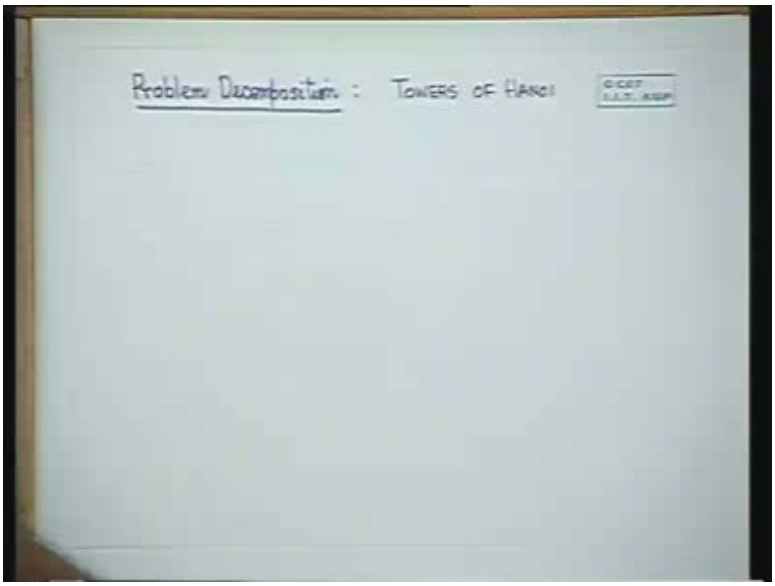
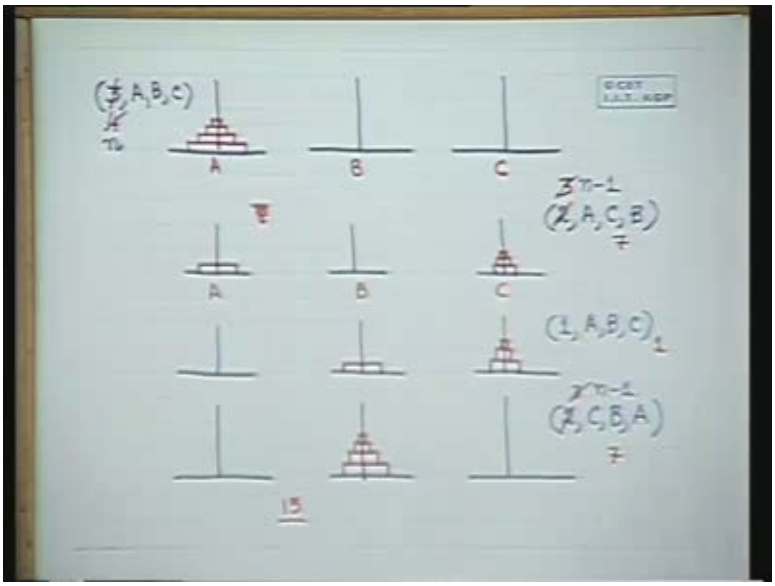
Programming and Data Structure
Dr. P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture # 09
Problem Decomposition by Recursion - II

We will continue our study on problem decomposition and we were solving the tower of Hanoi's problem. And we came to this situation that to solve a problem of n disks from peg A to B using peg C, we have to solve the problem of n minus 1 disks from A to C using peg B and then a problem of one disk that is move element from A to B and then a problem of n minus 1 disks from C to B using A.

(Refer Slide Time 01:07 min)



(Refer Slide Time 01:48 min)



So this is how we decompose the problem into sub problems and for single disk we know the solution directly. So we are now in a position to write out this solution in a decomposition and recomposition format, coming more closure to the steps of problem solving. So we now try our hand at solving the towers problem. Now what are the parameters? n, A, B, C, alright. These are the n pegs. What is the base condition? n is equal to 1, when n is equal to 1 what is the solution. Solution, now there is only. Now what is the solution consist of? The solution as we said, as we showed before consists of a sequence of moves A to B, A to C etc etc. So we have to return a sequence. So let us denote when n is equal to 1 the solution is a single sequence, single move in the sequence, so L is A, B only.

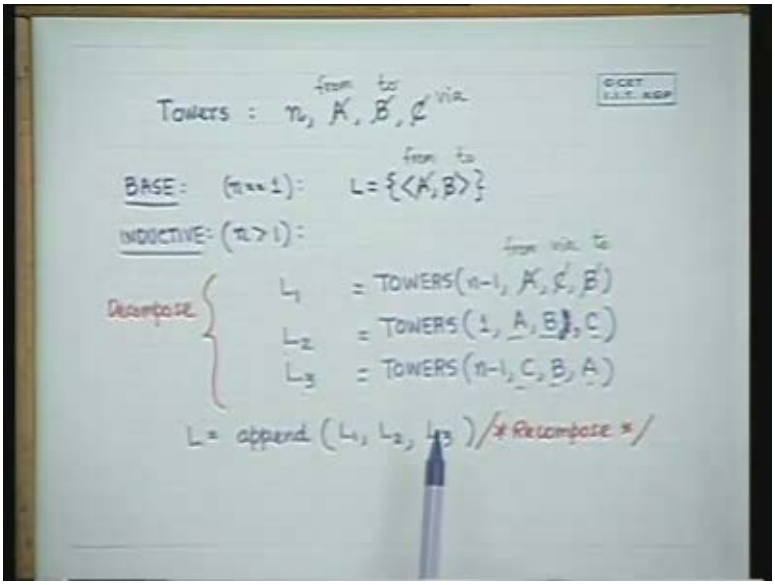
Now we are writing it out for the inductive condition. Please note that we are trying to show slowly reaching the algorithm style, the style of writing out the algorithm but in a sequence of steps. Each of them will be important because we will go back to each of them to see what we can do like we did in the max and min on the other problems. We will retrace that steps back to solve the problem. So the inductive condition is n is greater than 1. And what are our sub problems? Our sub problems were towers (n minus 1, A to C using B), towers (1 from A to B). I hope try and imagine the diagram mentally because slowly slowly we will have to get used to it, C to B using A and each of them will be returning something, a set or a list or a sequence of such pairs.

So let this return L_1, let this return L_2 and let this return L_3. Then what is L equal to? It is a list formed with this one followed by this one, followed by this one. Is that okay? This one followed by this one, followed by this one. So how does it come up? Let us say we will use a function join or append or something like that and let us use a function called append the joins L_1, L_2, L_3. So these are the decomposition steps and this is the recomposition of the solution and this is the basic step and all these from the inductive steps. So here we have the decomposition and here we recompose the solution back and obviously L is what is returned out of this problem. Now when we write out a function like this, there is something else which is left, these are the actual disks

and we have inductively defined it. So these are the actual disks. So now we have, these are the actual parameters, these are the actual parameters. I am sorry here also A B C will be there but when you write it out, you must have this as a formal parameter as a variable. Isn't it? This just cannot be an actual parameter.

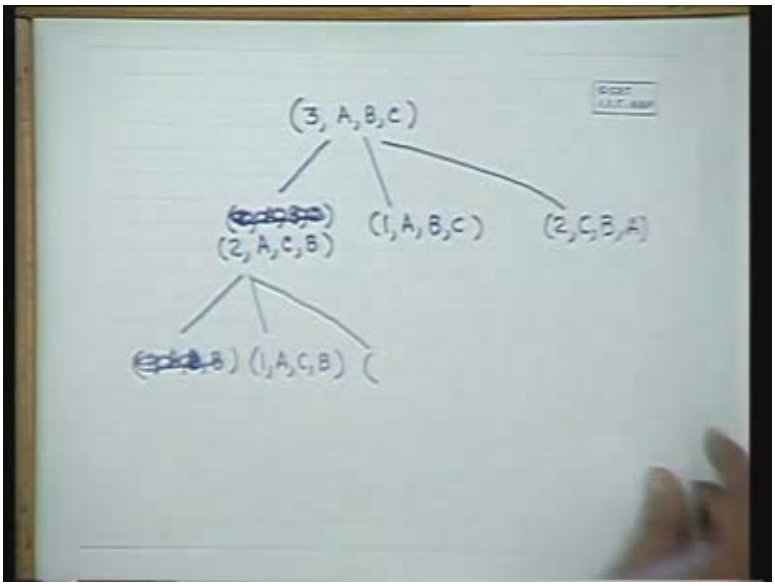
So let us convert it to variables and let us say this is called from, this is called two and this is called via and we replace A by from everywhere here, B by 2 and C by via in terms of variables here, here, here and here and you will get a function in terms of variables. So though it may be a repetition, I will still work out the recursion tree to see how it goes. Suppose you are solving n with A. Now what is the advantage by the way, just quickly. What is the advantage of having this, if somebody gives you A C B, B C A all of them are solved when you have variable otherwise you have to convert it because we will use recursion to solve these sub problems and unless they are written in terms of variables and arguments, we cannot write them down in terms of actual disk value because this has to be in terms of some variables.

(Refer Slide Time 08:55 min)



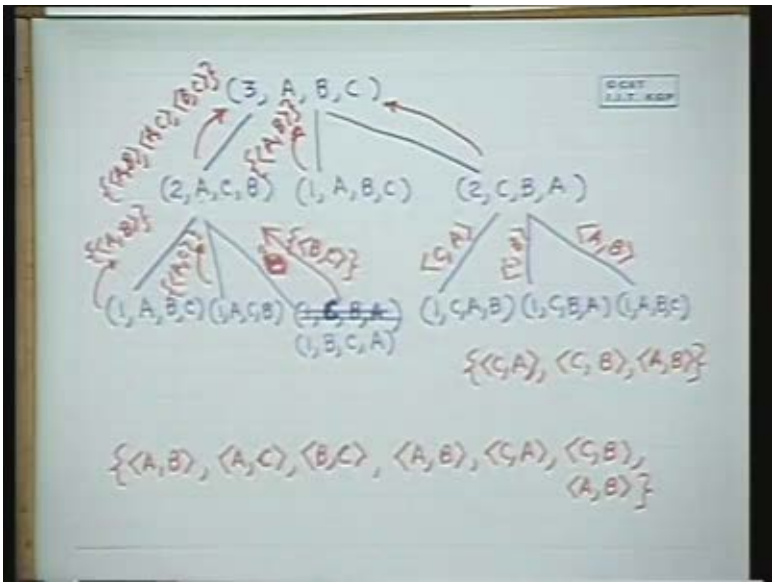
So if we have to solve 3 A B C, the decomposition would be 2 A B C sorry 2 A C B, 1 A B C and 2 C B A then this would be 1 A C B. Now this is from, this is to, this is via, so from to via, 1 A C B from two okay sorry this, this will be something else, lets rate all over again. 3 A B C, this is from, this is to, this is via and what is it? We have to solve 2 A C B, 1 A B C and 2 C B A, right. Now to solve this one, I love to solve 1 A B C, 1 A C B and 1 B C sorry, this is C B A, right. This one is done, to solve this one I have to solve 1 C A B, C A B this is fine, this is okay, yeah this will be B C A, 1 B C A right because this is from to via let's go back. This is from to via, this is from via to and this is via to from, so this goes here, this goes here and this goes here, that's fine. 1 C B A and now let's get this one correct, this is 1 A B C.

(Refer Slide Time 10:08 min)



So this is the decomposition tree which will be set up. What will this return? This will return A B, this will return A C, this one will return I don't know which way to write it anyway A B C this one, just write it this way B, C. This one will now return this followed by this followed by this in this sequence. So this one will return A B, A C, B C. This one will return A B, similarly this one will return C A, this one will return C B and this one will return A B and this will return the sequence C A followed by C B followed by A B.

(Refer Slide Time 14:56 min)



So this will return, I am writing it here C A followed by C B, followed by A B and this followed by this, followed by this will give you the final solution which will be A B. So this is how the

recursion tree develops and this is how the problem is solved. So we got a decomposition of the problem this way in terms of variables, in terms of the basis condition and an inductive condition. Now we are left to convert this to a program, we are very close to converting it to a program but before converting it to a program; we will analyze it a little. I do not want to convert it into a program immediately because there are several issues before converting it into a program. So we will, this is what we called the initial definition or the decomposition.

Now we are at a problem unlike the factorial or the Fibonacci where the definition given to us was not directly and inductive or decomposed one. They are having a look at the problem; we try to find out what is an inductive definition to this problem. So having seen a problem which was not very well known to us, we will now revisit a problem which is very much known to us. And to have a look at this problem we will see what sorting methods we have done and was there any decomposition or something related to that problem there, whatever we have done was there any decomposition or recomposition mechanism there. So we saw variations of exchange and bubble sort, so exchange sort we saw we know, bubble sort we know and we did the sort of tournaments, sort of something like that which you have done three kinds of sorting routines we have done. And in the first two sorting routines if you recollect the algorithm, there are two loops and inside these two loops you did something. You compared two elements and do an exchange.

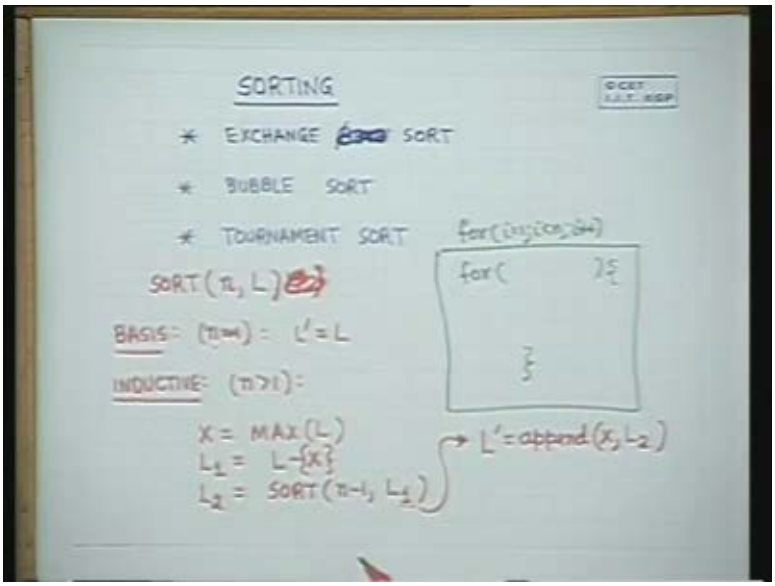
In exchange sort you compared the ith element with the jth element and in double sort, you compare the i and the i plus 1th element and you exchange and in tournament sort we set up a tournament, we found out the minimum then we found out the second minimum then we found out the third minimum that is the sequence in which we did it. If you look at these two sorting routines also and if you analyze the loops then you will see this part of the program computes either the minimum or the maximum. In one loop the minimum or the maximum element is determined and that is what the inner loop does, it finds the minimum or maximum element of i to n and put sit in i then increment i by 1, this is i equal to 1, i less than n, i plus plus this is the routine.

When i is 1 maximum element is found in array one then i is made 2 therefore the maximum element from 2 to n is found. When i is made 3, maximum element from 3 to n is found and this is the way it proceeds. Isn't it okay? This is exactly what the algorithm does. And if you see all these three algorithms have got a single decomposition rule. The rule is like this, sort a list of n elements and give me the sorted list L dashed, this is the sorted routine. What is the basis condition? n equal to 1. Then what do you do? L dashed is equal to 1, alright. What is the inductive condition? n is greater than 1 then the first thing that we do is x is equal to max of L, you find out the maximum element in all these three. Then you L_1 is equal to L minus x, this is the list obtained from L by removing this element X, alright. So if you write it this way, may be it will look better.

L_2 is equal to sort, suppose this returns L dashed, this returns a list of sorted elements which is L dashed. So L_2 you sort n minus 1 element. From which list? L_1. And what do you form L dashed finally after this? L dashed is equal to append x in L_2, X is put in the front of L_2 and you get L dashed which is a sorted list. Is that understood? I repeat. To sort n elements in a list L, your basis condition is and you are returning a list L dashed. Your basis condition is if n is equal to 1; L dashed is the same as L. The inductive condition is if n is greater than 1 then you find out the

maximum element in the list L. L_1 is L minus x, remove X from L, sort this list and then put it in the front of the… That's what you have done in exchange sort. Indirectly when i is equal to 1, you found out the maximum and put it in data one and the rest of the program does what. It sorts the rest of the array. After you have placed it, the rest of the whole program continues to sort the rest of the elements and this append is automatically done because you have already put the first element in data one, same in bubble sort and in tournament sort you are extracting the maximum one after another which is another way of looking at this program.

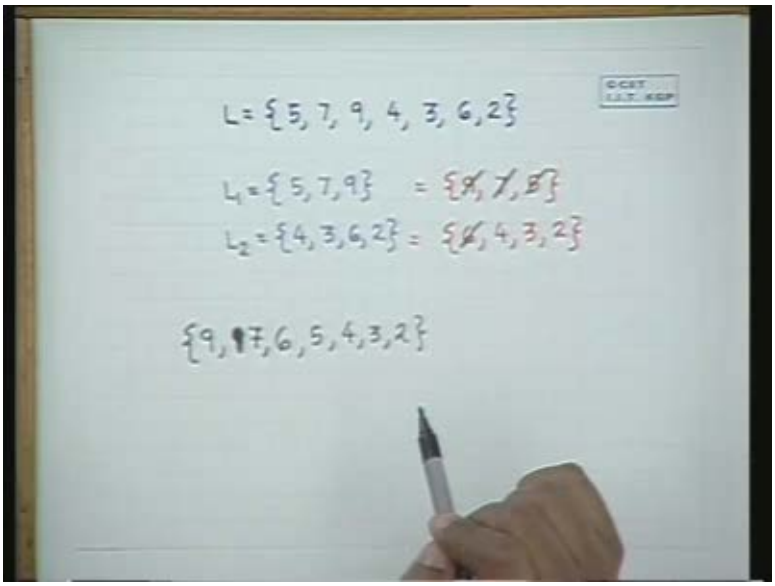
(Refer Slide Time 22:58 min)



So, just this one is the sequence of max extractions. So, all the three programs are based on a very very similar basic decomposition idea. Now we will see another decomposition which is slightly different from this. I do not want to go into the recursion tree of this, it's obvious. Suppose, I have got a list L of elements 5 7 9 4 3 6 2 and I break up this list to solve, I have to sort this, alright. I break up this list into two parts, any two parts which are non-empty. So I break up this list into two parts L_1 5 7 9 and L_2 4 3 6 2. Now inductively sort this and sort this. How I will inductively sort these two, I will see later on but if I can able to sort this and obtain, after sorting what will I get? 9, 7, 5. After sorting this one, I will get 6, 4, 3, 2. Now I can combine these two to get my original solution without sorting them directly.

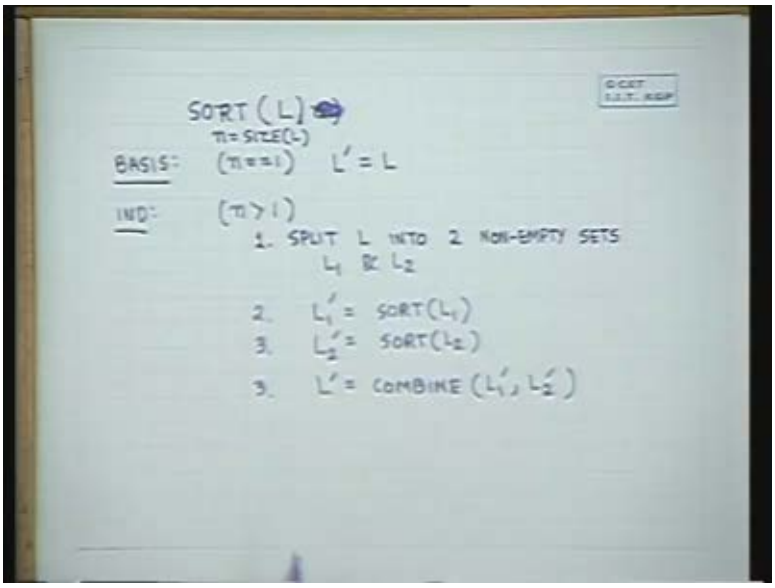
I prepare a tentative list, I look at the larger first element whichever is the larger of the two I put it here and remove it. Then again I see the first element of these two, whichever is the larger of the two, I put it here and sorry I put it here and remove it. I check the first larger of the two, whichever is the larger I put it here and remove it. The larger of the two I put it here and remove it then one of the lists becomes empty; I just copy the other thing here.

(Refer Slide Time 26:23 min)



So if I give you two sorted list, it is very easy to give me the resultant sorted element. So I have not found the maximum of any numbers here but I have got an idea of how to sort these elements in a way which is different from the way I used to sort it earlier. So let's try and write down what we are doing. To sort a list L of n elements what is the basis condition, same. If n is equal to 1 then L dashed which is the sorted list is equal to L. The other inductive condition when n is greater than 1, we apply the following steps. First split N into two non-empty sets L_1 and L_2, alright. The second step is the inductive steps, L_1 dashed is sort L_1 you can forgot or you can compute the size of the list. Here this means size of L, so first thing you can do is n is equal to size of L, if n sort L_1, the other sub problem L_2 dash is equal to sort L_2.

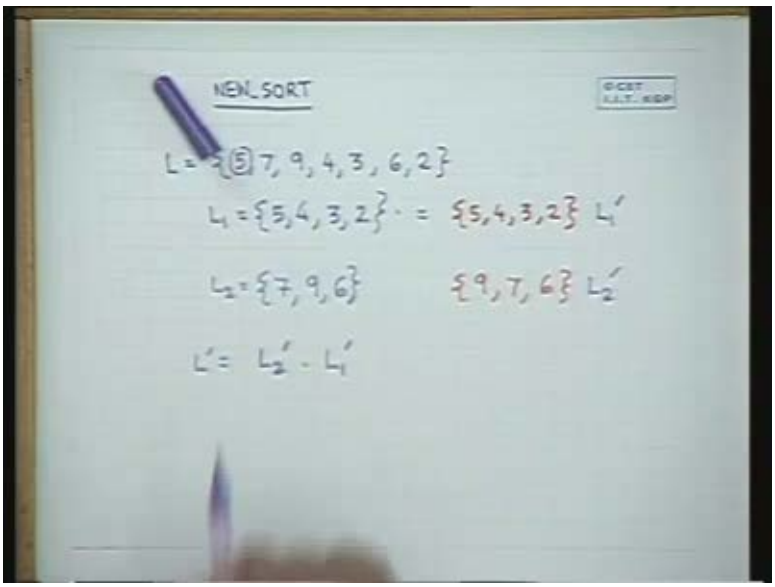
(Refer Slide Time 28:44 min)



And the third one you have to write a third function L dash is combined L_1 dash L_2 dash. And the combination routine is something we saw here. Given two list we are given two sets or two ordered list, we create a tentative list, temporarily list we take we find out for whichever of the two first elements is larger we put it in this list, remove it from here and continue doing till one list becomes empty and copy the other list at the end. So if we do this, we are going to get another sorting algorithm and the sorting algorithm will vary depending on where we have split the list also. We can split a list of n elements into 1 n minus 1 or 2 n minus 2 or half, half or three fourth, one fourth all of them will work. Any questions here? This is one way of solving the problem and if you are aware of the actual algorithm which, the final algorithm which follows this you can have a look at your books but we will go through it, this algorithm in its final version is called merge sort. We will come back to this again.

We will see a third way of decomposing, we have seen one before we will see another one now and now we will see a third way. Let's take the same set of elements which you were doing before. L, so now we have to sort this. We have seen max decompositions, we have seen split, sort and combine. Now we will see another one, pick out any element from here say pick out the first element and from to list in this way L_1 L_2. These two lists will be based on this first element. All those which are less than or equal to 5 will go into L_1, all those which are greater than 5 will go into L_2. So you start scanning from this one, 5 is less than equal to 5, so 5 goes here. Next 7 goes here, next 9 goes here, next 4 goes here, next 3 goes here, next 6 goes here, next 2 goes here, we have reached the end. Now can anyone tell me what to do after we have done this? Sort this, sort this, so after you have sorted this and after you have sorted this then how will you find out. So this is L_1 dash and this is L_2 dash and what is L dash is just this one, this one followed by this one.

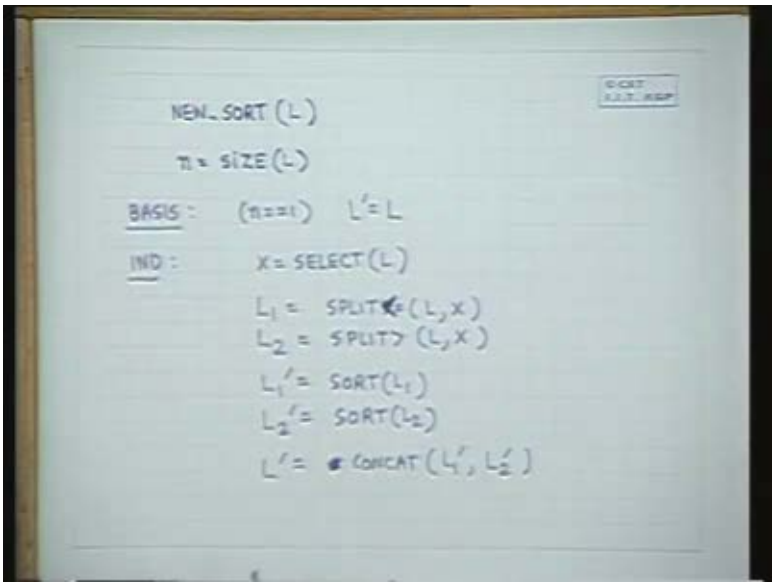
(Refer Slide Time 32:46 min)



So it is L_2 dash concatenated with L_1 dash or appended with L_1 dash or joined with L_1 dash. And this is slightly different way of splitting the two lists. In the previous one you split the two list flat without doing anything on them and then you combined them by a function which we

called combine. In this one we split them in a particular way so that the combination became just attaching them one after another. So this sorting algorithm L, suppose n is equal to size of L, here again basis condition is say n is equal to 1, L dash equal to L and the inductive condition x is equal to we have to select an element out like we selected first five, from L we select one element out, alright. And then you form two list, L_1 is equal to split greater than equal to in L, x or less than equal to whatever. We did less than equal to, so we will keep what we did.

(Refer Slide Time 35:23 min)

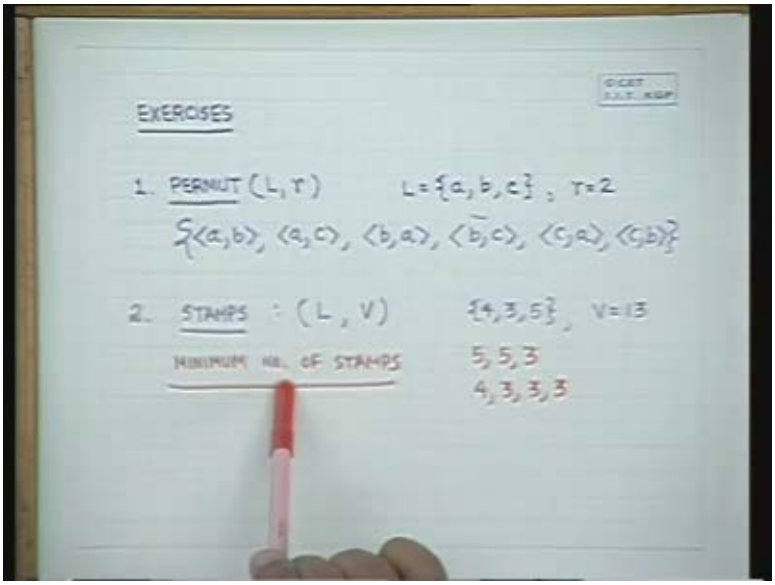


L_2 is equal to split greater than L, X and then L_1 dash is equal to sort L_1, L_2 dash is equal to sort L_2 and L dash is equal to just concatenate or join or append whatever. We understand what we mean by this. So this is yet another way of inductively decomposing the problem. So we see that for the simple sorting problem, the way of recursively decomposing the problem into sub problems are many, there is not a single way to do it. And coming to algorithm design, the first step before you reach the final solution as I mentioned before, you should be start writing your program straight away. You should think at all the possible ways in which you can really solve the problem, these are all initial solutions to the problem. And once we have a look at all the ways in which we can solve the problem then we analyze each of these solutions to find out which is the exact solution that we will take but before going into that analysis which involves lot of other issues, we will see first how to convert these problem into direct programs and then we will see how to improve these programs into more and more useful and more and more efficient functions.

Before closing I will give you two problems to solve for which you have to find out inductive definitions. How to solve them? You don't need to write out programs, you have to find out decompositions for solving these problems. The first problem that I give you is permut (L, r), L is the set, L is a set of indistinct elements. For example L maybe say a, b and c distinct, no repetitions and r is an integer say r is equal to 2. You have to provide, you have to solve the problem of finding out all two permutations of these elements. In this particular case, the answer is a set of elements a b, a c, b a without repetition b c, c a, c b. There is no order in each of these

elements; you can produce them in any order that is not the point here. The point is I want all the permutations. For example if this was, L was this and R was 4, there is no solution, answer is not possible. If r is 0, answer is not possible, so you get the basis conditions. If r is 1 then it is a, b and c. If r is 3 then it's all permutations of a b and c. So you have to write out, you have to find out how to solve this problem. The second problem is you are it is a, let us call it a stamps problem. You are given a set of stamps, stamps means you are given a set of values of stamps, alright. So you are given a set of stamp values L and you are given a value V, alright. Now you ought to find out a set of stamps which make exactly the value V. Given any, suppose you are given any such set for example if I give you 4 3 and say 5 and if I give you a value say 13 then a solution could be 5 5 3, another solution could be 4 3 3 3 alright, this is another solution.

(Refer Slide Time 41:47 min)



Now for every search, they may not be a solution also. I may give you such a set for which there is no solution and such a V for which there is no solution. In that situation you have say there is no solution that the stamps problem does not end in finding any wrong solution, you have to find that solution with minimum number of stamps. For example in these one, this will be the solution, not this. So we have got two problems as exercises and we will discuss solutions to these problems later on. So, I request you to try and solve these problems for the next class.