain the solution to the original problem. This is a technique which is used widely to solve a number of problems.

It is also easy to conceive of solutions to problems using problem decomposition because it often brings you the issue of inductive problem solving in which you solve the problem inductively and use the inductive principle to prove the correctness of a solution. Now, instead of going into the formal mechanisms of problem decomposition and recomposition. Our attempts will be to understand how to do it for a large class of problems and how to solve various problems using this technique. And once we know how to do it and how to do it in a wide sense so that is able to capture a set of solution instead of a single solution. Then we will see how given a problem we first get the initial solution by problem decomposition and then we refine the solution to obtain the final algorithm by a process of data structuring.

(Refer Slide Time 05:11 min)



So, programming methodology or algorithm design will consist of generating an initial solution by a mechanism of problem decomposition and then a mechanism of refinement of this initial solution to the final algorithm which will include the definition of data structures. So we will first takes some time to study problem decomposition, we will study through examples first and then generalize and synthesize our ideas and try to bring out methodology to solve problems or to write out good programs in general.

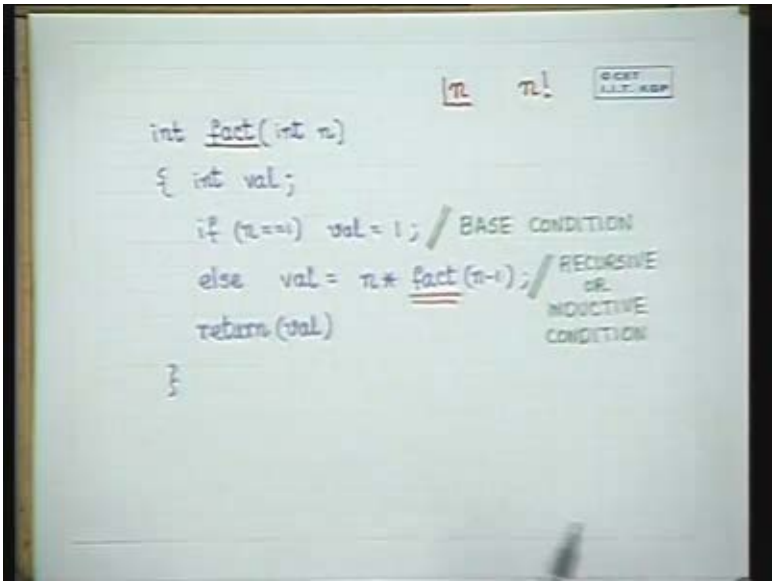
So first our objective is to go through some examples, understand what problem decomposition is about. Here I would like to stress please don't get yourself biased with the programming language when you do problem decomposition. Problem decomposition is independent of a programming language. Programming comes in at a latest stage, you program the solution in a particular language that language may provide you facilities for doing problem decomposition and recomposition and we may use those facilities. But the technique of decomposition and recomposition should be independent of the language in which you are writing the program. So we will try to find out what is meant by that and get a flavor of that, it is very difficult to formalize it at this stage. So we will do it through a sequence of examples. And the first example that we will pick is a problem that we have solved before.

This is the problem to find out the factorial of n, n factorial or n factorial. And we know that to solve n factorial, we had written out the recursive program which was looking like this. We had done this program before, we declared an integer and we wrote a variant of this program. We said that if n is equal to 1, val is equal to 1 otherwise val is equal to n into fact of n minus 1 and then we return the value or val as the value of this function. This was what we did. Now let us have a look at the structure of this definition.

Firstly it is an inductive definition because fact, in fact we are calling fact again but we are obviously calling with a smaller value of n. If you had called it with the larger value of n, this program will terminate. So what did we do with the factorial program? If we look at it in a more

mathematical setting, there are two aspects one is there is an argument n and there are two conditions. If something happens we know, otherwise something. Now this condition, here there is no recursion in this condition whereas here we have the recursive condition. This in all inductive rules are called the base condition because in an inductive definition if you do not have a bases thus inductive definition will not be valid.

(Refer Slide Time 09:33 min)

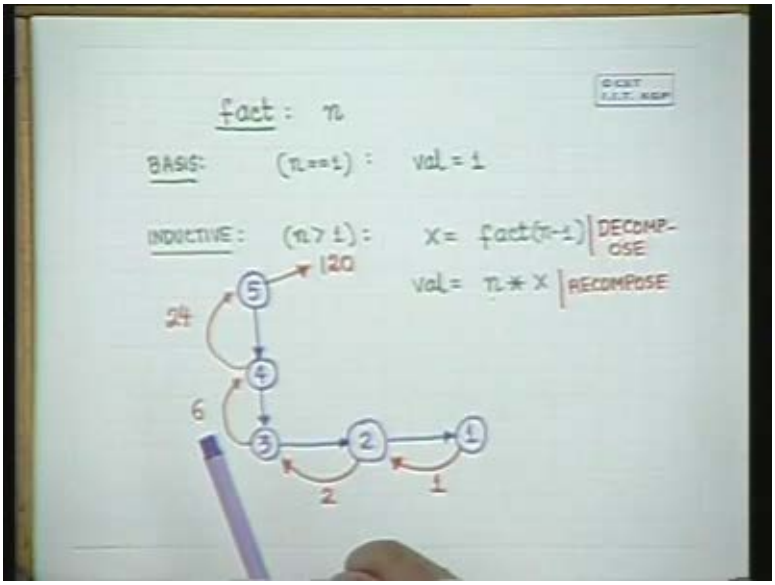


So you must have a basis condition. This is also called the recursion breaking rule when we look at it from a recursive point of view and this is called the recursive or inductive conditions. Now when you have factorial, so what is the decomposition look like? There are two parts, one is the basis part which says n equal to 1 result in value is equal to 1, there is no recursion. The second is the inductive inductive part or the recursive part. The condition here is n is greater than 1 and what happens here. Here if we look back at this val is equal to n into fact of n minus 1, what we did was we first decompose the problem into a single sub problem and solve the single sub problem fact of n minus 1.

Now this sub problem had n minus 1 which was less following the inductive principle. If this is true, if you can solve this and you can obtain the solution from a solution of this, then you have obtained the solution for this provided you have got a basis condition. So suppose we are taken x is equal to fact of n minus 1, this was the problem decomposition, the only and the simplest problem decomposition that we did. And how did we recompose the problem? We recompose the problems in val is equal to multiply n with x. So in the inductive part, if there is a basis part to a problem to a decomposition or to an inductive definition and there is a inductive part or a recursive part in which there are two phases as I mentioned before, one is the decomposition phase, the other is the recomposition phase. That is looking at the solution of the problem; you obtain a solution of the original problem. So given a solution of the sub problems, you obtain a solution of the original problem.

We will go through such examples in details to understand simple things first and then we will be able to handle more complex things in a more comfortable fashion. So, given 5, the basis condition does not, these conditions are mutually exclusive, these conditions cannot overlap. These conditions have to be mutually exclusive. So, only one of the conditions will hold. So based on the condition which holds here, you break it up, this is the diagrammatic representation of what we are going to do. The argument is passed through this then then then, here this condition holds the problem is solved. Once the problem is solved, this value is returned and here 1 is returned that returns the value of x at 2 which does 2 into 1 and returns 2. This does 3 into 2 and returns 6, this return 24 and this return 120. So there is a sequence of decomposition, returning and some recomposition occur here back.

(Refer Slide Time 14:47 min)

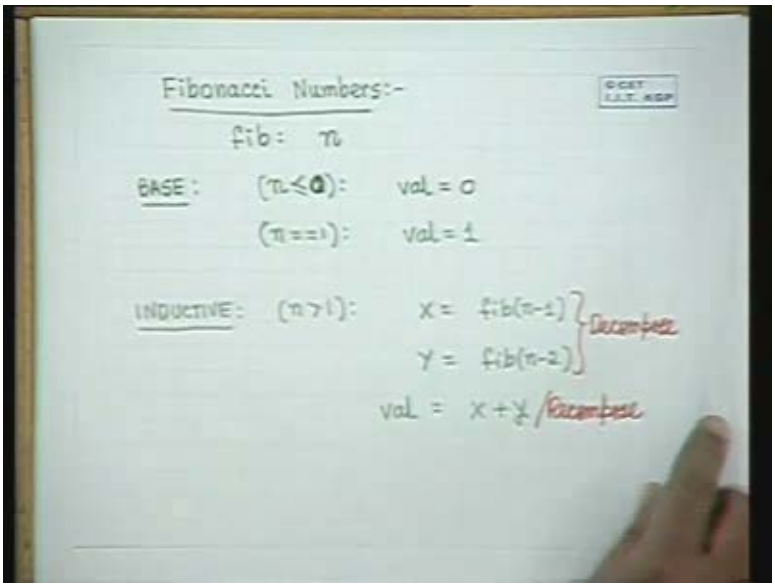


So at a node two things occur, one is decomposition and one is recomposition. So we can look at such problem solving mechanisms in two ways, one is the decomposition phase and a recomposition phase. Though in this particular example we have gone from a program back to its original solution slowly, we shall learn how to solve problems this way first before going into the final algorithm. So this is the simple case in which one single sub problem was taken into account.

Now let's take another problem. We will take another problem for which the inductive, the problem itself is defined by the inductive definition. We know that fib takes an argument n and we have got a basis condition or a base condition as n is less than equal to 1 and we say val equal to 0, let us say this is our condition, sorry val equal to 0 if n is equal to 1, if n is less than equal to 0 and we have got two basis conditions and when n is equal to 1 val is equal to 1. These are our two base conditions and what is the inductive condition? The condition is n is greater than 1 and in order to solve the Fibonacci number problem, you have to obtain f n. It is f n minus 1 plus f n minus 2. So we solve fib of n minus 1 which is one sub problem, we have to solve and get the solution and say x, in y we solve fib of n minus 2. These are our two decomposed problems and

we recompose our sub problem by the addition operation x plus y. This is the complete definition of the Fibonacci number problem. So these are our two basis conditions.

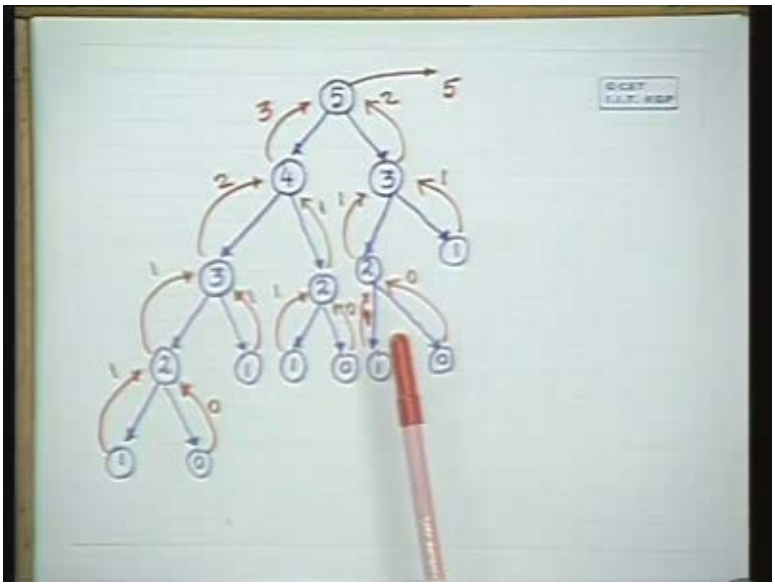
(Refer Slide Time 18:25 min)



Now we have got a problem decomposed into two sub problems and this is the recomposition. So how will it look like in a diagram? Again if we draw say 5, now we have got two sub problems and these two sub problems are actually independence of problems, they are not dependent on each other. When you look at it from the problem solving point of view, no efficiency nothing just whether it gives a correct solution or not. They are absolutely independence sub problems. So you have to solve 4 and you have to solve 3. Now to solve 4, you have to solve 3 and 2. To solve this 3 you have to solve 2 and 1. To solve this one again you have to solve 2 and 1, to solve this 2 you have to solve 1 and 0. To solve this again you have to solve 1 and 0. This is the complete decomposition rule. Well, you can see there are commonalities here and what we will do about them, we will discuss later on. But looking at the inductive definition, we just open up the whole thing and see what this induction, this is the decomposition tree. This is the decomposition tree. This decomposition tree was a simple straight line in the case of permutation problems and this structure has got two children in the case of Fibonacci number problem. And here now how do the values get returned? I am sorry, I have missed out something here, this two.

So this returns 1, this returns 0. To solve this I need to get a value from this. What is the recomposition from here? It's an addition 1 plus 0. This 1 returns 1, here 1 plus 1 this returns 2. To obtain a solution for this, this will return something, 1 this will return 0 say let's draw up all the return links 1 0 1 1 1. This returns 3, this returns 2 and this returns 5. So the decomposition and each of the sub problems returns to the original problem, the solution and the solution is recomposed and then this one again returns it back. This is the way the whole thing goes, alright. So this is the decomposition of the… How we will convert each of them into a program, we will see later on. But this is the way in which the problem is decomposed and this is the way in which the solution is formed.

(Refer Slide Time 23:04 min)



We shall come back to each them to see how we will write a program though we started from problems in which the program can directly be returned but we will now slowly move on to some different problems for which the decomposition itself is more important than writing of the program.

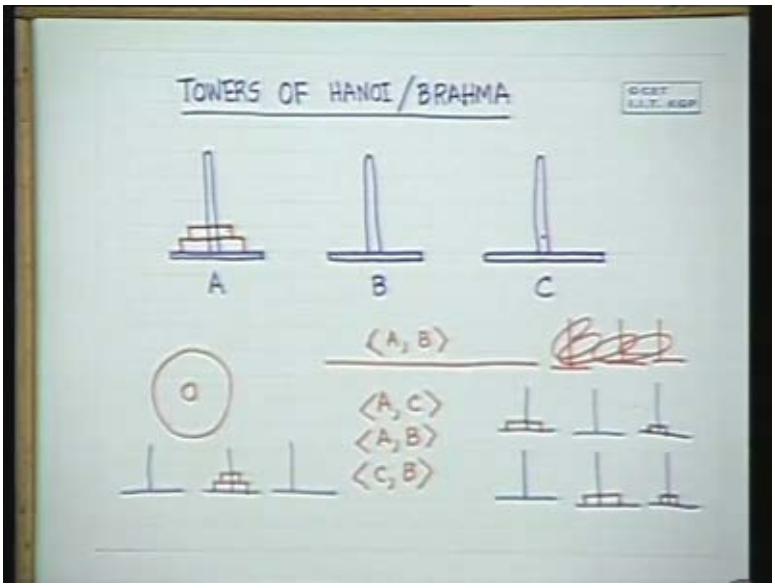
The next problem we will take is one of the most well-known problems; many of you may have done it before, it is also called the tower of, if you do not know the story I will tell it to you offline. It is said that there are… suppose there is a peg and there are three such pegs, you call them A B and C. And suppose I will tell you that there are disks obviously with the holes so that it moves into this peg and the cross sectional view of this disk will be like this. And if I give you one disk and ask you that how will you place this disk onto this, the only with the only move that we are allowed to do is you can move a disk from one peg to another. So moving one disk from A to B is no problem, you just take it out from here and put it in here.

So moving one disk from A to B, the answer is move A to B but suppose I give you two disks with the additional condition that this larger disk, there are two disks of different sizes and here there is a larger disk and there is a smaller disk. The smaller disk is placed on top of the larger disk. Now you ought to move both of them in to a similar configuration into B but never in your sequence of modes will you be allowed to place a smaller disk below a larger disk that is always smaller disk should be above larger disk. Then obviously you cannot place this one here and this one here just above it. So, the solution that you will take up is move the peg which is on top. Obviously you can move only the peg which is on top and you cannot carry together 5 a 6 peg together then could have solved the problem by just pulling all of them up and putting them down.

So you can move them only one by one, so the only solution that you have got is move from A to C then move from A to B and then move from C to B. Is that okay? So when you move from A to C, this peg will come here sorry this peg will come here and you will reach a configuration, I

am sorry the colors are… you will get a configuration in which one was here and the smaller one is here.

(Refer Slide Time 27:39 min)



Then the next A to B say this one will be moved here and you will get the configuration and the last one C to B, this one will move here and you will get the required final configuration. So now we are going to get the problem a little more involved, get little more involved into the problem and say that we have got A B and C and we have got three disks. Now how do we move all three from A to B? With the same rules, that no smaller disk and lower larger disk here to move only one at a time. So let's try out. We can move this one here, A to C. Is this visible? Then we can move out this one here A to B, it is now used moving this one here because that you could have done in the first step only. So we reach a configuration. Now we have only one option moving this here or retracing your steps C to A sorry C to B, you could have also moved C to A but if you move from C to A it could have not led you anywhere because if you move from C to A you would have to move this. You can move from C to A also, you can move from C to B also.

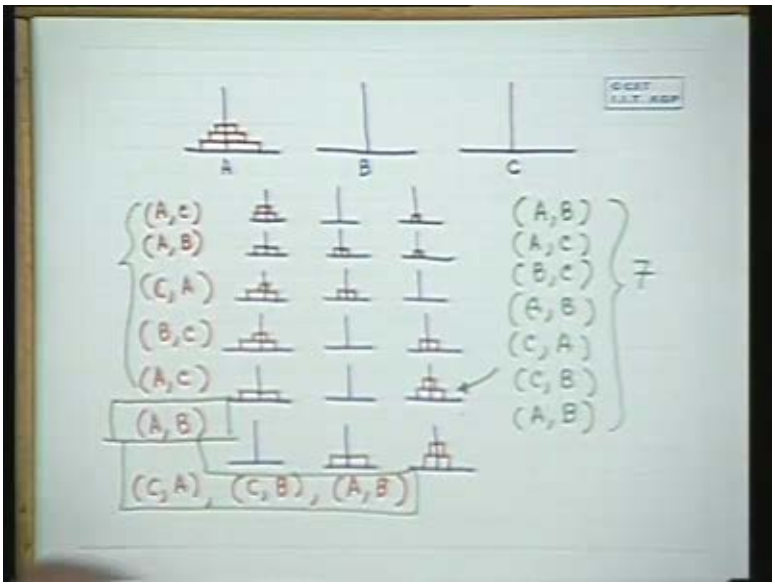
Suppose we take the choice from C to A let's see. Next, any other? Now the only alternative is to move this one here B to C, the next alternative I can move this one here but then I will not be able to move much because I will have to move this one back here because if I move the smallest one here, so the alternative that I have now at my disposal is A to C. So it seems we have reached somewhere because now we can move at least C to its position. We reached somewhere, we have reached the situation where we can move C to its position by a sequence of some trial and error. We went here and there and somehow managed to find the position. Look look, this is now very crucial position because this is the position where I know that if I can get to this position, I can solve the problem isn't it? Because if I move here then I know the sequence of steps because two disks I can move on top of this. That is now if I move from A to B, I have done this but this is very interesting because at this point I know I can forget about this fellow. All I have to do is move these two here which is similar to moving two disks from one peg to another, something of which we have solved before. Isn't it? Because now I need not touch this

one, so I have just to move these two disks back here for which I have done that problem before, I can use the problem of induction and I can straight away write down that to solve this one I will move this one here, this one, next one here and this one back. So this sequence of steps is C to A, C to B and A to B. I hope I do not have to write down these diagrammatically, it's quite clear. So now we have taken 1 2 3 4 5 6 7 8 9 steps to reach this position. If we go back and analyze further and come back and see what we have done then we will see that by a sequence of these 5 steps we have reached this position. Actually we moved in a random fashion, somehow just we moved something here and there whatever we fed but once we reach this position, we knew that we have solved the problem.

Now let's go back and see is this the least number of steps in which I can do this, at least from here to here this is the least number of steps because if this were not the least number of steps then two disk movement problem I could have solved in less than three steps. We saw that moving two disks we could not solve in less than three steps, so we can go back and see whether we can solve this problem faster and to solve this problem faster we have to have a look at what we done. We actually move, we have three disks here isn't it? We actually moved these two disks here, that's all we could have done. We moved the top two disks here, nothing else. And we have required 1 2 3 4 5 steps but to move two disks from one peg to another, we can easily do it in three steps. So this one we can revise and improve our solution up to here, these 6 steps that we have taken, these 5 steps that we have taken, can be improved and the improvement is to bring it from here to here I need to bring this down, this up and this back.

So all I need to do was A to B instead of this A to C which we did A to C now and B to C follow it up with this one and these, A to B, C to A, C to B and A to B. So in 7 steps we can solve the problem for 3 disks so that's well interesting that we have solved it in 7 steps and I leave it to you to try your out to solve this in less than 7 steps. You would have to lift two disks but you can sit on and prove it and since it's just a case of three disks, you can prove it by exhaustive elimination, there is no problem. You can try out all possible combinations of 7 steps and see this is the only one which is the solution.

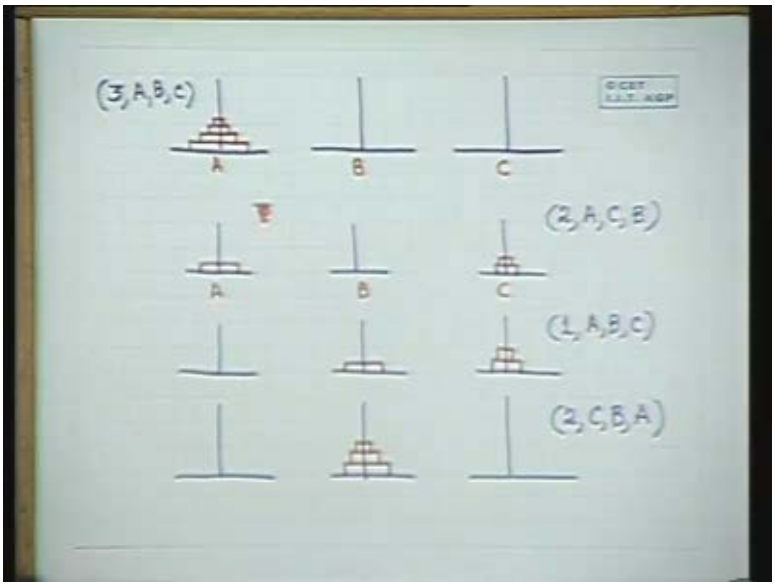
(Refer Slide Time 37:03 min)



But the question, next question is if I give it what have you understood from this problem? We have understood that to solve the three disk problem, we have used the solution of the two disk problem. And how have we used the solution of the two disk problem? To solve the three disk problem from A to B, we have to solve the two disk problems from A to C because the larger disk need not be touched. Then we have moved the larger disk from A to B and then we have moved and then again we have solved the two disk problem from C to B. So we can recollect what we have done about the three disk problem. This was the original position A B C, so to solve the tower of Hanoi three disk problem, we solve we obtain the configuration first like this.

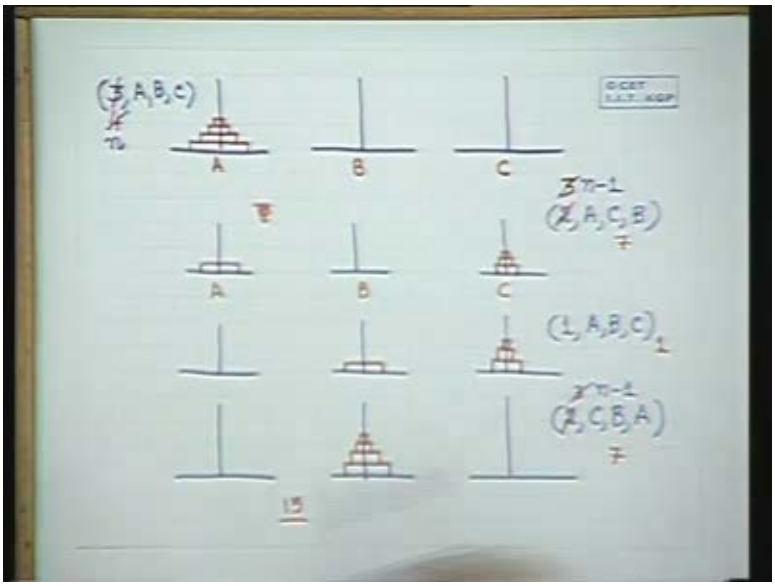
Look now we are not looking at it element by element. Slowly we are looking at it sub problem by sub problem. A, we solve this then the next step that we did was this and the next step that we did was move these two here. So we actually solve it in three steps of which two of them were inductive steps and one of them is a base step because moving this from here to here is the problem of moving a single disk and moving these two from here to here is the problem of moving two disks. So to solve the problem of tower of Hanoi with three disks from A to B using C, we solve the problem of two disks from A to C using B. We solve the problem of one disk from A to B using C and we read, solve the problem of two disks from C to B using A. Is that understood?

(Refer Slide Time 40:48 min)



So now to solve the three disk problem, now in order to solve the four disk problem what do we do? How do we now solve the four disk problem? Do we start like this random fashion? I think we have learned something here to solve the four disk problem, from here we can solve the three disk problem then we can solve the one disk problem and then again we can solve back the three disk problem and the number of steps that we will take will be 7 plus 1 plus 7. So we will take 15 steps to solve this problem. I will leave it to you to think over that this is the minimum number of steps to solve the tower of Hanoi's problem for three steps. So now generalizing this to n is very easy, this is n, this is n minus 1, this is 1 and this is n minus 1.

(Refer Slide Time 42:16 min)



So on the basis of this we have now got at least a solution, proof of minimality of number of steps is left to you, at least we have got a solution to solve the tower of Hanoi's problem. In the next class, we shall come back to this solution and study how, what is meant by the recomposition. Here we have, now we will see how we write it down in form of decomposition and recomposition and then we will see how we convert it into a program.