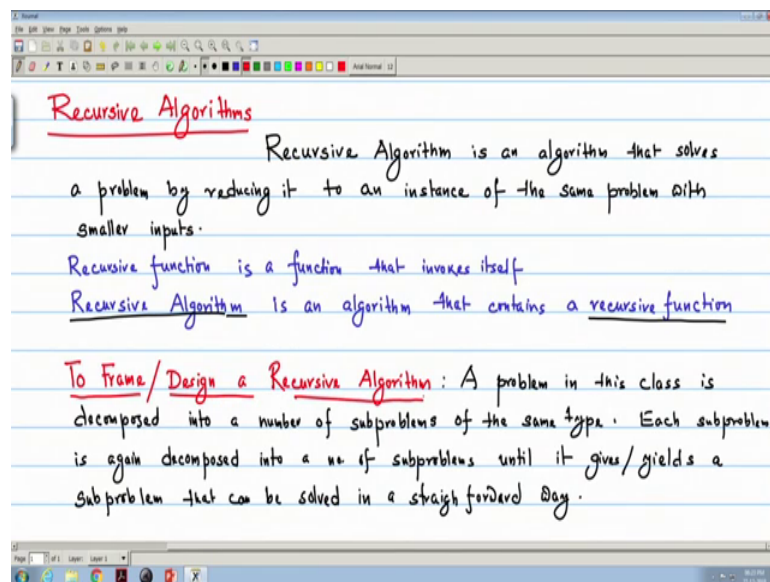


Discrete Structures
Prof. Dipanwita Roychoudhury
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 29
Recursion (Contd.)

We have learned the Recursion and Recursion we have defined it as a process of defining an object with itself. We have learned the function, sequences and sets and structures how they have recursively defined. Now, today we will read the recursive algorithm which is a very important and elegant way to solve problems.

(Refer Slide Time: 01:01)



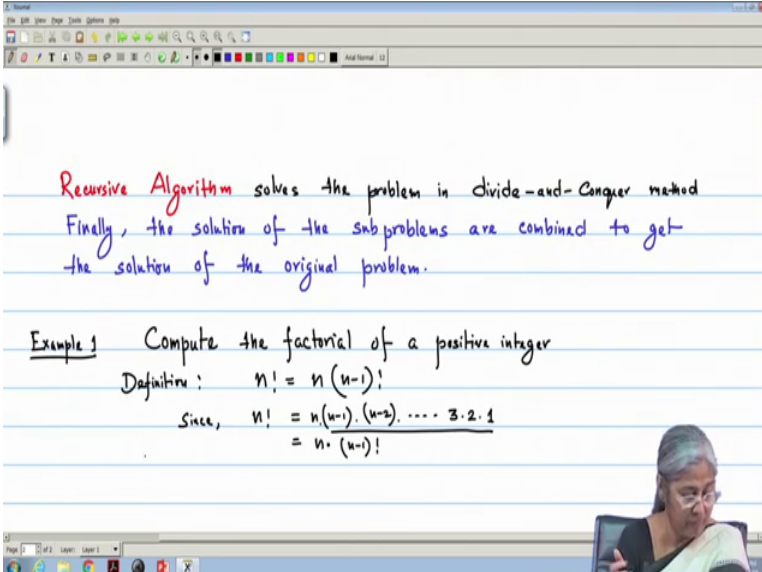
So, you can define that recursive algorithm is an algorithm that solves a problem by reducing to an instance of the same problem with smaller inputs. So, we defined a recursive algorithm; it is an algorithm that solves the problem by reducing it to an instance of the same problem with the smaller inputs. Now, if we remember we have defined the recursive function a function. If we remember recursive function that is a function that invokes itself; now very simple way we can tell the recursive algorithm is an algorithm that contents a recursive function.

So, since already we know the recursive function and recursive function that invokes itself. So, an algorithm that contains a recursive function we will call that is a or in this way we can frame a recursive algorithm. So, what is the approach to write or to frame a

recursive algorithm? So, approach is so to frame or how to frame or to design a recursive algorithm. Since, just now I mentioned that recursive algorithm contains a recursive function. And, since we know now how to form a recursive function then automatically that will the algorithm will call itself.

So, we can define in this way that a problem in this class is decomposed into a number of subproblems. It can be one or more than 1 sub problems of the sub the nature of subproblems are the same as that of the original problem. So, I can write into a number of some sub problems of the same type. Then each subproblem is again decomposed into a number of simple sub problems until it gives a or it yields a sub problem that can be solved in straightforward way. So, the approach is nothing, but divide and conquer method.

(Refer Slide Time: 08:38)



Recursive Algorithm solves the problem in divide-and-conquer method
Finally, the solution of the subproblems are combined to get the solution of the original problem.

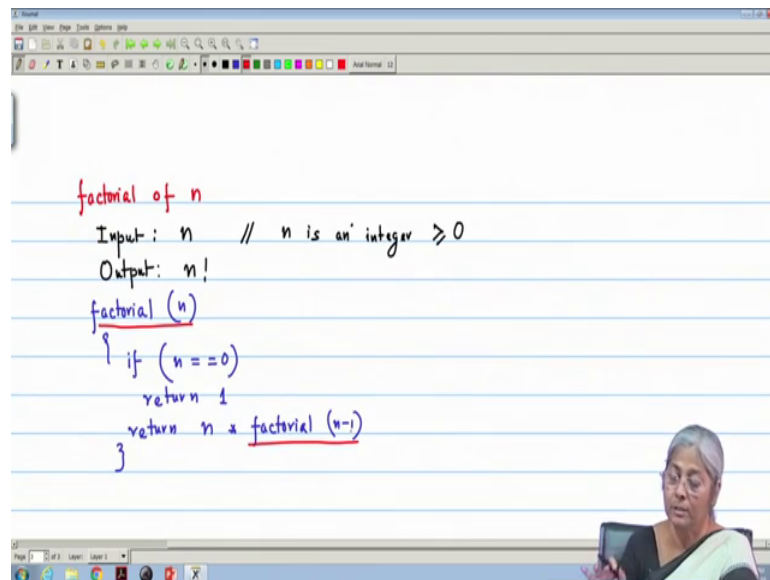
Example 1 Compute the factorial of a positive integer
Definition: $n! = n(n-1)!$
Since, $n! = n(n-1)(n-2) \dots 3 \cdot 2 \cdot 1$
 $= n \cdot (n-1)!$

So, I can tell that recursive algorithm solves the problem in divide and conquer method. So, that approaches the divide and conquer method that one problem it divides into a subproblem, but nature is same it is of same type. And, again each subproblem another same type of subproblems is and until we get a sub problem in such that as, if it matches with a given condition or it can state straightforward way I can get the solution. Then we add all the solutions of the subproblems then finally, we the solution of the subproblems are combined to get the solution of the original problems.

So, now we describe the method of recursive algorithm design by with a number of simple problems. So, first we see our one example of one example one very simple example of recursive algorithm, that how to compute the factorial of an positive integer. Example is computing of compute the factorial of positive integer. Now, if we limit remember the recursive function then how to compute the factorial? All of you know the definition of factorial that we know the definition that in factorial is n into n minus 1 factorial. Since, we know the definition of factorial that it is n into n minus 1 n minus 2; the product of all integers up to 1.

So, this is same as it is up to n minus 1 n minus 2 then it is nothing but, the n minus 1 factorial. Now, if we see the nature that when we are computing the factorial n that n factorial is n into n minus 1. That means, if I write a function then as if this function with smaller inputs because, n minus 1 is less than n with smaller inputs I can frame a function. And, recursive algorithm contains this type of function a recursive function or algorithm if we write that contains this recursive function that will form a recursive algorithm.

(Refer Slide Time: 04:05)

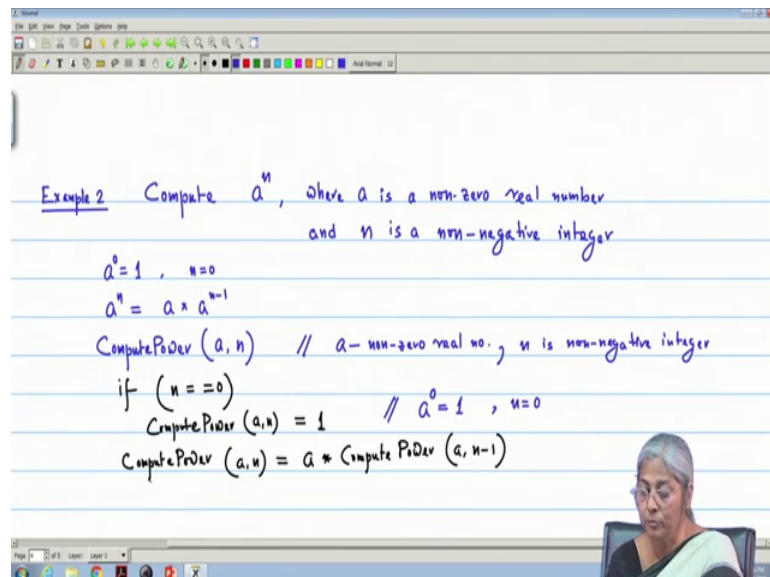


So, how we use that algorithm, if we write the factorial you write a factorial or computation computing factorial of n . So, we can write my input is n and n is an integer greater than equal to 0 and output is n factorial. So now, I can write if I give the function

name the factorial n then, if n equal to 0 then return 1 else return n into factorial n minus 1.

So, we see that here the this is the function is factorial function is factorial n and we have written when we are computing, that thing the function n into factorial n minus 1 which is of same type with smaller inputs; that means, n minus 1 is less than n. So, it satisfies the definition of the algorithm that has it the same the original problem is decomposed into a subproblem with smaller imports. So, this is a simple example of computing factorial n which is a recursive algorithm.

(Refer Slide Time: 16:53)

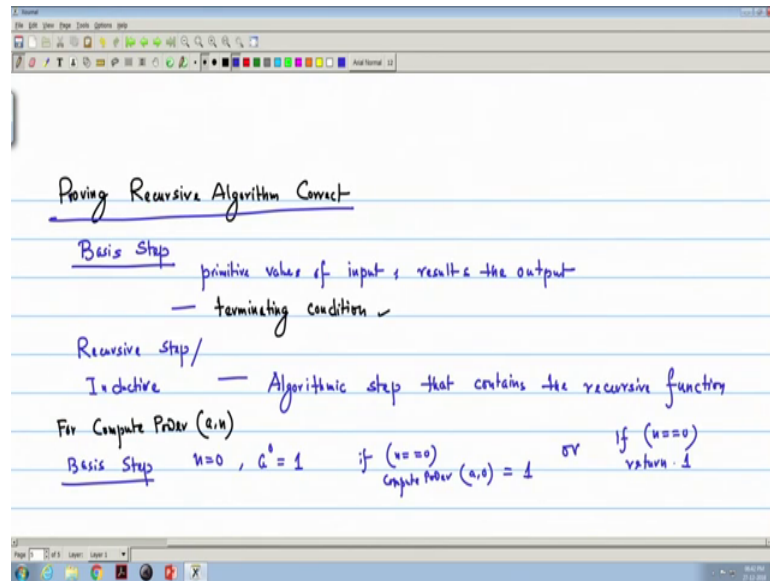


Now, we see another example say we compute power. So, we compute a to the power n, where a is non-zero real number and n is a non-negative integer. Now, we know the a to the power 0 is 1 we know. So, when n equal to 0 it will return 1. So, for n equal to 0 it will because 0 equal to 1, for other values of n I can write a to the power n in to a to the power n minus 1. So, as if when we will compute we want to compute a to the power n.

It can be decomposed into a to the power n minus 1; that means, if I write a function to computer to the power n then the similar function with inputs as n minus 1 a n minus 1; that means, that since n minus 1 is smaller than it. So, with smaller inputs I can frame another function of same type. So, one algorithm that contains a recursive function and it will be recursive algorithm. So, very simple way if I can write, I give the name that function is compute power.

So, it is a give compute power and my inputs are a n, where a is non-zero real numbers and n is non-negative integer. Now, I can write that if n equal to 0 then compute power I can write returning, I can write compute power a n equal to 1 else compute power a n equal to a into compute power a n minus 1 that is a function with smaller inputs I can write. So, this is a recursive algorithm.

(Refer Slide Time: 21:29)



Now, how we can prove that it is giving a correct result; that means, proving recursive algorithm correct. Now, if we remember that when we have raised the recursive function, the way we have proved that the recursive function gives a correct result and it is using the mathematical induction. Now, if again if we see that the two simple examples that if n equal to 0 return 1. So, as if this is my terminating condition; that means, there is there is some condition that what the algorithm will stop otherwise it is return in into factorial n minus 1.

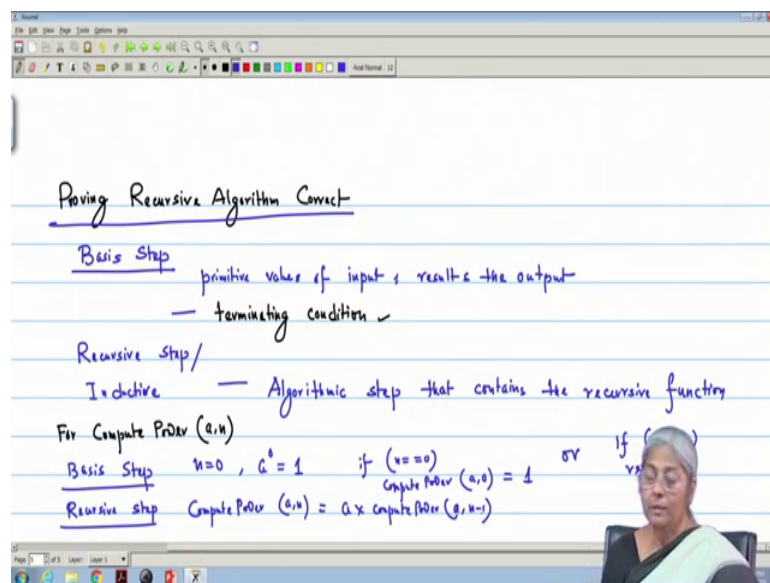
So, I can think that this is my basis step n equal to 0 then return 1; that means, for n equal to 0 it gives some value. And, otherwise that I am writing a recursive step which actually contains my recursive function. If I see the next example again it is same that if n equal to 0 it return 1. Because, all of we know that that a to the power a to the power 0 equal to 1. So, this is equivalent to my basis step because, for n equal to 0 it is giving 1. And, then computing power that is a recursive step and again this step contains the function and it

is a recursive function because it is compute power n . And, then it is decomposed into a function which required the inputs r a n minus 1 and my n minus 1 is smaller than n .

So, those the same way we have proved the recursive function gives correct result, the same way we can tell that the we can tell that we can prove the recursive algorithm correct. So, here also there is some basis step just now what we have seen. Here also there is some basis step which is actually for some primitive values of input or primitive values of input results the output. And, in respect of recursive algorithm we can tell this is nothing, but my terminating condition; that means, when the function can be computed straightforward way when we stop.

So, this is my terminating condition. Then with what is my inductive step or recursive step? So, this is my recursive step or inductive step. It is that the algorithmic next step that contains the recursive function. So, for the previous example of computing power I can tell my basis step that for n equal to 0 ok. So, I can write for compute power n my basis step is basis step is n equal to 0 a to the power 0 equal to 1. And, what we have written that if n equal to equal to 0 then return 1 or compute return 1 or compute power a 0 equal to 1 or I can write if n equal to equal to 0 return 1. So, this is my return 1. This is my terminating condition.

(Refer Slide Time: 27:47)



And what is my recursive step? My recursive step is that compute power a n is a into compute power a n minus 1.

(Refer Slide Time: 28:16)

Assume for $i=n$, the result is true,
i.e. the recursive step gives the correct result—
 $\text{computePower}(a, n) = a^n$

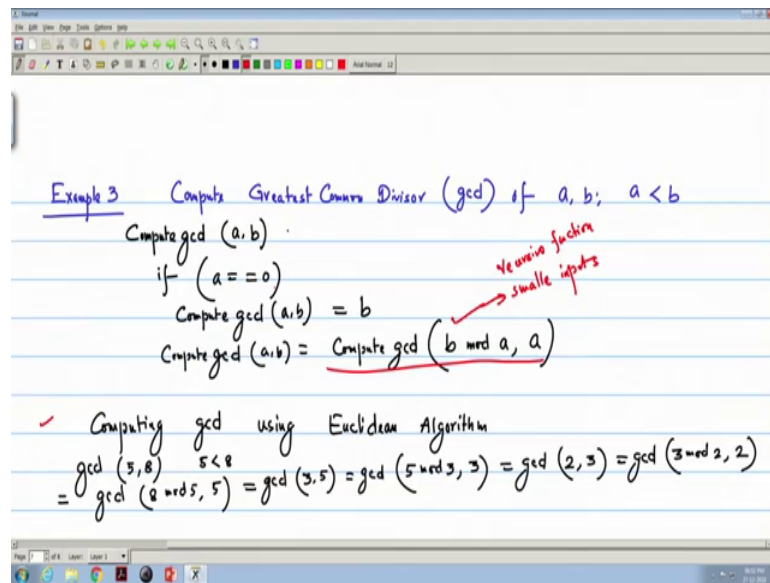
for, $i = n+1$ (using Mathematical Induction)

$$\begin{aligned}\text{computePower}(a, n+1) &= a * \text{computePower}(a, n) \\ &= a * a^n \\ &= a^{n+1} \\ &= \text{result is correct}\end{aligned}$$

Now, if I use mathematical induction so, if we assume for i equal to n the result is true; that means, here result is true. That means, that is the recursive step recursive step gives the correct result gives the correct result; that means, my compute power a^n gives a to the power n this is my correct result. So, for i equal to $n+1$ for i equal to $n+1$ using mathematical induction this compute power a^{n+1} is a into compute power according to that algorithm this becomes a^n . Now, for using my mathematical induction this is equal to a to the power n since, the result is true we have assumed.

So, this becomes into a to the power n and this becomes a to the power $n+1$. So, for compute power a^{n+1} this is also a to the power $n+1$ and this is result is correct. So, here also we see that recursive algorithm also we can prove using mathematical induction. This is very obvious because, the way we have defined the recursive algorithm that algorithm contains a recursive function and the recursive function proving we have used that mathematical induction. So, if we can prove in this way the same thing. Now, another very popular example or of computing Greatest Common Divisor the GCD using Recursive.

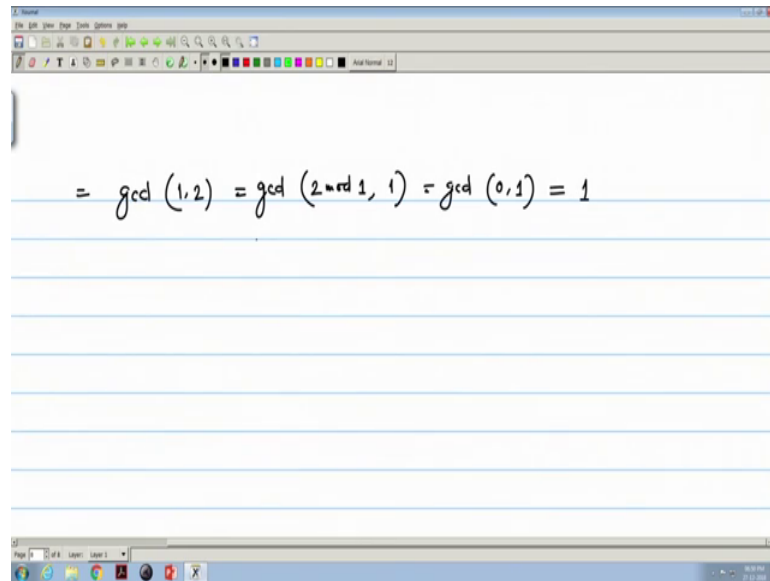
(Refer Slide Time: 31:11)



We can write recursive algorithm, that I gave my third example as the computing a compute greatest common divisor; normally called gcd of two integers a and b and a less than b. Now, we can write the compute gcd function as the compute gcd a n a b, just if we write that is if a equal to equal to 0. Then compute gcd a b equal to b else the recursive step compute gcd a b equal to compute gcd b module a and a.

So, this is nothing but all using the computing gcd using Euclidean all algorithm. So, computing gcd using Euclidean algorithm, see if I take an example of say am I want to compute gcd 5 and 8 5 less than 8. So, I can write gcd 5 8, I can write gcd 8 module 5 5. This becomes gcd 3 5 gcd 3 5, then according to the algorithm this becomes gcd of 5 modulo 3 and 3. It says gcd of 2 3, then this becomes gcd of 3 modulo 2 and 2.

(Refer Slide Time: 34:30)


$$= \text{gcd}(1, 2) = \text{gcd}(2 \bmod 1, 1) = \text{gcd}(0, 1) = 1$$

If I continue then 3 modulo 2 is this becomes gcd 1 2 is gcd 2 modulo 1 1. So, this becomes gcd 0 1 and now gcd a b equal to 0. So, this is equal to b so this is 1. So, this is our Euclidean algorithm and we can get now, see here the way we have written this is that compute gcd a b and this is compute gcd b mod a. So, actually this is with my smaller inputs, this is my every time we are getting that recursive function with smaller inputs, this is my recursive function with smaller inputs. So, we can compute gcd or recursively so, this is one recursive algorithm for computing gcd.

And, with this example we see that it is nothing, but the implementation of Euclidean algorithm. And, in this way we can frame the simple some simple examples I have given that how we can design, how we can develop some recursive algorithm. Well, which actually there is some basis step in algorithm form; we can write that is my terminating condition. And, the recursive step that is my which contains the recursive function; that means the same function with smaller inputs.

Now, we will continue with some bigger examples that how to frame or how to design the recursive algorithms in my next class.