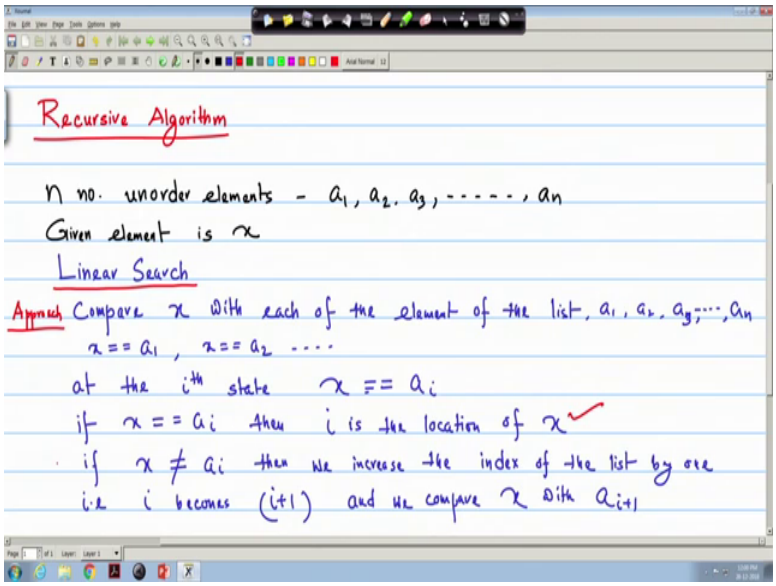
**Discrete Structures**
**Prof. Dipanwita Roychoudhury**
**Department of Computer Science & Engineering**
**Indian Institute of Technology, Kharagpur**

**Lecture – 30**
**Recursion (Contd.)**

So, we are discussing about the Recursive algorithms. In the last lecture we have learned how to frame or how to design small recur recursive algorithms, where mainly we have considered the algorithms which contains some recursive function.

Now, today we will see a different type of algorithms we will continue the discussion on the recursive algorithms only, but today we will consider different type of algorithms not that algorithms containing recursive function they know. So, we can write recursive algorithm for our to solve the problems.

(Refer Slide Time: 01:07)



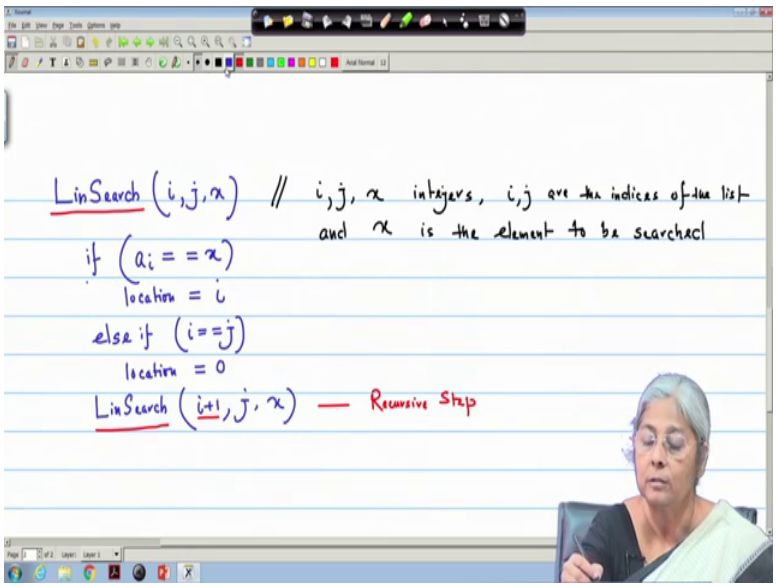

So, we know that searching is a very important and important technique where the problem is that we have a n number of elements and n number of unordered elements we considered unordered elements say a 1 a 2 a 3 up to a n and given element is x. So, one element we searched or a given element is x. Now, the problem is that searching the problem is that we have to search or we have to find out whether x is in this list or not this within this a 1 to n elements.

So, linear search what we do what is the approach? The linear search that we compare x with each element of the list starting from a 1 a 2 a n. And if so, we compare x with each of the element of the list a 1 2 n. Now, if at the; that means, x is compared to a 1 then if not then if it is not equal, then x is compared to a 2 in this way we will go to at the ith state we will compare x with a i.

Now, if it is equal then if x is equal to a i then i is the location where i is the location of x. So, this is the is a positive search i x exist in the list, but if x is not equal to y if x not equal to a i then we increase the index of the list which is i list by one; that means, i becomes i plus 1. And we will compare and we compare x with a i plus 1 and then again we will continue that whether x equal to a plus 1 or not and the same thing will go. So, this is our approach of linear search this is my linear search approach.

Now, see here there is no mathematical function involved what last day the or last lecture we have discussed mainly the factorial in or to find out the GCD or to find out the now power a to the power n when the algorithm contents the one recursive function. So, here no mathematical function is there, but these we can write this approach or this algorithm we can write a recursive algorithm to solve this problem of linear search. So, we write the function of linear search.

(Refer Slide Time: 07:11)

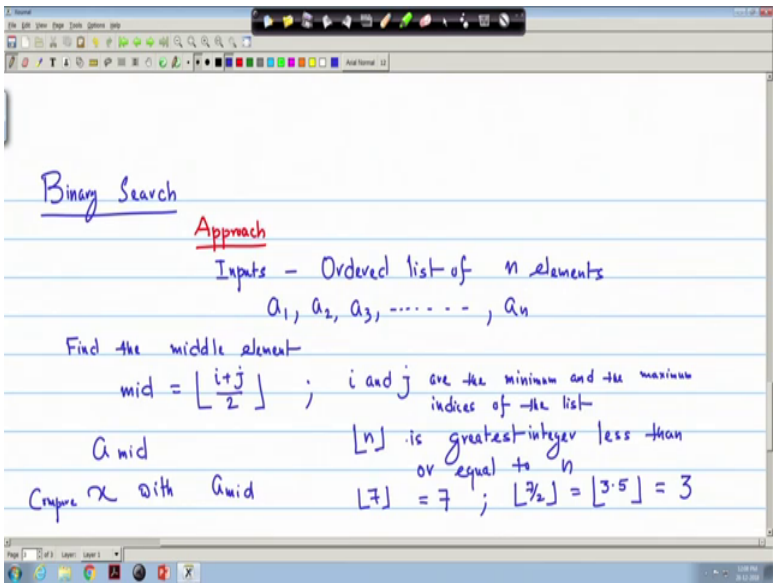



We give the name that say lin search the function name is lin search and the inputs are i j x. Where, i j x are integers here i j at the index of the list or the indices of the list and x is

the element to be searched. Now, if a i at the ith stage. So, ai equal to x then we can write that location is i. Else if a i not equal to x then we will compare the two indices; that means, if i equal to equal to j; that means, my full list as is already compared and then we do not get the element x in the list. So, we can write that location equal to 0. Else we can write that may lin search i plus 1 j x. So, this is my recursive step this is my recursive step.

See that my function is lin search on the inputs are i j x and if it is not equal to ai then I have increased my index i plus 1 and j. So, actually this is smaller number of inputs the way we have defined the recursive algorithm and then we again we call the same function; that means, the function in books itself with smaller inputs. So, this is the recursive function of lin search the linear search ok. So, we can easily we can write the linear search algorithm.

(Refer Slide Time: 10:43)



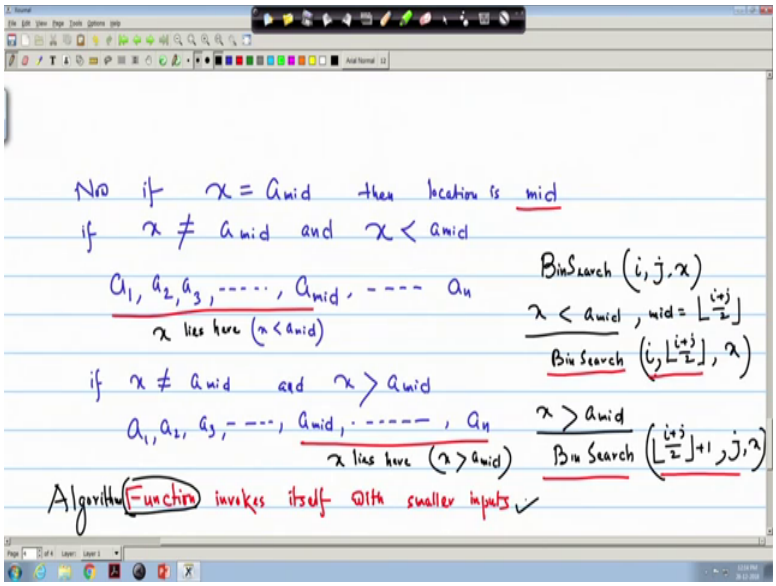

Now, if we consider the binary search, then we can write the binary search. So, quickly we see that what is the approach what is our approach see again we have a list of n elements, but in this case the elements are ordered list. So, inputs are here the input inputs is a ordered list; that means, it is a sorted array or sorted list ordered list of n elements. So, ordered list of n elements and right this is a 1 a 2 a 3 up to a n.

Now, here we find a middle element find the middle element as a mid equal to floor or firstly, we see the middle location and then we will get the. So, we see first the location

the middle. So, it is i plus j by 2, where i and j are the minimum and maximum index of the list indices of the list. And this is a floor function we normally we defined say floor n is the greatest integer less than or equal to n. See if it is ceilings 7 then it is equal to 7, but if it is ceiling 7 by 2. So, it is 3.5 sorry so, this is a floral floor 7 by 2. So, this becomes greatest integer less than or equal to n here n equal to 3.5 so, this becomes 3.

So, I get a mid is a integer. So, that I get a mid is a integer value and now I can find the middle element as the a mid. Then I will search or I will shortened my list first I will compare x if x is the element we searched then I will compare x with compare x with a mid.

(Refer Slide Time: 14:49)




Now, if x equal to a mid then my location is mid then location is mid; that means, I got the element x exist in the list. If x not equal to a mid and we get x less than a mid since my input list is a sorted; that means, my x if it lies then it must be within the first half of the list. That means, now if I consider this is a 1 a 2 a 3; now I got a mid I have computed and then a n.

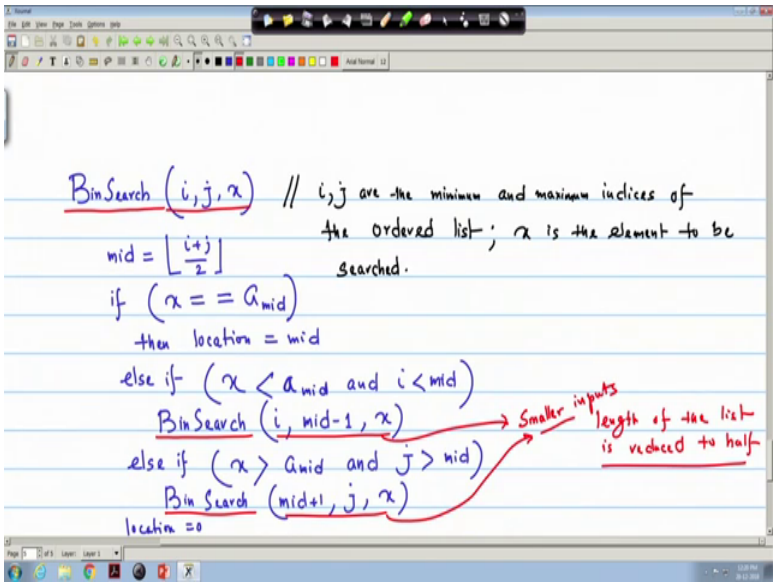
So, if x less then a mid; that means, x must lie between the first half. So, x lies here x lies here if it exist in the list; that means, if when x less then; when x less than a mid and if x not equal to a mid and x greater than a mid, then if I consider the sorted list like a mid to a n then my this lice in the right half of the list since it is a sorted.

So, these becomes x lies here; that means, x greater than a mid z greater than a mid. And see then we will search if I want to write a procedure that recursive algorithm say bin search I give the procedure name the bin search then originally if it is ij x this is my element. Then if it is x less than a mid; x less than a mid then and say my where my mid is say floor i plus j by 2. Then for the first case when it is less than I will be my bin search will be bin search will be that i. Now I have shortened the list to be searched as ceiling sorry this is a floor function i plus j by 2 x.

So, it is for smaller inputs, now if x greater than a mid the second case, then my bin search; bin search will be because it is in the right half of the list. So, it becomes floor i plus g by 2 plus 1 and j x. So, what we see that the way we have defined the recursive algorithm that the function in books itself with smaller inputs. Now, we see the size of inputs become lesser it is first this is the same function and the input size here the original list is actually halved it is partitioned into 2 and this is my left halved that i from j it becomes that I plus j by 2.

Similarly, when it will be in the right half when x greater than a mid then this is again the right half; that means, mid plus one mid is i plus j by 2 floor of that. So, this plus 1 to jx again this is the function. So, the function invokes itself with smaller inputs. Only thing here instead of function I can write my algorithm so, this is here I can write the algorithm invokes itself with smaller inputs. So, this is the way the recursive function of binary search we can frame. So, now if we write the binary search algorithm that we give the name the some bin search and the inputs are same as i j x.

(Refer Slide Time: 20:49)




Only here one thing we must note that it is a i j are the indices are the minimum and maximum indices of the ordered list; that means, sorted in any decreasing in order or in increasing order the ordered list and x is the element to be searched.

Now, first we have to find out we find out the mid the value of mid the middle value we give the name as the variable name as mid it is floor i plus j by 2, then we compare that if x is equal to the middle element which is a mid. Then location equal to mid; that means, we get the element x in the list location equal to mid.

Else if we give that x less than; that means, if it is not equal to a mid then we search whether x is less than a mid x is less than a mid and i less than m; that means, whether it is in the first half i less than mid ok; i less than mid. Then I will call bin search as already we have seen i with the smaller inputs; that means, it will be i mid minus 1; that means, my first half the left half and x.

Else if it is not if I do not get the element in the first half then else if will check that x greater than a mid; that means, if x not less than a mid then x must be a mid greater than a mid and j greater than mid; that means, my right half. Then I will call again bin search and I will call again bin search with smaller input, but this time it will be mid plus 1 then the index j and x. Else if I even if I do not get any of the comparison is true; that means, it is not a mid it is I do not get that x less than a mid and i less than mid or x greater than

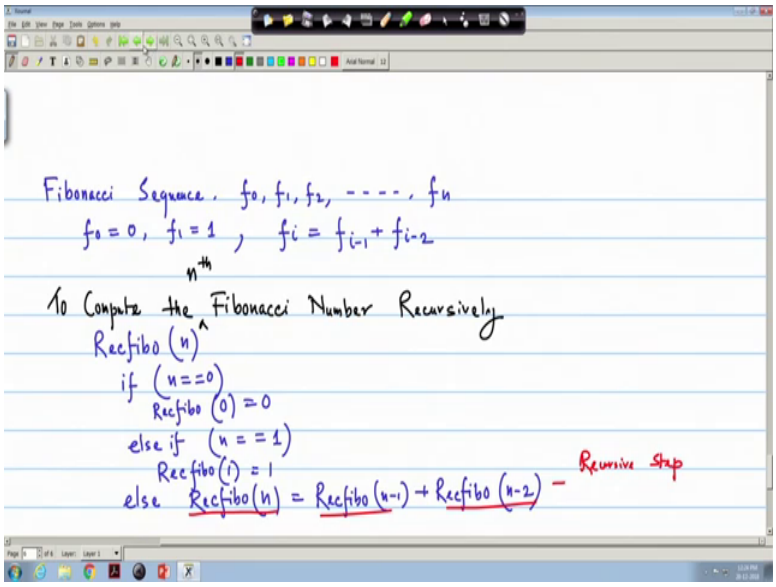
a mid then which is the x does not exist in the list; that means, location equal to 0, then I get location equal to 0.

So, here also we see that first this is my recursive algorithm with the input i j x if it is not the middle element then my list where we are searching the x element the given element x that we are we are invoking the same function with the smaller inputs i and mid minus 1. So, this is the smaller input set some smaller inputs because my j instead of j we are taking we are taking mid minus 1.

Similarly, when therefore, the write up we are considering again this is the smaller inputs, it is a same function we are invoking because it is in the right half. So, mid i becomes mid plus 1 and j. So, here that j becomes mid minus 1 so, lesser inputs here i becomes mid plus 1 so, again it is that. So, the both the cases the length of length of the list is reduced to half. So, it becomes the smaller inputs. So, it is a recursive algorithm.

Now, we see a different type of now recursive algorithms, but again we have learned that as a recursive function we have seen that Fibonacci numbers we know now the Fibonacci sequence.

(Refer Slide Time: 27:25)

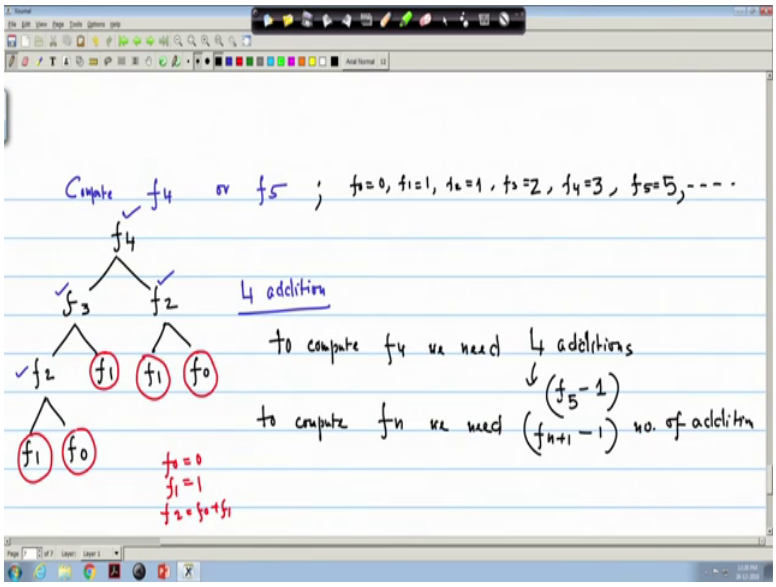



If we remember that we know the Fibonacci sequence as see if I give a 0 f 1 f 2 f n is up to nth Fibonacci number where f 0 equal to 0 f 1 equal to 1 and we can write that f i is f i minus 1 plus f i minus 2. So, it depends on the previous two elements.

Now, we can compute or we can write a recursive function for computing the Fibonacci numbers we see that how we can. So, that thing; so, we write the function we give the function name as say recfibo to compute and give that to compute the Fibonacci numbers recursively. We can write to compute the nth Fibonacci number say number recursively and I give the name as say recfibo n because it is a nth Fibonacci number. So, if n equal to 0, since I know as the initial condition f 0 equal to 0.

So, I can write that recfibo 0 equal to 0 it is given. Else if we see whether n equal to 1 then also the it is given that recfibo 1 equal to 1. Else so, it is if it is not then else we write the recursive step that recfibo n is equal to recfibo n minus 1 plus recfibo n minus 2. So, this is my again this is my recursive step; this is my recfibo n; this is in my with smaller inputs so, this is my recursive step.

(Refer Slide Time: 31:05)




Now, if we see the an example; that means, if I want to compute that compute f 4 or say f 5 just if I draw that the way it is computing we know that f 4 the recursive step is f 3 and f 2. Then when it is f 3 is computed again it is f 2 and f 1 it is f 2 is f 1 and f 0. Now, since the and see f 2 is f 1 and f 0, see my f 0 and f 1 are given; that means, we can compute straightforward way. So, these are the things I know since because f 0 equal to 0 is given and f 1 equal to 1 is given.

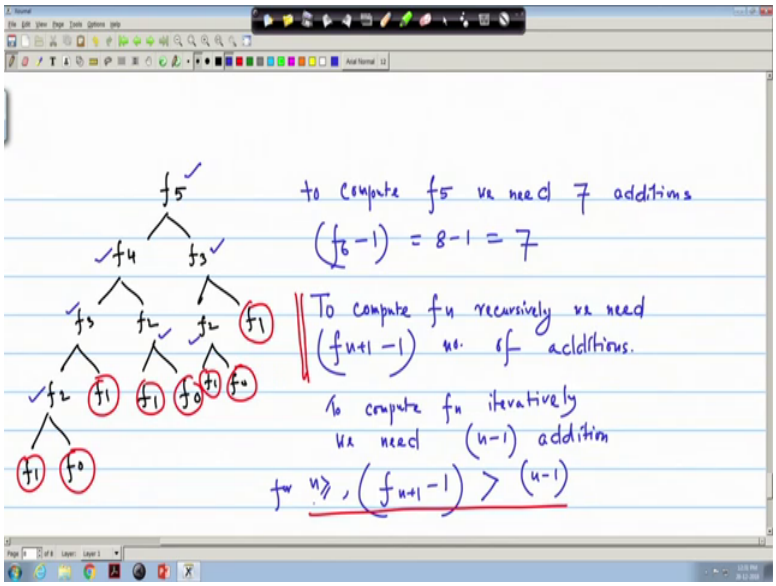
Now, but at each stage of these say like f 2 here f 3 here I need a addition because it is f 2 is f 0 plus f 1 I know because my f 2 is f 2 is f 0 plus f 1. So, for these case I need 1 2 3 4

I need some for addition. Now, if I quickly remember the numbers if I write the numbers that f 0 equal to 0 f 1 equal to 1 f 2 equal to 2 f 3 equal to 3 f 4 equal to 5 f 5 equal to 8 and so on.

Now, see when we are computing f 4 when we are computing f 4 I need to compute f 4 we need 4 additions that we can write to compute; that means, it are 4 this 4 I can write as say f 4 minus 1 because, f 4 is 5 f 0 is 0 oh sorry I have this is it we known this is this sequences f 0 is 0 f 1 is 1 f 2 is 0 plus 1 f 3 is 2 f 4 is 2 plus 1 3 and f 5 is 5 and so on.

So, I have 4 when we are computing this becomes f 5 minus 1. So, this becomes when we are computing this becomes if 5 minus 1 for addition. So, in general we can write to compute; to compute f n we need f n plus 1 minus 1 number of additions if we quickly do the for f 5.

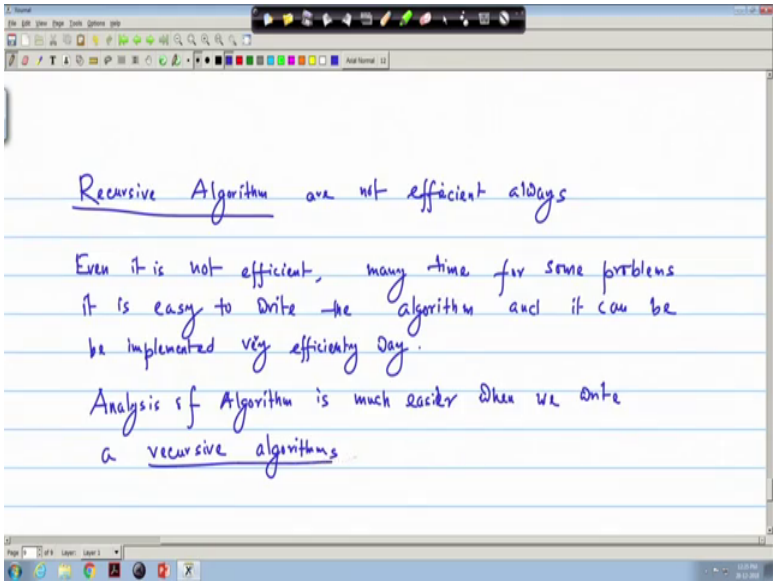
(Refer Slide Time: 35:07)




Then this becomes f 4 f 3 f 3 f 2 f 2 f 1 f 1 f 0 here it is f 1 f 0 this is becomes f 2 f 1 f 1 f 0. So, here also we see we will stop computing when this becomes f 0 or f 1 f 1 f 0. So, in this case I need 1 2 3 4 5 6 7. So, to compute f 5 we need 7 additions; that means, which is 7 is equal to f 6 minus 1 is 8 minus 1 is 7. So, just now what we have seen that to compute a pen recursively to compute fn the Fibonacci number recursively we need f n plus 1 minus 1 number of additions ok.

Now, if we write a function where we want to find out the Fibonacci number iteratively; that means, to compute fn iteratively we need simply n minus 1 addition. So, for n greater than equal to 1 always that f n plus 1 minus 1 is much greater than n minus 1. So, what we conclude from here; that means, we conclude that not always the recursive functions are good that means it is not efficient.

(Refer Slide Time: 38:07)




So, we can conclude that recursive algorithms recursive algorithms are not efficient always; are not efficient always because, for this problem of computing Fibonacci number we see that iterative program better than the recursive program. But, we will use recursive algorithms because many times these are f not only efficient it is implemented much easier way; that means, we can write the code even if it is not efficient. So, even it is not efficient many time for some problems it is easy to write the code or to write the algorithms not only that and it can be implemented very easily efficient way although the complexity is not good.

So, that is why the recursive algorithm is very important in computer science and for a set of problems actually only recursive algorithm is the solution. So, we will see that how we can find out this efficiency of the complexity when we write that the recursive algorithm. That means that analysis of algorithm is analysis of algorithm is much easier when we write a recursive program, when we write a recursive problem or recursive algorithm recursive algorithm.

So, in the next class or next lectures we will see that how we can using the recursive algorithms we can do that thing.