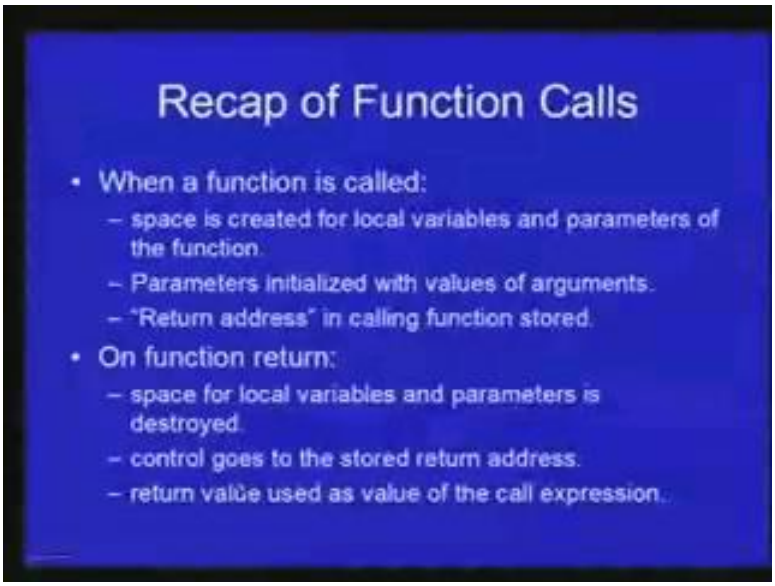
Introduction to Problem Solving and Programming

Prof. Deepak Gupta
Department of Computer Science Engineering
Indian Institute of Technology, Kanpur

Lecture No. # 20

In the last lecture, we have talk about the techniques for recursion for problem solving, and we saw that essentially using recursion a function can call itself. And we can explored this capability by defining its solution to a given instance of a problem in terms of a simpler instant of the same problem. And we saw our several examples for choosing recursion for solving simple problems. Today we will take about how recursion actually works, and what happens really when a function call itself, and how the what is really happening inside the machine when function it calling itself, and hold the whole team work.

(Refer Slide Time: 00:59)

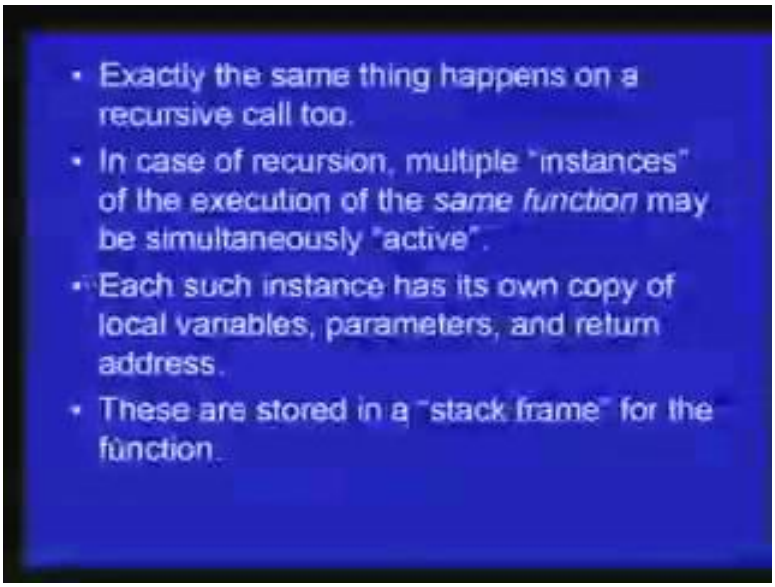


Therefore, to capitulates what happens when a function is called by a another function, so recollect when a function is called the following things happen, space is created for the local variables and parameters of the functions which is being called, the parameters are initialized with the values of the corresponding arguments. And the return address in the calling function is told, what do you mean by return address? Is the late in the calling

function where control will go back, once the call function return and this is the just statement of the order of the expression following the function called.

And when the function return, the space which have been created for the local variables and parameter of the function that is destroyed, the control goes back to the told return address; therefore, suppose to go back in the calling function, and the return value of the function is used as the value of the call expression. Now with recursion exactly the same thing happens, when recursive call is made, there is no difference in terms of the kind of things happening when the when in normal non-recursive call is made.

(Refer Slide Time: 02:14)



But the important thing to understand is at in the case of recursive function multiple instances of the same function may be simultaneously active. What does mean is the following. Consider for example, the factorial function, and we call factorial n let us assume we have to define factorial recursively has been solved in the last lecture. So, when factorial n is called before this, this instances of called with the factorial function finishes that is the before it returns, the call is made again to the factorial function this time the parameter n minus 1, and before that finishes again a call is made to a factorial function with the parameter n minus 2 and so on and.

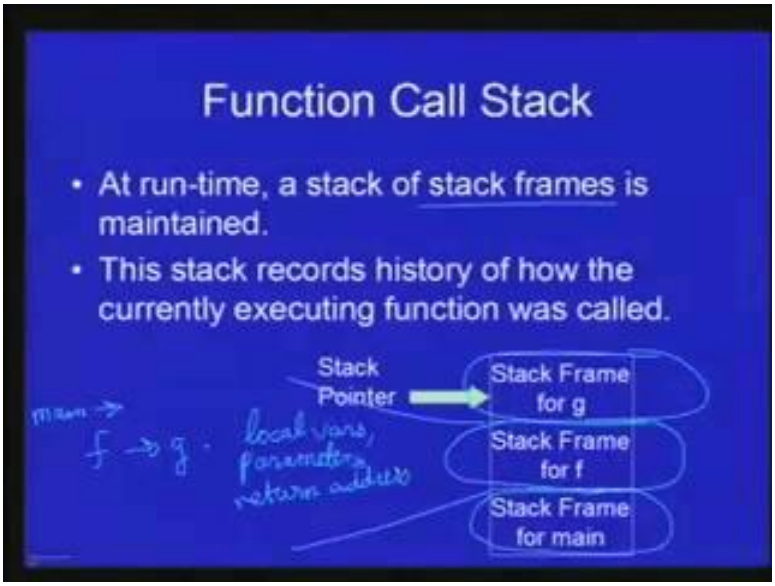
So, at the same point in time, it is possible that multiple instances of the factorial function are of the whatever the recursive function is they are simultaneously active, that

is their not yet finished. Of course, one is currently executable, but the others are suspended and waiting for the call function to return.

Now, each such instances have its own copy of local variables parameters, and they return address. In another words if factorial n was called and that call factorial is n minus 1 that call factorial n minus 2 and so on, for each of the instances of the factorial functions which are simultaneously active. Each will have its own copy of the parameter n, there were no local variable in the definition, and also of the return address where the control is supposed to go back; once that instances of the (( )) call to the factorial function return.

So, all this things the local variables parameters, and return address for a particular function call instances are stored in what is known as the stack frame for the function, and it will becomes clear shortly write it is called as stack frame.

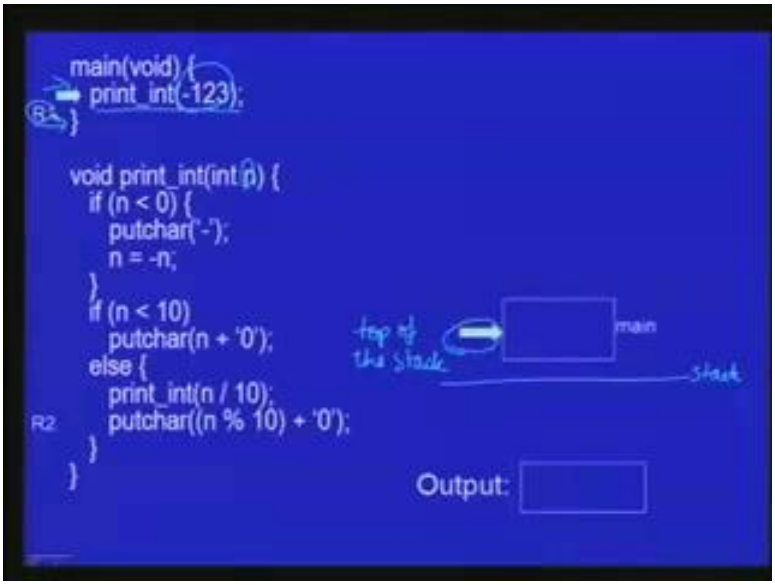
(Refer Slide Time: 04:13)



So, essentially at the runtime what is known as a function called stack is maintained by the systems. So, it is a stack of stack frames, what you mean by stack is what is normally mean by stack in English you say, if stack is essentially if I (( )) place one on the top of the other. And anyone want to remove something from the file you removed it from the top of the file, and when you place something on the file should place it on the top of the file. So, the function calls stack is similar. So, let say the function f called the function g, and f itself for space called by the main function.

So, the function calls stack will capture this step. So, each element in the stack is a stack frame. So, at the bottom is the stack frame for main, and one top of that is the stack frame for f, and one top of that (( )). On the stack frame for any function will contain space for the local variables for the function, parameter of the function, and the return address form the function where control is supposed to go back once the function return.

(Refer Slide Time: 05:30)



So, let us now see the in terms of an actual recursive functions what is really happening at the runtime. So, the example have taken is the print in function from the last lecture which prints the given integer n. So, let us assume that the main function calls print int, where the parameter value minus 123, and let say the control is a before the called to the print int space print int function in main.

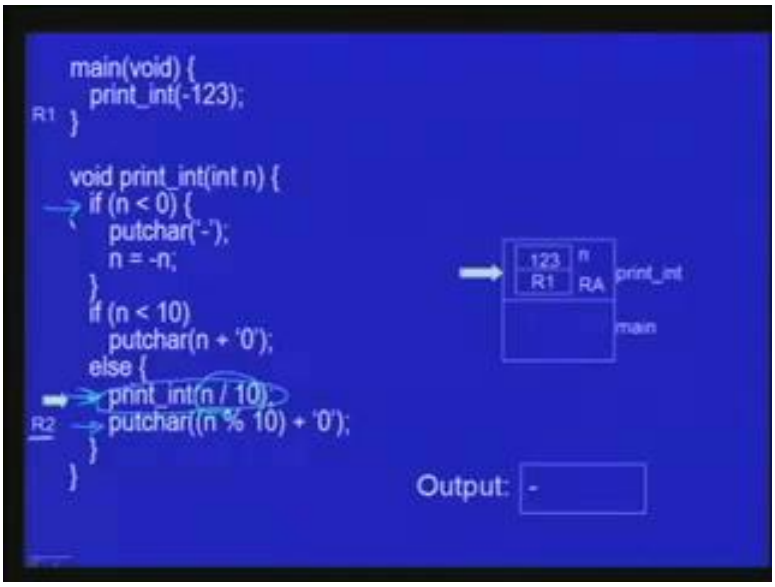
So, at this point int (( )), this what the stack looks like, only the stack frame for main is on the stack, and are not concern to much about what is really contained will be more concerned with what the stack change for the print in function contained. And this arrow points to the stack frame at the top of the stack is called the stack pointer, and as function gets called this stacks frame to exposed on the top of the stack and so the stack pointer will move upwards, and when function return they are stack framed will be removed from the stack and the stack pointer will move downwards.

So, in the first step when the called print int function is made, what will happen if that is the stack frame for the print int function will be created, and that will contain space for

the parameter n, there are no local variables of this function, and value of n in this stack frame will be initialized to the value of the argument being fast to this function which in this example happen to be minus 123. In addition this stack frame will also contained the return address, where control is supposed to be go back one with the invocation of the function print int return, nope print int print int is called from this location in the function meant, and once print int in return the execution (( )) at the next statement. So, I am calling this location in this program s r 1.

So, when main calls printed the return address stored in the stack frame for print int will be r 1, and that indicate where the program (( )) function return.

(Refer Slide Time: 07:55)



So, when when the function print int called this stack frame for the function print int has been created as you can see has space for the parameter n, how the function print int and the return address, the return address is R1 which happens to be this location in the program and the value of n is minus 123 which was the argument which was pass to the function and the control has gone to the first statement in the print int function.

So, let us now trace the execution of the print int function for this value of n, and see exactly how the recursive is called servant to work. So, the first statement is if if can be less than 0, now n is minus 123 which is indeed less than 0. So, this condition evaluates to prove.

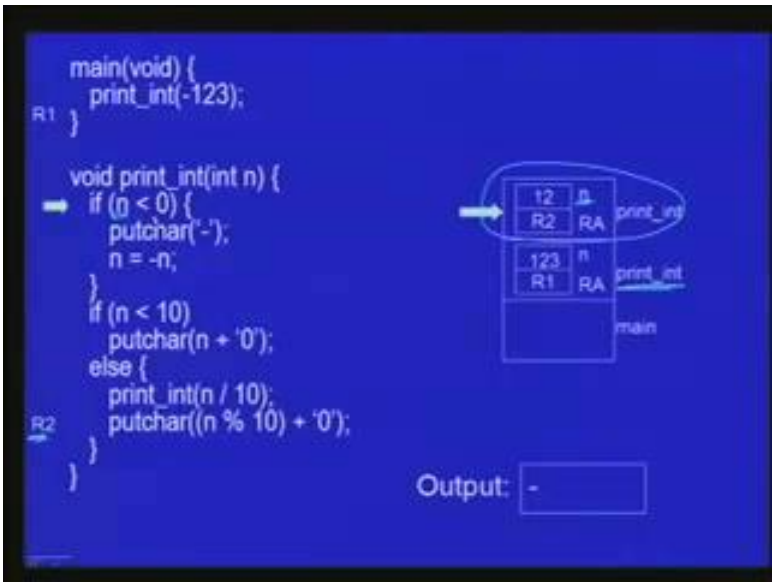
So, the control goes back to be goes to the next to the first statement in the then part of (( )) statement which is the put char the minus sign, and so when we do this in the output the minus character will appear nothing else changes to that solves that happens. And then the next statement is n assign minus n when that executes a course the value of n here just changes to plus 123.

So, that what happen? Now, the next statement check if n is less than 10 which of course, the loop case is fall. So, we start executing the body of the else part, now here is the recursive call to be print int function. Now, again what is meant to happen is that if the frame for this call to print int will be created that will also have space for n and for the return address, and the value of n for that called to print int will be the value of this expression which is being passed s arguments, now the value of this expressions happens to be 123 divided by n which is 12.

So, the stack frame which gets created will have space for another instance of the parameter n, which will have the value 12, and that should be the top stack frame and the return address will be this address R 2, because once this called print int return, we control you suppose to come back here, and execute this put char statement which follows the called print int function.

So, that is what happen from this function is called, and of course as soon as the function gets called apart from the stack frame etcetera, then created control will again go back to the first statement of the print int function.

(Refer Slide Time: 10:37)



So, this is what happen? This is the new stack frame, which has been created, note that it also helps the parameter n here, but this time it is, it has the value 12 which is the arguments which was passed from the previous invocation of the print int, and the return address will also different the return address is this location R2. And now we are executing we are started to execute another instants of the function print int while the previous instant is still not finished, that is what happen print function.

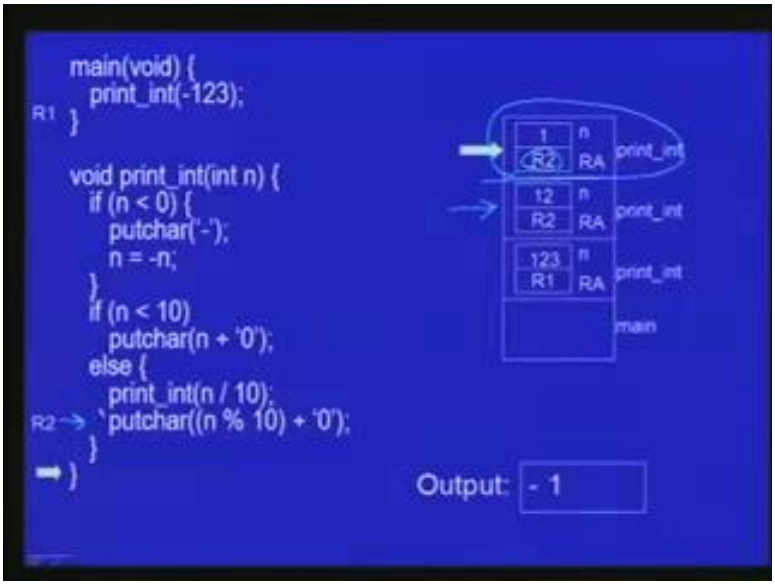
So, again we started executing these courses, but it, but note that the value of n is now the n in the top most stack frame, and which is 12 at the running time and the value of n is no longer 123.

So, even though in the program it is the same end, but at the runtime is a two different ends, because each instant of the function call as (( )) copy of the local variables and the parameter. So, let see. So, again the control will flow the same way, since n is not less than 0, n is 12 in this case, so this statement is kept, and is not less than 10. So, we start executing the else part, and again there is recursive call to print int with the arguments and that n. Now this end now evaluates to 12, because that is the value of n in the top most stack frame. So, whenever local variables or a parameters is exists its value is value for that variables or parameter in the top most stack frame.

Even though below that stack frame there may be other stack frame for the same function, which also have the variable n or which have also the same variable or the

parameters. So, the value of n here is current sheet well and n by 10 would evaluate therefore to 1.

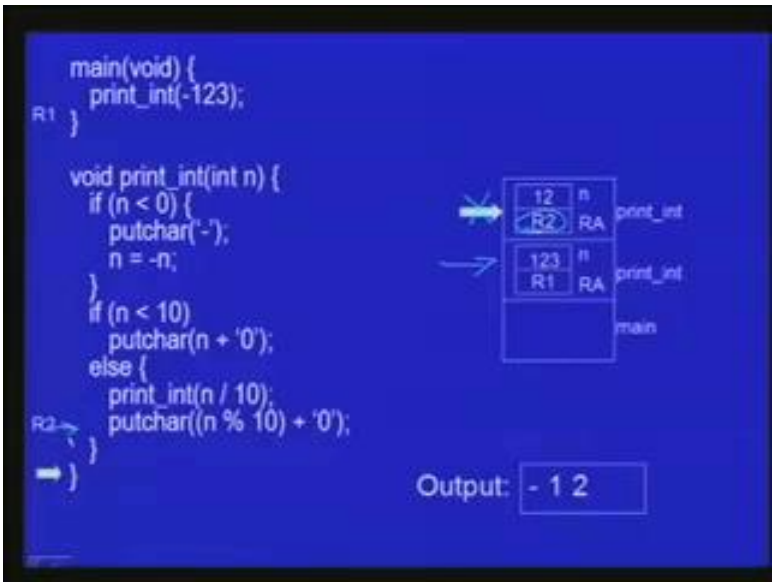
(Refer Slide Time: 12:36)



So, again the same thing happens when this time print int is called recursively, it moves stack frame is created with space for n and the return address, the return address is again R2, because again the print int has been called from this location. And therefore, when it return the control should come back to this statement R2, and once again the function start getting executed from the beginning, now for the value of n is equal to 1.

So, again n is not less than 0, but n is less than 10. So, put char in plus 0; that means, put char be f is equal to 0 plus 1 which gives the (( )) of 1, and so the put char statement result in the character 1 being printed on the output, and the else part is of course kept and the control goes to the end of the function. Now, if the function has finish and the function is about return, now when the function is going to return (( )) to happen is that this stack frame is going to be destroyed or it is going to be popped off from the set, this will again become the top most stack frame, and the controls will go back to the location or to in a reverse to this location which is the statement immediately following the call to the print int in the calling function which also happen with the print int in this case.

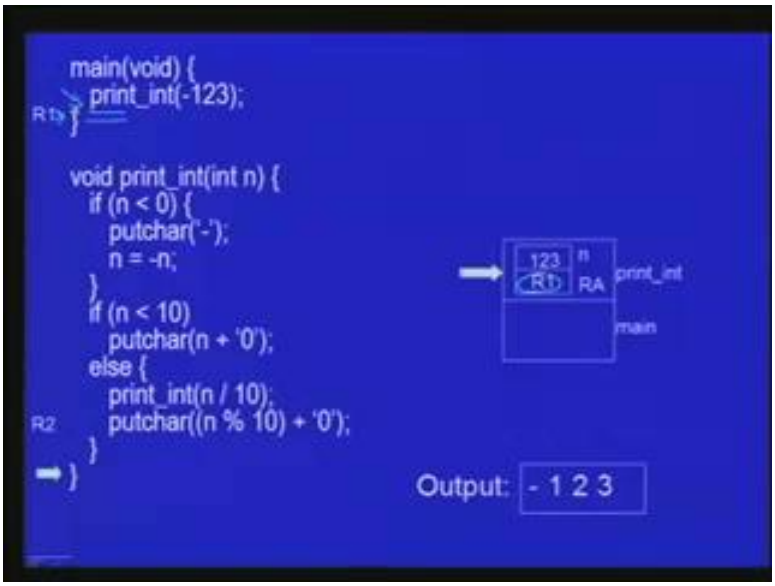
(Refer Slide Time: 14:04)



So, when these instances of print int return that is what happen? This stack frame for the top most print int, if it destroyed and control goes to the return address which was stored in that stack frame. So, we are now again executing the second instances of the print int function which was suspended when it made recursive call to itself.

So, now the value of n here in this expression is again 12, because the top most stack frame is just one, and that has the value of 12 for n. So, when this state's when this put char statement executes n percent 10 gives the value 2 that plus the x code of 0 gives the x code of recorrected 2, and put char print the character on the screen. And now this instants of the function is also about the return and when the return happens again the same thing happen, this stack frame will be deleted, this stack frame will be come the top of stack frame and control will go back to this return address which again is the address of this put char statement.

(Refer Slide Time: 15:15)



So, that is what happen? Now, we are back in the very first location of the print int function, the value of n is now again 123, that gets printed that is divided by 10 and the remainder comes out to be 3; 3 plus (( )) is equal to 0 gives the x to power 3, and get printed. And this invocation of print int is also now about return, and this time when the return happen the control will go back to the location R1 which is the end of the function made.

So, the main function will return or if there was some another statement in the main function after print int, then both statements should have started executing, and the stack frame for mean you would now become the top most stack frame.

(Refer Slide Time: 15:59)

So, the function has return. So, finally, we are back in the main function, all invocation of the print int including all recursive invocation are finished, and the output is minus 123 as expected.

(Refer Slide Time: 16:16)

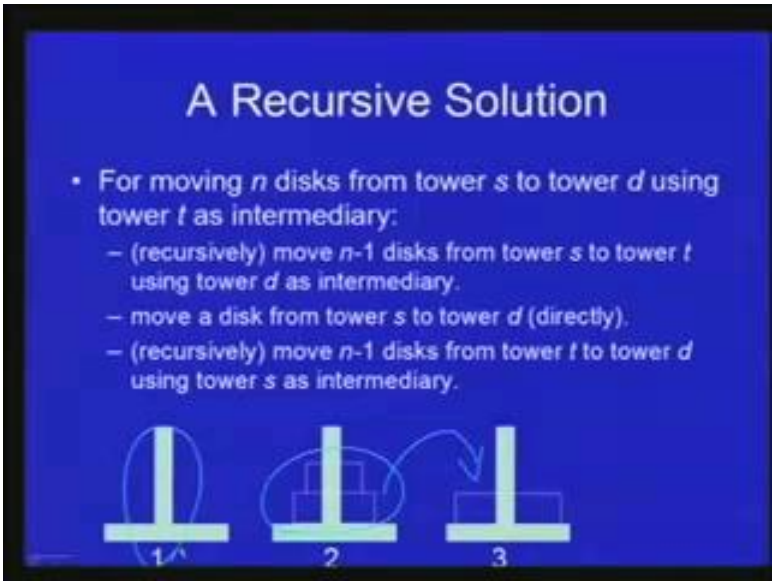
Lets now take another very interesting problem this can be solved very elegantly using recursion, and which is actually quite difficult to solve without using the recursion, this is like a puzzle and is called the problem of the tower of Hanoi.

So, essentially in this problem, you have the 3 tower, let us call them 1, 2, and 3; and in initially in the within the tower number one, there are the n disk which has plate on top of the error; each disk has a whole within it, so it can be put inside the tower, and the disk at the bottom has been largest diameter, and the diameter is decreasing as we go up. So, this picture shows three disk, but in general there could be any number of disk.

So, we assume that there are n disk in general, now the task is to move the n disks from tower 1 to tower 3, and in the one move what can be done is at the top top most disk from any tower can be removed, and it can be placed on top of the stack on any other tower. But we have to always follow the constraints that will larger disk can never be placed on a smaller disk. Now, this actually appear to be very difficult problem and for large values of n it can be actually very difficult to solve, if you tried hard and you might be able to obtain the solution for n equal to 3,

But you will find that as the value of n becomes larger, the solution become harder and harder to obtain and without using without thinking about the recursion in general algorithm is not all that is you to obtain,

(Refer Slide Time: 18:12)



But let us think about this problem using recursion. So, the top disk move n disk from tower s to from tower d, and the third tower is here because so that we can play some disk on this tower during the processes as, as an intermediate stack for the disk. So, from the tower s to tower d we have to move this end, now how can we do that? Is think about

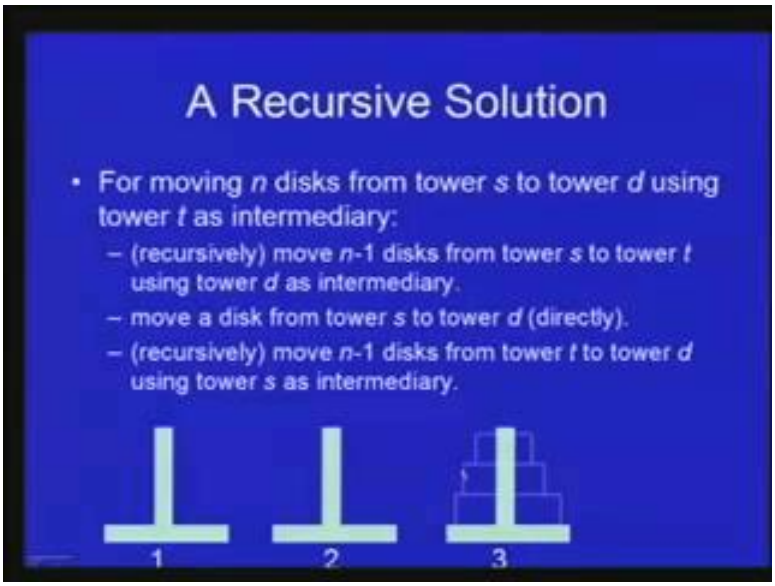
it recursively, when the base case is simple, the base cases is n is equal to 1 and when n is equal to 1 all we have to do this take the disk from the tower s and place it on the tower d that is not a problem at all, but if n is more than 1.

So, let us think about this problem recursively. So, suppose n is more than 1, now suppose from how by matrix the top n minus 1 disk can be move from the can be move from one tower to the another two another tower somehow. And of course that somehow is not really (( )) it can be achieved by recursion. So, the first step is to recursively move in top minus 1 disk from the tower s to the tower t, that is the temporary tower using the tower b as an intermediate.

So, whenever you want to move from the disk you have to specify one intermediate tower where disk can be stored on their way to final destination. So, suppose somehow we are able to do that; that is we are able to move that top n minus 1 disk from the sources to the temporary tower in the destination has the temperate. So, suppose somebody allows us to do that or show that how to do this (( )). Then the next step is to move disk - bottom disk from the source to the destination.

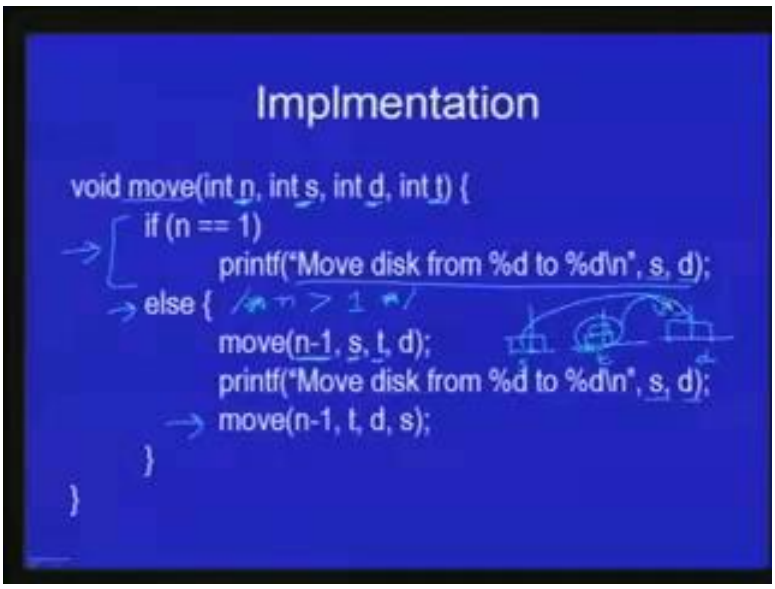
So, this is an easy steps, this is we can always perform; and finally, now again the task get to move these two disk from the intermediate tower to the final destination tower, and for this move we can use now this original source has an intermediate tower, because it is it is now empty. Now again how do we move disk two disk from here to here, next the simpler instant of the same problem. So, we can use recursion.

(Refer Slide Time: 20:47)



So, let us assume that by the recursion we can do that, but one's do that we can see that the problem is finished and we are obtain the solution. So, you can see using the recursion the solution to do this problem is extremely simple to obtain as a matter of that. So, let us write a program to do this.

(Refer Slide Time: 21:08)



So, I have written this function move which we calls the tower of Hanoi problem, it has four parameter, n is the number of disk which have to move, s is the source tower which could be 1 2 or 3, d is the destination tower which could be again be 1 2 or 3, where t is

the intermediate tower or the temporary tower. So, initially if the disks are in the tower 1 and we have to move to the tower 3, then we call it function has moved value of n whatever the number of disk take 1 comma 3 comma 2, the source is one, the destination is three, and the temporary or intermediate tower is 2.

So, this the base step if n is equal to 1, then we just send the move to the top disk from the source to destination directly, this can be complete an similar steps. Otherwise then n is greater than one, we use regression; so the first move is n minus the top n minus 1 left from the source to the temporary using the destination d as the intermediate. So, at the end of this steps the configuration will look something like this, this the source, this is the destination and this is the temperate. So, the top n minus 1 has been move to the temporate, and the move disk from source to destination. So, this disk goes where and finally, move this n minus 1 disk from the temporary to the final destination, these three disk go here.

So, that for the solution is actually extremely simple, one two think about recursively, I leave it an exercise for you to grace the solution using the technique that we just saw for the print int function, and convinced to ourselves for it is the small values of n let two or three, that it does work correctly by tracing the sequence of the recursive function called and creating stack frames, and putting values of the parameter and so on and so forth. So, that is the end of the lecture, in the next lecture we continue our discussion on the recursion and talk about some situation where we should not be used recursion, and how we should not use recursion in an apparently or an efficiently, because I will see it is heavy often to write the program for extremely inefficient when not return properly using the recursion.