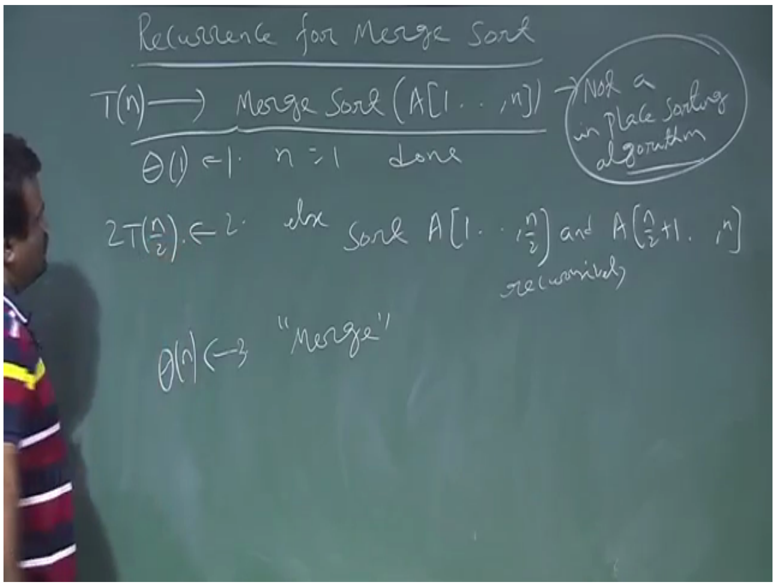
**An Introduction to Algorithms**
**Prof. Sourav Mukhopadhyay**
**Department of Mathematics**
**Indian Institute of Technology, Kharagpur**

**Lecture – 04**
**Recurrence For Merge Sort**

(Refer Slide Time: 00:25)



Recurrence for Merge Sort

$T(n) \longrightarrow$ Merge Sort $(A[1..,n])$ $\rightarrow$ Not a in place Sorting algorithm

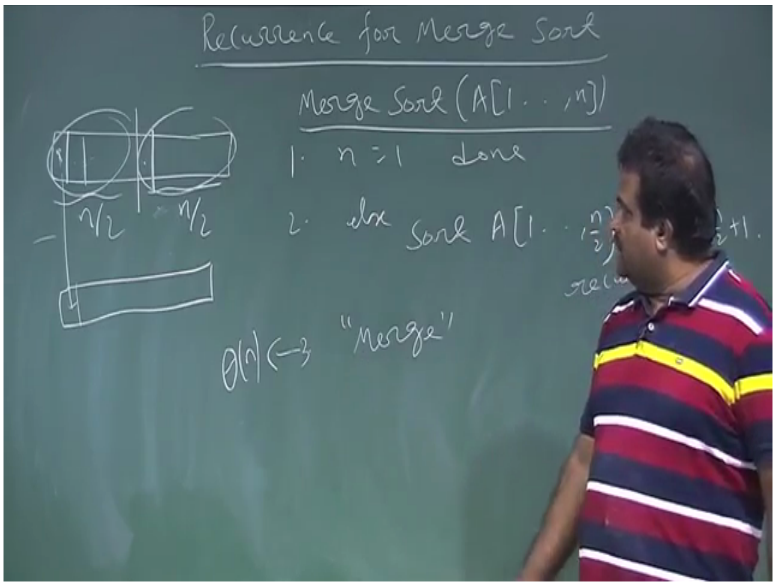
$\Theta(1) \leftarrow 1.$ $n = 1$ done

$2T(\frac{n}{2}) \leftarrow 2.$ else Sort $A[1..,\frac{n}{2}]$ and $A[\frac{n}{2}+1.,n]$ recursively

$\Theta(n) \leftarrow 3$ "Merge"

So we talk about the time complexity of the merge sort or the analysis of the merge sort. So, just to recap in a merge sort what we are doing we have this we have a array of size n and then we if n is 1 we return we are done otherwise else we recursively sort this sort this sub arrays 1 to n by n by 2 and a n by 2 plus 1 to n recursively by calling the same function merge sort and then we call the merge.

(Refer Slide Time: 01:10)


Recurrence for Merge Sort
Merge Sort (A[1 . . , n])
1. n = 1 done
2. else Sort A[1 . . , n/2]
recur
n/2
n/2
θ(n) ⟵ "Merge"

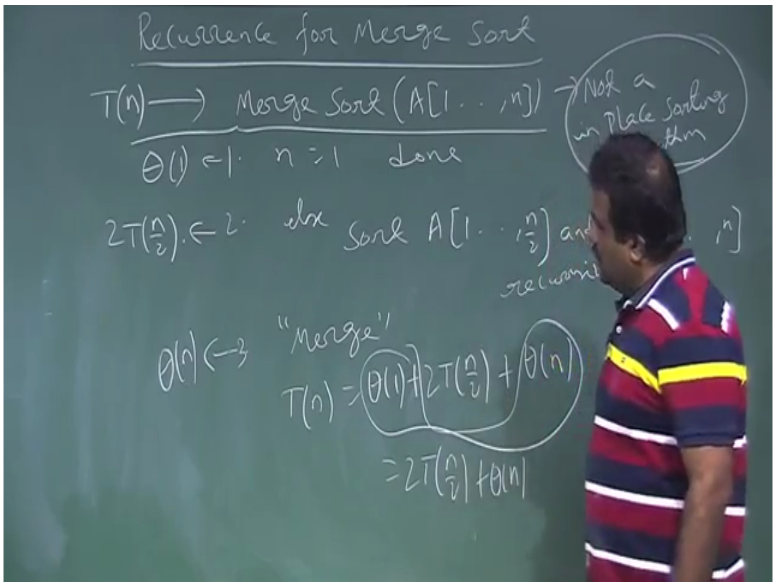
So, basically we have this array now we partition into 2 part this is n by 2 this is n by 2 I mean if n is even otherwise it will be lower in a upper in a we sort this sub array this sub array recursively and then once we got the 2 sorted sub array once we got 2 sorted sub array this is sorted this is sorted we call merge; merge function; merge sub routine to get a sorted array. So, for merge what we are doing we are taking the minimum. So, minimum will be here in this sub array minimum will be here. So, for this merge we need to take help of extra array. So, this is not in place sorting algorithm for this merge sub routine we need to take a help of a extra array.

If we want this merge to be in linear time. So, why linear we will come to that. So, we compare these 2 whichever is the minimum we will put it here and then we point to the next one like this we compare like this. So, this is basically time complexity for this is order of n this is linear time because we are just reading this every time we are returning we are outputting 1-1 element. So, there are n element total n by 2 n by 2. So, basically we are spending linear time for that.

So, and, but this for this merge we need to take help of a extra array to store this element otherwise we cannot store here in a linear time not possible to store this in a linear time the merge can be done in the same array that is not possible. So, so this is not a in place sort. So, this is not in place sorting algorithm; algorithm not a in place sorting algorithm. So, this you must. So, this is the thing.

Now, now we want to know the time complexity for this suppose T n is the time to sort n element now how much time we are spending here; here we are spending theta 1 time now this is theta of n now how much time we are spending here. So, we have a 2 call 2 call of a same size n by 2 n by 2 of the same merge. So, it will be basically 2 T n by 2. So, the T n will be sum of this.

(Refer Slide Time: 03:52)



Recurrence for Merge Sort

$T(n) \longrightarrow$ Merge Sort $(A[1 .. , n]) \rightarrow$ Not a in Place Sorting thm

$\Theta(1) \leftarrow 1.$ $n = 1$ done

$2T(\frac{n}{2}). \leftarrow 2.$ else Sort $A[1 .. , \frac{n}{2}]$ an $, n]$ recursi

$\Theta(n) \leftarrow 3$ "Merge"

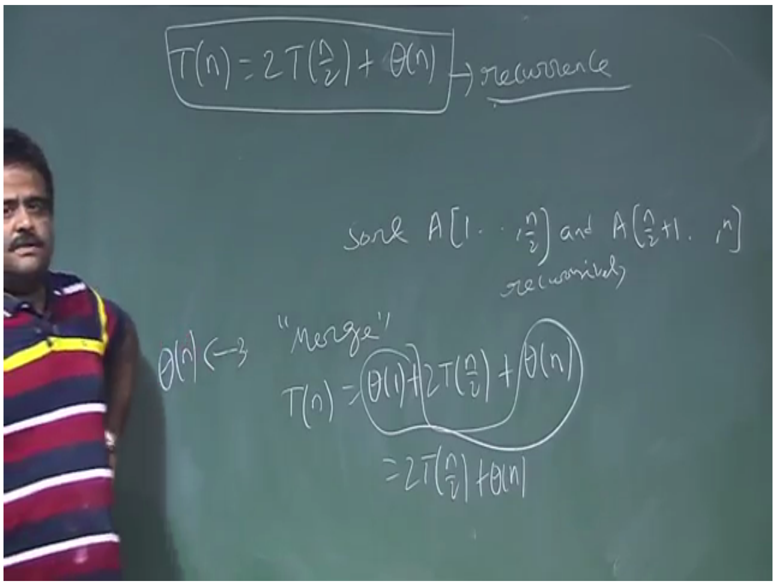
$T(n) = \Theta(1) + 2T(\frac{n}{2}) + \Theta(n)$

$= 2T(\frac{n}{2}) + \Theta(n)$

So, the T n will be basically theta 1 plus 2 T n by 2 plus theta of n.

So, this theta of n and theta all will combine. So, this is basically 2 T n by 2 plus theta of n. So, this is basically the time complex this is basically recurrence related to this is the function related to time complexity. So, this is function is called recurrence function.

(Refer Slide Time: 04:34)

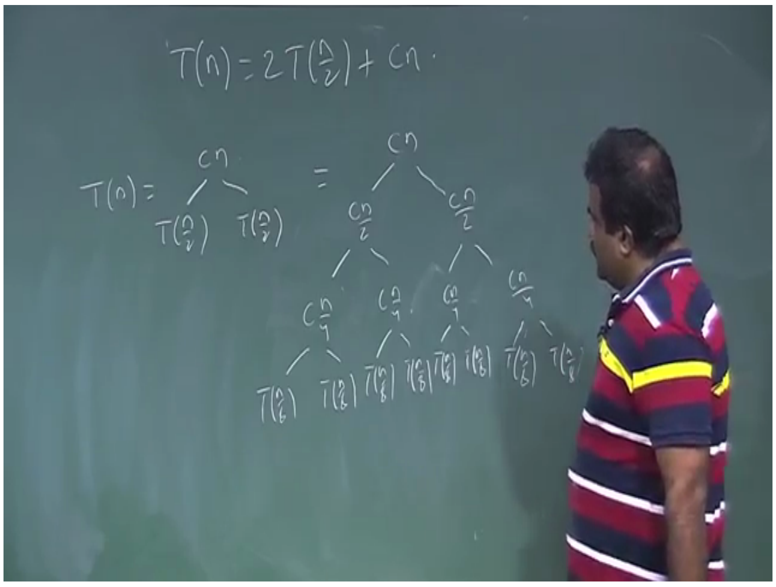

$\boxed{T(n) = 2T(\frac{n}{2}) + \theta(n)} \to$ recurrence
sort $A[1 \ldots \frac{n}{2}]$ and $A[\frac{n}{2}+1 \ldots n]$
recursively
$\theta(n) \leftarrow$ "Merge"
$T(n) = \theta(1) + 2T(\frac{n}{2}) + \theta(n)$
$= 2T(\frac{n}{2}) + \theta(n)$

So, this is called recurrence of this algorithm. So, we got this recurrence. So, basically we got T n is equal to 2 T n by 2 plus theta of n or some c of n theta of n.

So, this is what is called recurrence or recurrence relation now the question is how to solve this recurrence we need to get the solution we are not happy with this type of expression we want to know whether merge sort is a order of n square algorithm or order of n log n algorithm or order of n cube algorithm or order of n algorithm. So, that is of our interest we do not; so, we do not happy with this recurrence we need to have the solution of this recurrence. So, we need to solve this recurrence. So, now, the question is how to get the solution of this recurrence.

So, to solve this recurrence we can use what is called recursive (Refer Time: 05:41) effort. So, basically you are going to solve this like theta n we can get just c of n some constant n.

(Refer Slide Time: 05:49)



So, basically what this is basically we have a array of size n we divide the array into 2 sub arrays and then we call the merge and then we recursively sort these 2 sub array then we call merge to get the solution of the whole array.

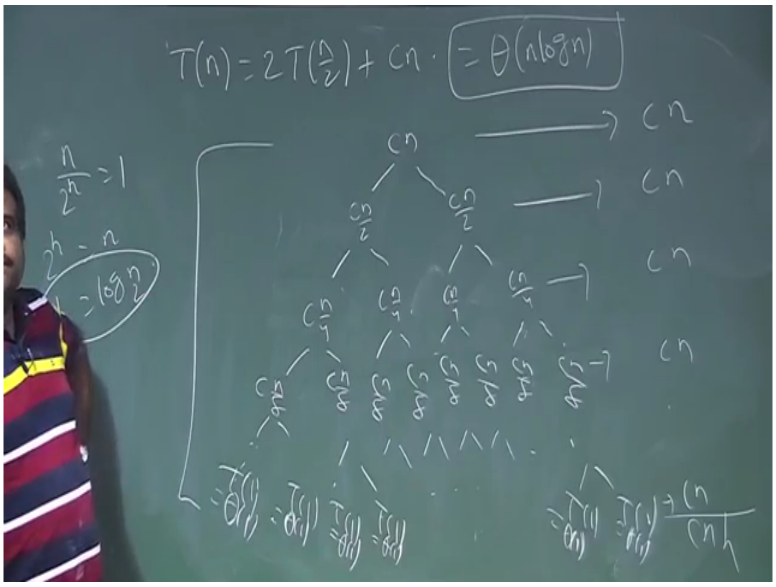
So, this is our T n. So, T n is basically can be write as c n T n by 2 T n by 2. So, basically we have a problem of size n this is what is called divide and conquer technique. So, this is a problem of size n we divide the problem into 2 sub problems and then we recursively solve this sub problems this sub problems and this will again take T of n by 2 T of n by 2 and then once we have the solution of this 2 sub problems we merge this with a cost of c n that is called merge in the merge sort. So, that is the c n.

So, this total time is basically sum of this now again this is a problems physically this is a sub problems, but this is again a problems of size n by 2 again we can further divide it into the sub problems sub problems. So, c n; so, this will be again can be c n by 2 T n by 4 T n by 4 similarly c n by 2 T n by 4 T n by 4 because this sub problems of size n by 2 now again we further divide this when we call the merge sort on this sub problems on the sub array on this sub array on the sub array we again call the merge sort. So, it will divide into sub array. So, like this we continue until we reach to the size one array then we merge come back and like this.

So, again we further divide this into c n by 4 T n by 8 T n by 8 this is the merge cost c n by 4 T n by 8 T n by 8 this is c n by 4 T n by 8 T n by 8 this is basically c n by 4 T n by 8

T n by eight. So, this way we will continue and the; our time complexity is sum of all this. So, this way we will continue and we stop when it will reach to the when n will be n because if it is 1 then we cannot further divide into sub problems.

(Refer Slide Time: 08:40)

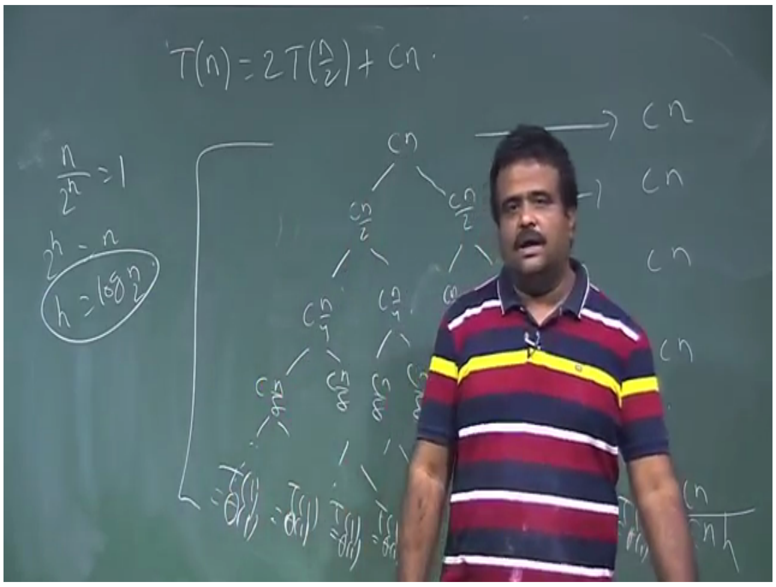


So, we each branch will stop. So, this is c n by 4 all will be c n by 4 c n by sorry, c n by 8 c n by 8 c n by 8 c n by 8 c n by 8 c n by 8 c n by 8 c n by 8 like this and all branch will stop at T 1 T 1 T 1 like this all branch will stop at T 1 because also we reach to the T 1; that means, there is the size of the problems that sub problems is of size 1 then we cannot reduce it further we must stop their and that T 1 will take theta of 1 time because we have to. So, if we have to sort all the 1 element if we have to solve only a array of 1 that is already sorted. So, theta of 1, so, all the T 1 is basically theta of 1 because if our problems size is 1 then nothing to be done it is already solved.

So, this is basically all the branches will come like this and the all the branches will stop this now our time complexity is the sum of this nodes basically. So, now, the question is how to get the sum of these nodes. So, T n is basically sum of all this. So, how to get the sum we can get the sum level wise we can get the sum of the levels and then the level of the sum. So, if you do that what is the sum of this level c n what is the sum at this level c n what is the sum at this level c n if we just see it is all the sum is c n. So, sum at all level is c n.

Now, what is the height of the tree? So, now, what is the sum of this level c of n into h h is the height of the tree now what is this height of the tree now how to get height of the tree if you observe here this is n by 2 this is n by 2 square. So, if the height is h then this is n by 2 cube like this it is coming down. So, if the height is h then n by 2 to the power h is basically coming to be 1 we stop at T 1.

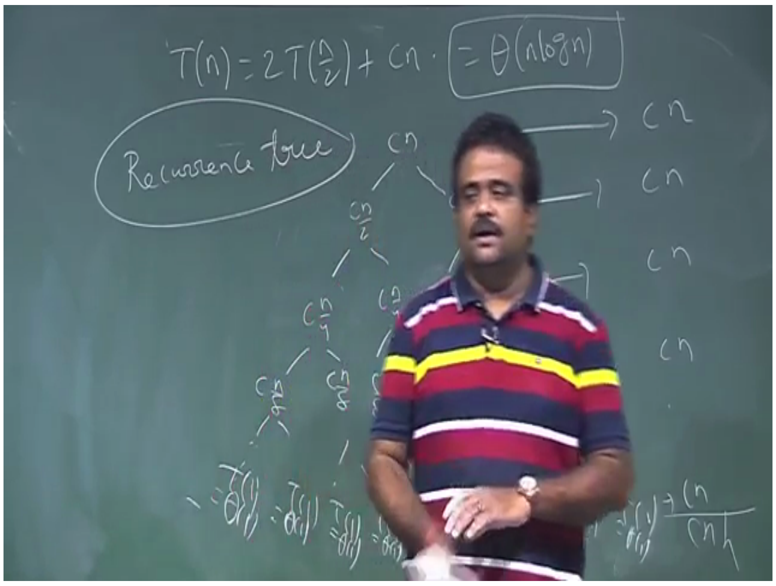
(Refer Slide Time: 11:01)



So, from here we can say 2 to the power h is n. So, h is log n base 2. So, height is height is log n.

So, this is basically c n log n. So, this is basically solution of this is n log n. So, this is the. So, this is the time complexity for merge sort. So, this is what is called recursive tree. So, this method is called recursive tree method this is called recurrence tree or recursive tree.

(Refer Slide Time: 11:47)

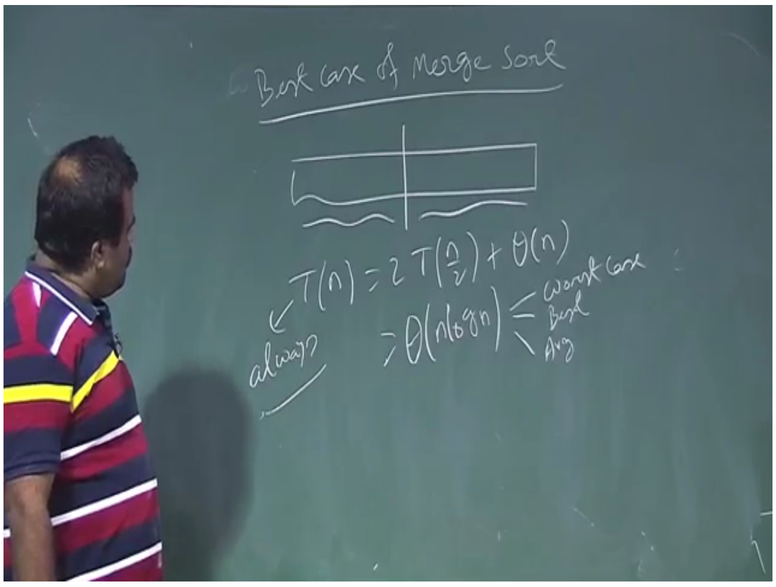

$T(n) = 2T(\frac{n}{2}) + cn \cdot \boxed{= \theta(n \log n)}$
Recurrence tree
cn
cn
cn
cn

So, recurrence tree or recursive tree; so, this is one way we can try to get the solution there are some method to solve the recurrence what is called substitution method that is the inductive method in the method of induction we prove that because see in this here what is the guarantee that at i th level it will be c n we are just using our incentives on that it is going like c n c n c n. So, at the 10 levels also it will be c n.

But again what is the guarantee that it at the 10 level it is c n. So, we need a proof that proof we are not doing. So, that is why this method is not a full prove method. So, this method is. So, we have a full prove method which is called substitution method to solve the recurrence where we will see by the method of induction we can prove that our solution is this. So, this is the merge sort time complexity. So, now, what is what is worst case for merge sort what is the best case what is the average case if we want to do.

So, can you tell me the worst case of the merge sort or what is the best case of the merge sort what is the best case of the merge sort or what is the. So, what is the best case of the merge sort?

(Refer Slide Time: 13:16)

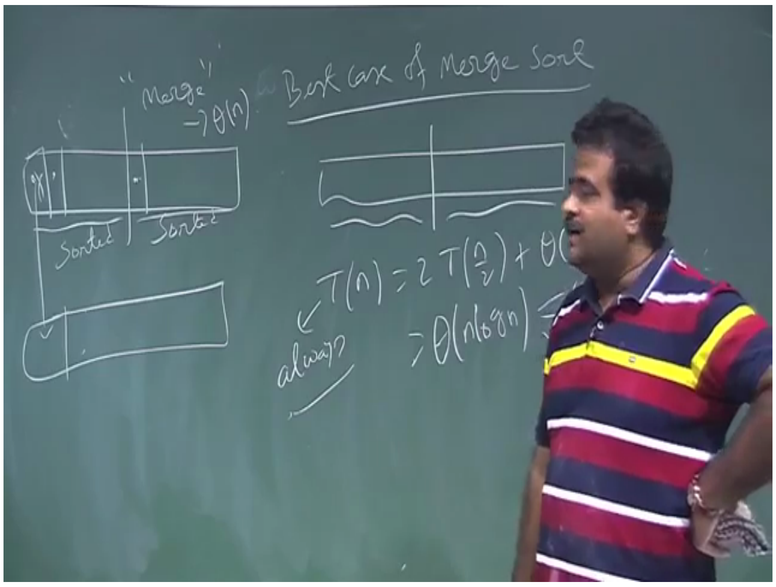


So, you want to do the type of analysis of merge sort. So, like in insertion sort we will see that if the input is sorted it is best case, but is this; the same is here. So, if the input is sorted in the merge sort we need make any difference if the input is are equal element will it make any difference whatever input may be we are not caring we are just going to the middle we are dividing the array into 2 sub array we sort them we sort them once we get the solution then we merge them.

So, you we really does not bother about the input we are just going to the middle of the array we are dividing it we are sorting this part we are sorting this part then we have merging that is it. So, it is always recurrence will be like this for any input 2 n by 2 plus theta one always for all type of analysis this is the always the recurrence will be like this. So, the solution is always this; this is for all the cases worst case there is because all the cases are same worst case best case and average case also because we really does not bother about the input pattern we are just divide it the array into 2 sub arrays then we sort this sub array recursively we sort this sub array recursively then we call the merge. So, thus this will be the recurrence in any input for any input.

So, that is the case now we talk about so, but the this is this is a l log n time algorithm whereas, worst case where as the insertion sort is order of n square algorithm so, but whichever is the better insertion sort or the merge sort insertion that is a extra advantage in the sense it is a in place sort we do not need the extra array to sort it, but merge sort is not in place for that merge we have a sorted array. So, this is this is sorted this is sorted after calling the recursive call this is sorted.

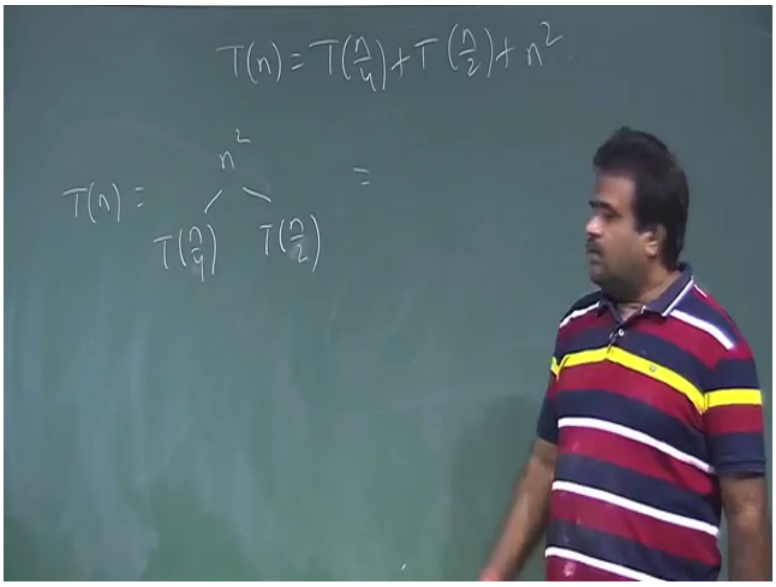
(Refer Slide Time: 15:30)



Now, this is now we will need to call merge. So, merge is merge sub routine. So, then we need to take help of extra array. So, where what we are doing we are taking the list element of this compare the least 2 element whichever is the least 2 among this we are outputting their then we have comparing the next least like this we are doing. So, this is the merge sub routine. So, for that we need to and merge we want to do in linear time. So, for that we need to we take help of a extra auxiliary array of size n otherwise we cannot make it in linear time if we have to handle this in the same given array this is our a array. So, this has to be clear. So, that is why merge sort is not in place sorting algorithm.

So, now we will take another example of recurrence where we will use the recursive tree method to get the solution. So, that is basically.

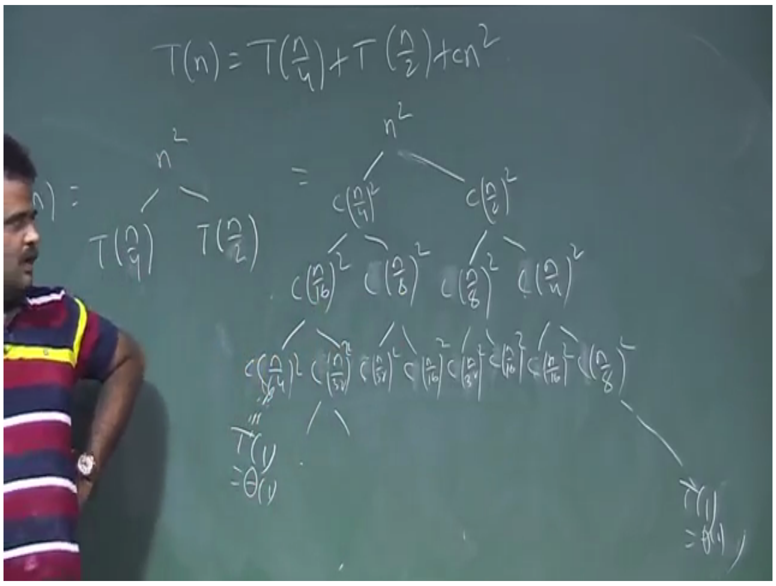
(Refer Slide Time: 16:59)


$T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + n^2$
$T(n) =$
$n^2$
$T(\frac{n}{4})$ $T(\frac{n}{2})$
$=$

So, let us have another recurrence like suppose we have a recurrence like this T n is equal to T n by 4 plus T n by 2 plus n square suppose we have this recurrence and we want to. So, I found we can get this recurrence again this can be generated by a algorithm which is again divide and conquer in nature. So, we have a problems of size n we divide the problems into 2 sub problems and here they are not equal size one sub problems size is n by 4 another sun problems size is n by 2 then once you have the solution of this 2 sub problems we combine this 2 sub problems with a cost of order of n square and this is basically combine step or merge step.

So, this type of recurrence we can get from such a divide and conquer now the question is how to solve this recurrence. So, we will again use the recursive tree method to solve this. So, T n is basically. So, we can write this as this T n by 2 T n by 4 like this then again this is a problems of size n by 2 and then again we can divide sorry this is say n by 4 and this is n by 2. So, again this is a problem of size n by 4. So, we can again divide into sub problems.

(Refer Slide Time: 18:38)



So, then this is basically n square and this will be basically c. So, now, our size is n by four. So, we put the same recurrence we are assuming the in the subsequence steps also we are having the same recurrence.

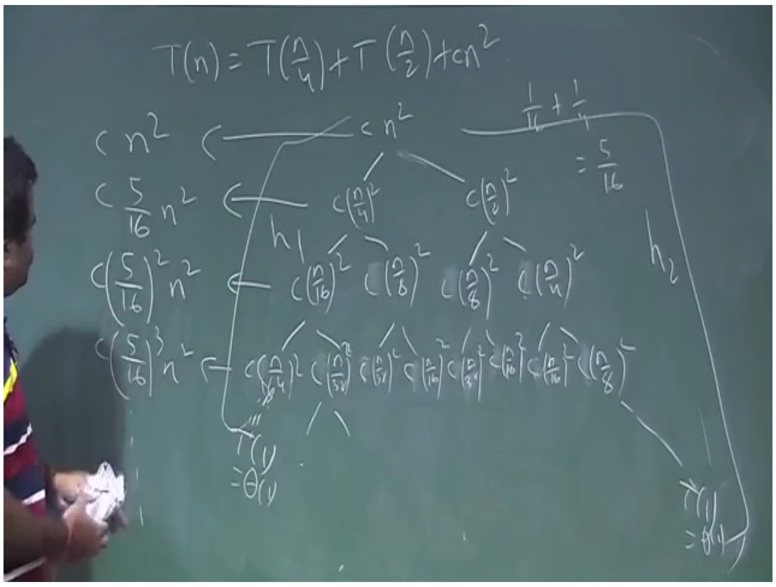
So, it will be c oh; it do not have c here, we can put a c over here c is a constant if it is theta of n square we can put c of n square, so, c of n by 4 square. So, that is basically n by 4 square you can just write in this way then it will divide into this T of n by 16 and then this is T of n by 8 and similarly here we can have. So, this is n by two. So, c of n by 2 square and this is again we put n is equal to n by 2, so, T of n by 8 T of n by 4.

So, basically here further we are dividing into sub problems; sub problems with the same recurrence. So, then again we divide into sub sub problems. So, this is of size this is of size n by 16. So, if we divide it into sub sub problems it will be T of n by 64 and this is T of n by 32 and this will be again T of n by 32 and this will be again T of n by 16 and this will be again T of n by 16 sorry, T of n by 32 and this is T of n by 16 and this is T of n by 16 and T of n by 8 and this will be c of that square c of this square c of this square c of this square and we will continue. So, which branch will n first. So, this is going rapidly. So, this will end. So, each of this branch will end with T 1.

So, each of this branch will end T 1. So, this will end first than this one. So, this will be going little longer. So, everybody will end at the at the point T 1 everybody will end at the point T 1 so, but this branch will end first because its going rapidly to this and this is will be c of that square c of that square c of that square c of that square c of that square c of that square c of that square like this.

So, every branch will be ending with T 1, T 1 is means theta 1 like this theta T 1 is theta 1. So, every branch is will be ending at theta 1. So, our time complexity is basically sum of the all nodes, so, now, how to get the sum. So, we will get the sum level wise the; what is this level sum.

(Refer Slide Time: 22:01)

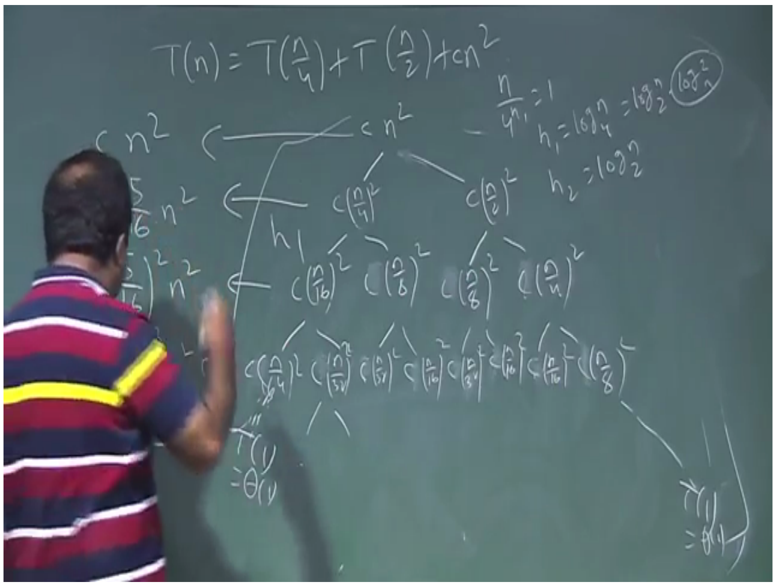


This is n square. So, what is this level sum. So, this is basically what 1 by 16 plus 1 by 4. So, this is 16. So, 5 by 16. So, this is basically c of that. So, this is c c of 5 by 16 n square ok

Now, if you just calculate this will be getting c of 5 by 16 square n square coming if we are if we just we can verify this then this will be c of 5 by 16 n cube like this. So, this way it will be coming. So, this way it will be coming now. So, again here we are using our intuition what intuition that in case it like this. So, what is the at the i th level we are thinking that it will be 5 by 16 to the power i, now what is that prove that proof is not we are doing here. So, so either we have to proof that by method of induction or something else then we can say this is a probable method so, but we are not doing we are just using the intuition we are no presence to go up to this level after this, but we are using our intuition it is going like this. So, it will go like this.

So, this is the way it is going. So, now, if the height of this is the h 1 and the height of the bigger tree is h 2 then we can say this sum then we can say. So, up to h 1, so, h 1 is the earlier ending the tree. So, up to h 1 it is bounding like this. So, up to h 1 it is a complete binary tree and up to h 2 it is the complete. So, basically it is bounded by sum of this up to h 1 left side and bounded by sum of that up to h 2 now we want to take the sum of this term. So, what is the sum of this term and what is the height basically; what is h 1. So, we want to get h 1 h 1 is basically; so, h 1 is basically what. So, this is 4 this is 4 square this is like this, so, 4 16, so, 16 into 4, so, 4 4 plus 4 cube.

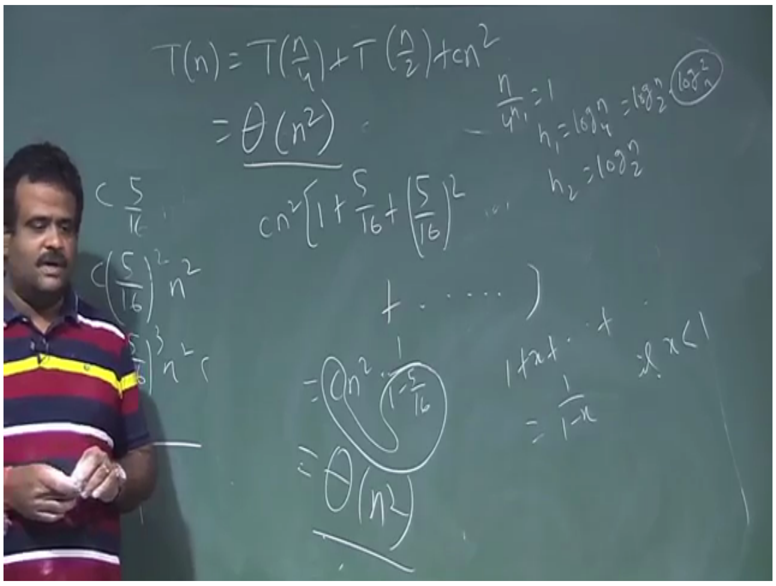
(Refer Slide Time: 24:57)



So, basically n by 4 to the power h 1 is equal to 1. So, h 1 is equal to basically log n base 4 similarly here also this is basically 2. So, h 2 is basically log n base 2, but this again we can make it into base 2.

So, log 2 base n. So, this is a constant. So, height is a order of log n. So, height is order of log n. So, now, we need to take the sum of this and that will be the solution, so, how to get the sum of this? So, basically, so, basically we have this sum. So, it is basically c c n square into 1 plus 5 by 16 plus 5 by 16 square plus dot, dot, dot, so on.

(Refer Slide Time: 25:43)


$T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + cn^2$
$= \Theta(n^2)$
$\frac{n}{4^{h_1}} = 1$
$h_1 = \log_4 n$
$h_2 = \log_2 n$
$c\frac{5}{16}$
$c(\frac{5}{16})^2 n^2$
$cn^2[1 + \frac{5}{16} + (\frac{5}{16})^2 \ldots + \ldots\ldots)$
$1 + x + \cdots +$
$= \frac{1}{1-x}$ if $x < 1$
$= cn^2 \cdot \frac{1}{1-\frac{5}{16}}$
$= \Theta(n^2)$

Now, this is a finite times, but if we want to make it infinite this would be less than and now this is basically c of n square this is basically this series power series 1 plus x plus x square like this.

Now, this is basically 1 by 1 minus x if x is less than 1. So, this is basically 1 by 1 minus 5 by 16. So, this is again a constant merge with this. So, this is basically theta of n square this solution of this is theta of n square. So, this is basically theta of n square, but again this is not a probable method because this recursive method we are not we are assuming all level it is going like this. So, it will be 5 by 16 to the power i at the i level, but that proof we are not doing.

So, but this is giving us a idea what could be the solution for the recurrence so, but in the next class we will talk about probable method which is the substitution method where will take help of the induction proof method proof method of induction and then we will see that, but there we have to assume our guess we are given a solution we have given the recurrence what we have assume the solution then we have to proof the solution, so, that we will do in the next class.

Thank you.