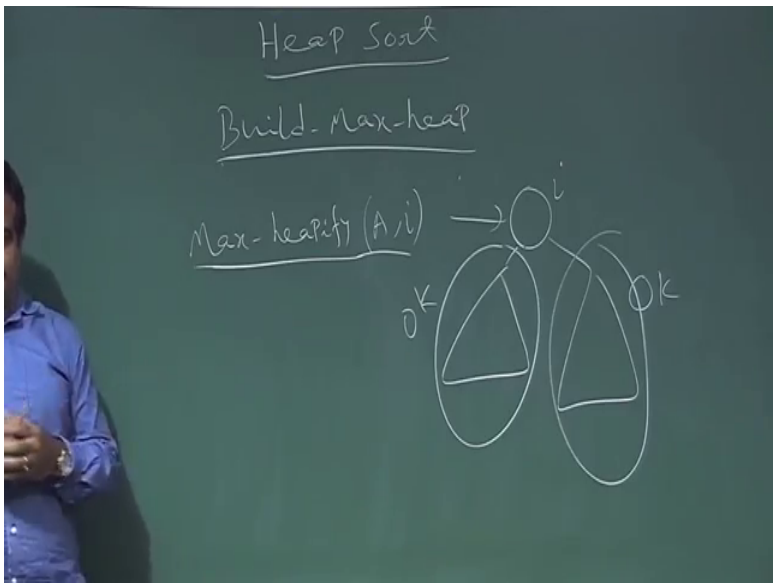
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 14
Heap Sort

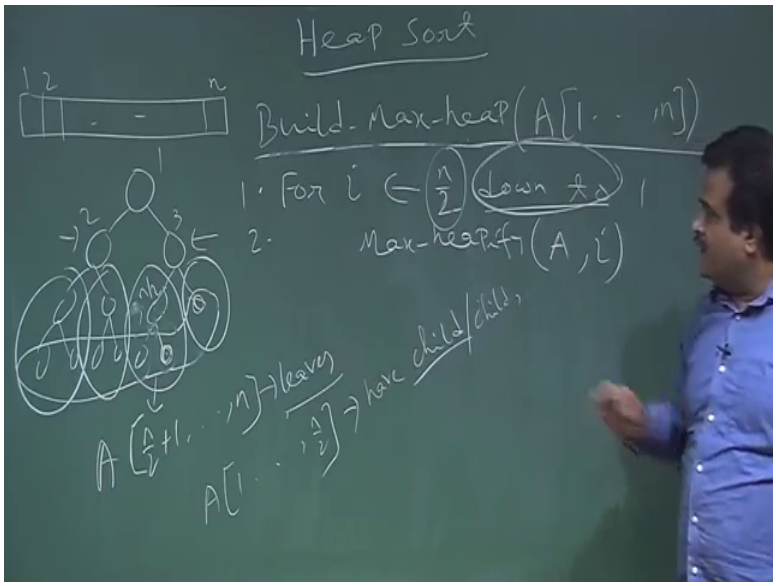
(Refer Slide Time: 00:26)



So we talk we are going to talk the build max heap so; that means, so far we have seen the max heapify sub routine. So, this is a sub routine where we call this at i. So, i th node. So, for that we need to have a assumption that everything is ok here and everything is ok here in the left sub array right sub array and only we have a violation over here. So, then we call the max heapify at this point.

So, this will exchange with the maximum of this 2 child and then again we may need to call the max heapify on the exchange node because that may violate the max heapify property of max heap property of that. So, this way we continue. So, this code; we have see that is the max heapify code. So, now, we will talk about how we can use this max heapify to build a max heap array we have given a unordered array. So, how to build a max heap array?

(Refer Slide Time: 01:45)



So, that is our next operation which is called build max heap. So, this is basically taking an.

So, the input is an array which is basically 1 to n an unordered array and it will output will be a max heap array. So, this code is very simple code this is for i n by 2 down to 1 and then we call the max heapify this is for this is for loop for i n by 2 down to 1 then we call max heapify. So, this is the code this is the code for build; build max heap, very simple code only 2 2 line code pseudo code.

So, now this is from n by 2 down to 1. So, why it is n by 2 down to 1 that we will see. So, why not n plus 1 to n; so, basically we want to see who are the element who are the element from n by 2 n by 2 plus 1 to n. So, suppose we have array say we have given a array of size n. So, this is our array it is starting from 1 to up to n. So, if you draw the tree. So, this is 1 2 3 like this. So, this is like this, like this.

So, suppose this is the tree corresponding to this array now. So, this leaves are basically these are the leaves; leaves are basically nodes from n by 2 plus 1 to n. So, these are basically leaves these are basically leaf node because they have no child because if they have to have child then the child will be. So, if this node has a child then the child will be a 100 2 i 2 plus 1. So, 2 i is basically n plus 2 there is no n plus 2 or n plus so; that means, these are basically leaf nodes.

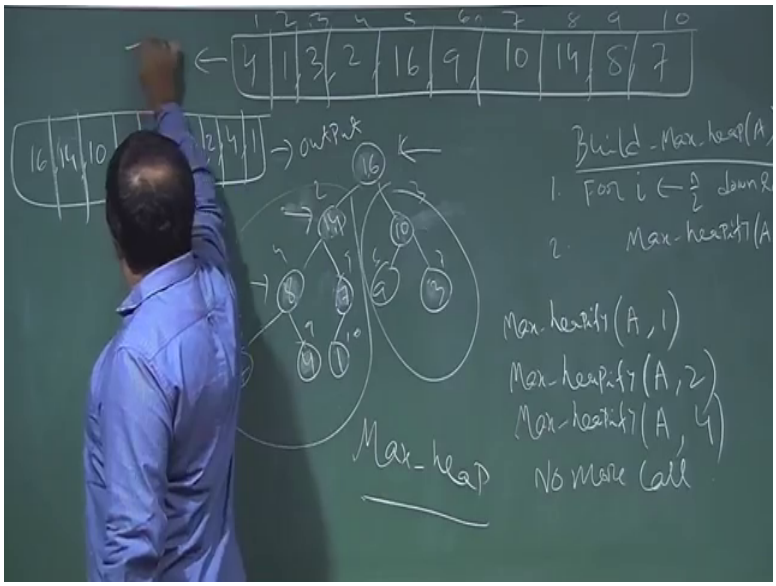
So, but n by 2 has a child because n by 2 has if we 2 i. So, n by 2 has child at n. So, i is equal to n by 2 child has a, so the nodes n by 2. So, 1 up to n by 2 they have the child all the nodes they have childs, child or childs. So, that is the reason and if a node is leaf node it is already max heapify leaf node means they have no child. So, this is already max heapify they are they are already max heapify. So, we do not need to call the max heapify for those.

So, that is why we start this loop from n by 2 down to 1. So, we will just start it from this node. So, this is the node n by 2 th node. So, we call the max heapify here because other after that these are the old leaf nodes for this nodes; this is already max heapify because these are the node these are the node only single node single root there is no child. So, only we may have violation over here. So, that is why w call this loop from n by 2 down to 1 why down to 1 because if we call n by 2. So, we may have a violation over here so; that means, it may be, but every time.

So, will call max heapify for max heapify we need to have the assumption that left part is right part is. So, that assumption is there because this is the left part this is already max heapify this is already max heapify. So, we call the max heapify here. So, it may only get only max heapify can only handle only one violation. So, that way again we call this like this. So, when we reach here we know that this part we already fixed this part we already fix like this.

So, when we reach here we know this is already fix this is already fix we may got a violation here. So, that we can fix by this calling max heapify like this. So, that is why it is down to 1. So, this way we will keep on fixing and we reach to the root. So, this ultimately will be a max heap array of this. So, will take an example, this is the basically the code will come to the time complexity of this, but before that let us take an example to execute this code. So, let us take an array this could be any array could be a input. So, let us take an array.

(Refer Slide Time: 06:53)



Suppose we have this array say 4 1 3 2 16 9 10 14 8 7 suppose this is our given array this is the input and we want to make this array as a max heap array, so, now, we just from we just this is our heap. So, we just view this heap as a data as a tree. So, this is 4 this is 1 this is 3 this is 2 this is 16 and this is 9 10 and this is 14 this is 8 and this is 7. So, this is the way we view this as a tree now we call this build max heap.

So, just to build max heap, so this is the code for build max heap. So, for i n by 2 down to 1 here n is equal to 10. So, we have how many element 1 2 3 4 5 6 7 8 9 10 n is equal to 10. So, n by 2 5, this is 1 2 3 4 5 6 7 8 9 10. So, and then we call the max heapify a comma i. So, basically we these are all leaf nodes these are all leaf nodes. So, this is the first node which is not a leaf node it has its child. So, this is the n by 2 th node. So, we start calling the max heapify from this node.

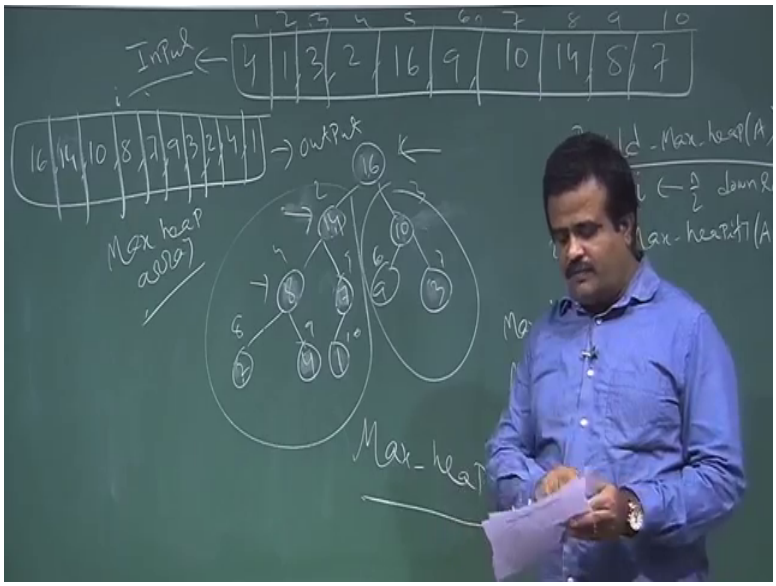
Because after that every nodes are leaf node which is already max heapify. So, no need to call. So, we just start calling the max heapify from here. So, we call max heapify from here we check with this. So, this is get a. So, this is max heapify. So, then we call the max heapify, this is the loop in 2. So, then we once we call the max heapify it will exchange the maximum. So, this 14 will come here 2 will come here then no more call then we start with here. So, 3 will exchange with ten. So, this 10 will come here 3 will be here then we this is our i. So, this will exchange with 16.

So, 16 will come here and 1 will come here again we have to call then 7 will come here and 1 will be here and then finally, we call this at here now when we calling this if we observe everything is fine over here everything is fine over here that is why we are going down to 1 because we are keep on fixing and going to ensure that max heapify when you call max heapify we need to have the assumption that left sub array is right sub array is.

Then only we can call this max heapify. So, we call this max heapify here. So, what it will do now it will exchange this maximum with maximum child. So, 16 is maximum. So, 4 will come here. So, 16, this is the call this is the max heapify call a on 1. So, then it will exchange with this then we have to again call the max heapify because it may violate a with 2. Now again it will exchange with the maximum 4 will come here 14 will come here. So, again we have to call the max heapify a comma 4 index 4. So, again it will exchange with maximum.

So, 8 will come here 4 will come here then this is the leaf. So, no more call no more call. So, this is the max heap array. So, then I exhaust. So, these array is a max heap array. So, our array is change now this is the input and the output array will be just 16 14 10 16 14 10 then 8 7 9 3 2 4 1. So, this is the, this is the output.

(Refer Slide Time: 11:56)

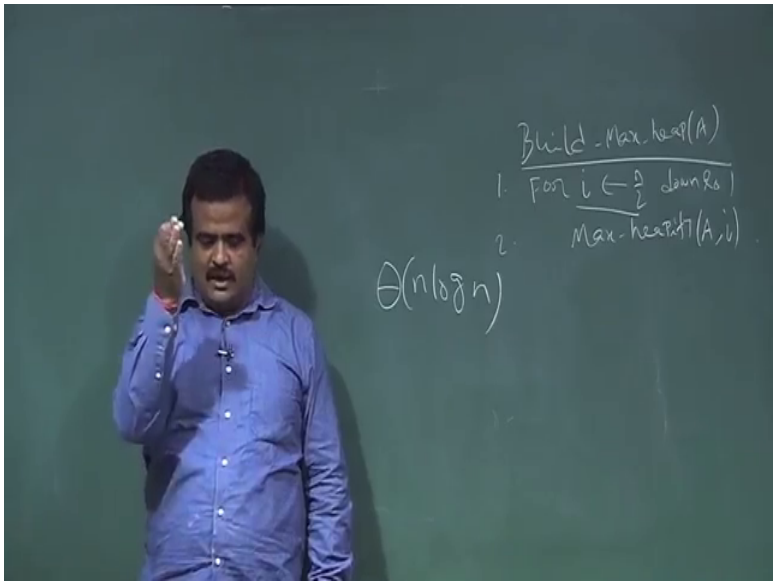


This is the output after build max heap and this is the input this is the input array. So, this is unordered array which is not a max heap array.

But we can make this array to be max heap array and this 3 we are viewing this is we are our visualization in reality this 3 does not exist we do not have pointed to implement this tree to view this tree we know that if a node is i then the child will be i plus i 2 i 2 i plus 1. So, that is the way we are viewing this tree. So, this is a max heap array because if we check any node i it is greater than from his child. So, any node i is greater than from 2 i 2 i plus 1 any node this is the array if we take any node i then it is greater than 2 i 2 i plus 1.

So, that is why this heap is a this is a max heap array and this is the output of this build max heap now the question is how what is the time complexity of this build max heap. So, that we have to see. So, that we have to calculate the time complexity. So, how much time we are spending in build max heap. So, so what is the intuition? So, it is for loop it is for loop of size n and each time we are calling the max heap now.

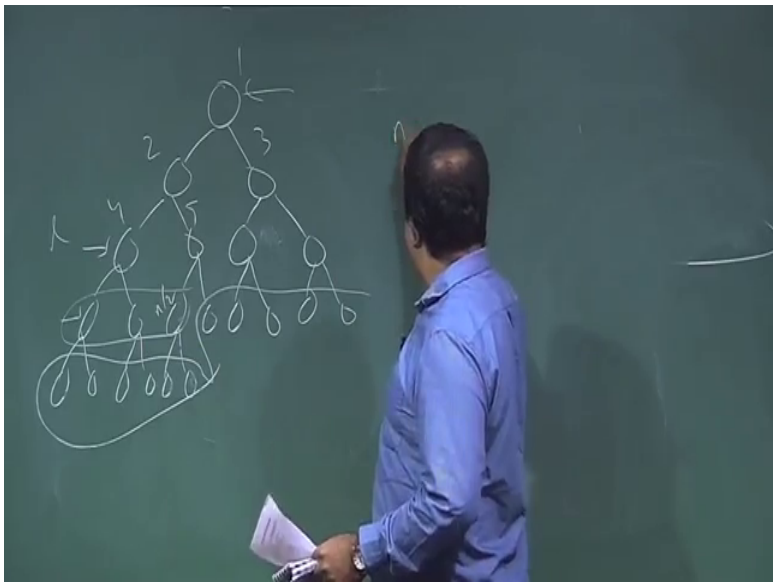
(Refer Slide Time: 13:23)



So, intuitively it should be order of n log n because each time we are calling max heap and that max heap can be again further call an consist of further call this is a recursive can come because it may fix there and come down then again. So, maximum it can go to the root height of the tree. So, that is why it is n log n, but this is the intuition, but we really it will take order of n. So, that analysis we will do.

So, it is not really n log n because we are not this max heap call when you are doing it is really not going to the height of the tree for each node why suppose this is our array 2 3 4 5 say 6 7 like this.

(Refer Slide Time: 14:14)



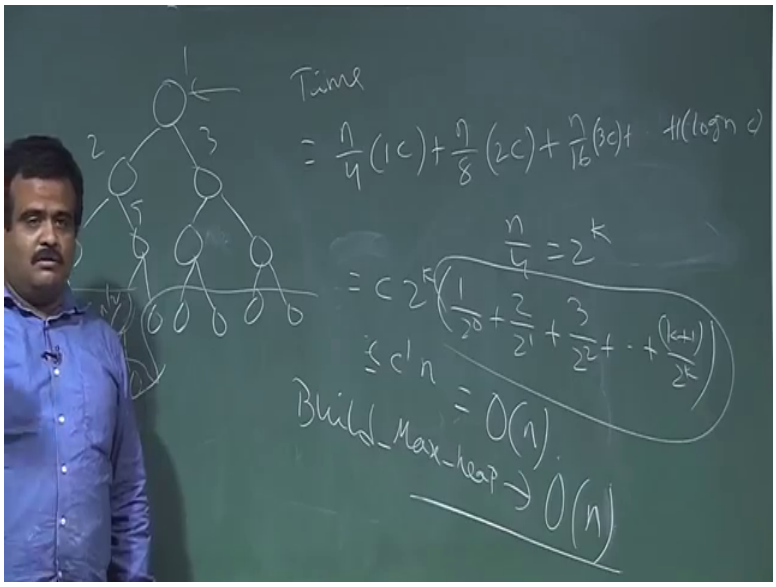
So, this is the say tree and this is the node n by 2 i mean. So, let us draw once more now these are all leaf nodes and this is the n by 2 th node, now we are calling the max heapify starting from n by 2 now for these node the max heapify call can is only once.

Because this node will compare with the maximum of this 2 then we may have to exchange then we stop there is no further call. So, the node which are in this level 1 level above from the low level 1 level above the leaf level these are all leaves 1 level above the leaf level for them. The number of call is one, this will be one call only because it will exchange with this and then stop because we are reaching to the leaf, but the node which are here for them we call the max heapify it may exchange this and then again we may have to call. So, it may go this for them 2 max heapify call for them.

But 2 if they are 2 level above from the leaf so, that is why it is not log n for all it; it will be log n for the root if we call the max heapify here this max heapify call for this node may come down and consist of log n when we call for with this max heapify. But not for the all I mean not for all the nodes we will have log n many max heapify call in the worst case; that means, if a node is lies in l level from the bottom then it may cause us the order of l.

So, that is the analysis we have to do. So, the number of nodes which are just one level above for them number of call is just one max heapify and what is the number just one level above. So, that number is n by 4.

(Refer Slide Time: 16:39)



So, n by 4 into 1 c plus and then 2 level above n by 8 into 2 c for which are the nodes which is 2 level above from the down from the for them we may have to call twice. So, this may be exchange with this again we may have to call this max heapify like this. So, 2 c plus n by 16 into 3 c this way, but for the root we need to call 1 into log n.

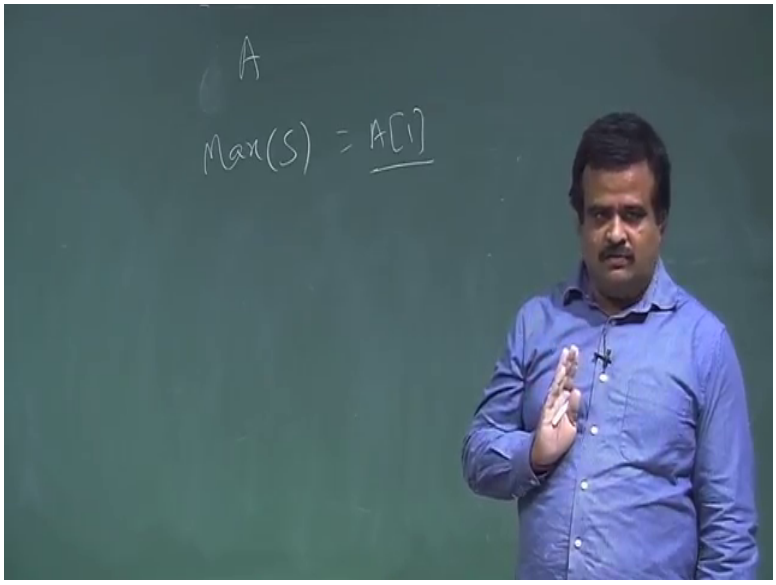
So, that is. So, log n log n c this is for the root. So, this is the time. So, not for every node we are spending log n time. So, this is important. So, this if we have to simplify. So, this is the time complexity for build max heap. So, if you simplify this. So, this will be basically c into. So, if you take n by 4 is equal to 2 to the power k just we take n by 4 equal to then this is c into 2 to the power k in common. So, 1 by 2 to the power 0 plus 2 by 2 2 to the power 1 plus 3 by 2 square like this plus k plus 1 by 2 to the power k this way.

Now, this term is bounded this can be proved this is p g p mix series I mean finite series. So, this is bounded by a constant. So, this is basically less than equal to c some another constant c prime into n. So, this is basically order of n algorithm. So, build max heap will take build max heap will take sorry n linear time and that is why it can it will give us a

sorting algorithm build max heap will take linear time it is not n log n because for not for all node we are spending log n call.

Because the nodes which are just 1 level above for them just a one call and that is 1 c and number of such node is n by 4 the node which are just 2 level above and that is 2 c a number of nodes is n by 8 like this. So, this is the time complexity for build max heap. So, now, we will talk about how we can use this 2 have a heap sort how we can use this max heap array or max heapify this. So, this is the way this is basically nothing to do with sorting still now we have not talk about sorting yet.

(Refer Slide Time: 19:44)



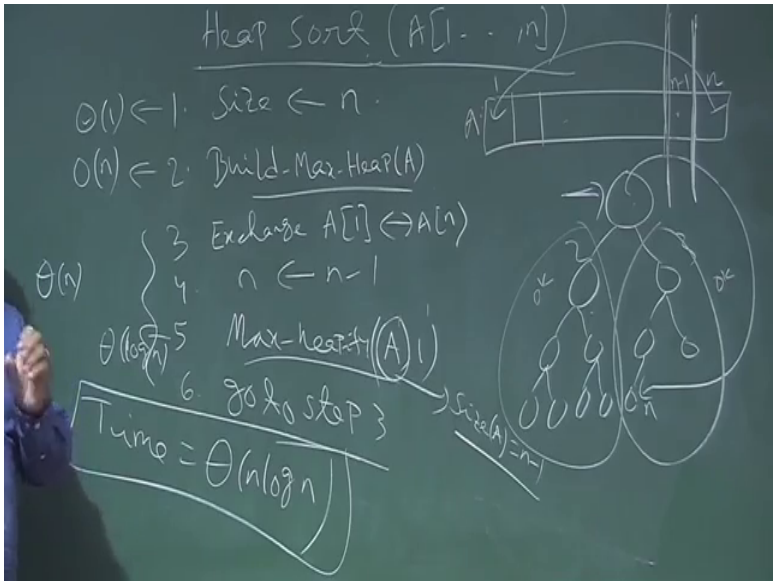
So, this is basically priority queue and this is basically to maintain a set S where we should perform this type of operation like we should able to return the maximum. So, max of s. So, if we have a array and if we make it max heap array then max of S is basically the first element after making the max heap after calling the build max heap and then the extract max extract max means we must delete it. So, we just extract exchange and then again we call the max heapify.

So, again it will make the max heap array like this. So, that is the operation we can do. So, another operation is increase key. So, we can go to that particular position we can change the value and if we change the value. So, again to make it max heap array we need to call the max heapify at that position. So, that may cost us again logarithm time depending on which position we are calling. So, worst case it may come from root to the

height of root to the leaf. So, that is the decrease key operation. So, these are the operation we must able to perform.

So, this heap data structure is used as a implementation of the priority queue now we will talk about how we can make use of this max heap data structure to have a sorting algorithm. So, that is called heap sort.

(Refer Slide Time: 21:19)



So, now, we will talk about the heap sort which is basically using this max heap array. So, the heap sort this is the sorting algorithm. So, it is taking an input of size n. So, what we are doing we are taking size of this array to be we are defining size n.

So, we have a array of size n 1 to n this way we are doing. So, this is our input. So, size is n we are just taking the size now we are calling the build max heap in a first sort build max heap on A. So, this is the making the array as a max heap array. So, we have any unordered array and it is making the array to be a max heap array. So, once we have a max heap array we know the maximum is rooted here. So, this is the array. So, something like this.

So, this is 1 2 3 like this. So, this is n. So, this is after calling the build max heap and this is max heap array. So, this is the maximum. So, now, this is our situation. So, this array will change to the max heap array. So, this is the situation now what we do we know this is the maximum. So, we just exchange we just send this guy to the end we know this is

after calling this build max heap this array. So, we know this is the maximum. So, we send to the end and we reduce the size of the array.

So, we exchange this we just exchange; exchange A 1 and A n. So, whatever value over here we send it here this is the max heap array after calling this build max heap whatever value over here we send it here and we bring this value over here now once we bring this value over here. So, this may violate the max heap property. So, again we mean again to make it max heap array sub array because we just disrupt this we forget this element because this is the element which if we sort it this is the position where the maximum will be sitting.

So, we just forget that. So, we just do what exchange this and then we reduce the size by 1 size minus 1. So, n minus 1 basically or we can just say n is equal to n minus 1 yeah size minus 1 or n n we replace by n minus 1 so; that means, we just forget this part now our array is up to n minus 1 now again we call max heapify max heapify a at the point.

This, but this a array is now size is size of a is now n minus 1 not n that we have to remember we call the max heapify at this point because this as exchange with this element. So, it may violate the max heapify property here and then we can remember this is already this is because this was the max heap array. So, we may have violation here. So, this will keep on fixing and again after this call it will be again a max heap array. So, then again we will do this same thing. So, we can just put n is equal to n minus 1 here instead of size.

Again we go to the; this 1 go to step 3 and this will continue until n is equal to 1 once n is equal to 1 we stop. So, basically then again we put this n minus 1 here. So, this is the second maximum and then we disrupt this element from here. So, this is our sub array then this way we continue until we reach to the n is equal to 1. So, this is the way we keep on reducing the size and we fix the largest element this way and every time when we exchange this with the last element then that may violate the max heapify property.

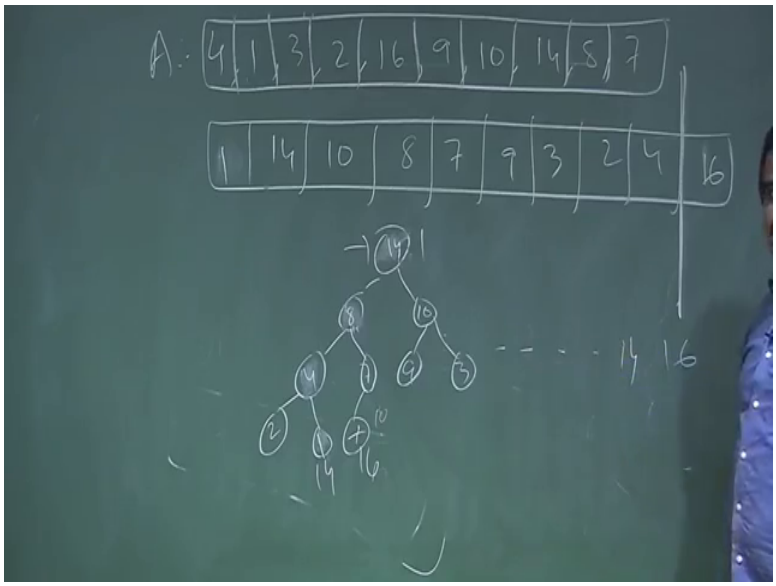
Then we have to fix by calling the max heapify we will take an example, but we further let us talk about the time complexity. So, this is theta 1 and then build max heap will take linear time this is once and then we are exchanging this and this is a loop of size n and each time we are calling the max heapify property. So, this is the log n time will take

because it may go. So, the time complexity is basically n log n time is basically n log n. So, this is a n log n time algorithm and this is a in place sorting algorithm.

Because we are not taking help of any extra memory everything we are doing in the array. So, this tree just we are viewing the tree we are visualizing the tree in reality this tree does not exist we are not using any pointer to build this tree. So, we are just viewing this tree. So, everything we are doing in this array given array everything. So, that is that is why it is called in it is a in place sorting algorithm. So, we are not taking any extra storage to have this. So, let us take a quick example quick execution of this.

So, this is a n log n time algorithm and this is called heap sort some quick example on this heap sort. So, suppose we have say this is the input say yeah. So, max heapify. So, maybe we can take.

(Refer Slide Time: 27:55)



So, say suppose this is our given input this is we have 14 8 7. So, this example we have taking to get the build max heap. So, this is our given array we need to sort it. So, what we do we first make it a max heap array?

So, we called the build max heap if we call the build max heap it will be like this 16 14 10 this part we have seen 8 7 9 3 2 4 1. So, so this is the build max heap. So, if you draw the tree. So, 14 10 8 7 9 3 2 4 and 1 this is the tenth node. Now what we do after build max heap we exchange this and this. So, we exchange 14 and 16.

So, this 16 will come here and 1 will be here. So, this 16 will come here and 1 will be here and we forget this part now the array is reduce by 1 size 1. So, again we call build max heap here. So, it will exchange this maximum of this 14 will. So, 14 will come here and 1 will be here again we have to call build max heap. So, 1 will come here 8 again we have to call build max heap. So, one will come here and 4 will go here we stop. So, again this is a max heap array max heap sub array.

So, again we exchange this here with the low this is this is again will exchange with this. So, 14 like this. So, 16 14 like this, it will be we continue this until it will end to the all the exhaust the all the array element. So, this is the implementation, this is the execution of the heap sort.

Thank you.