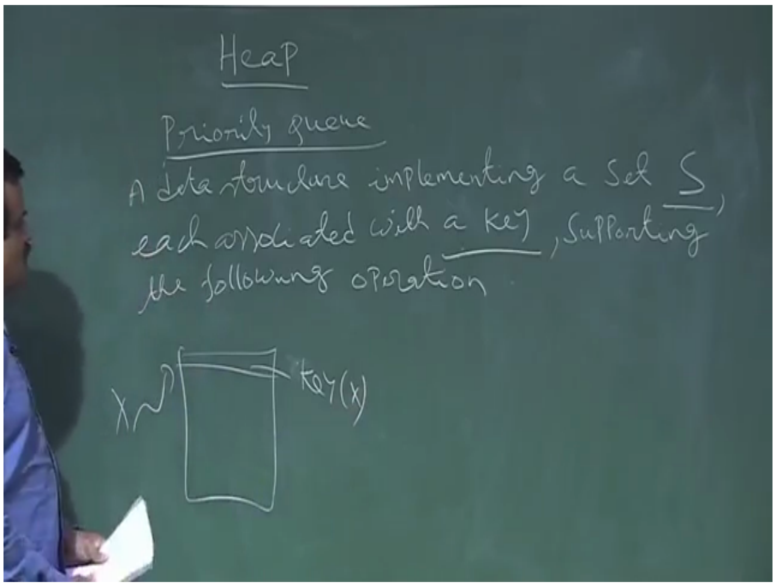
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 13
Heap

So, we talk about very cool data structure which is called heap and this heap is I mean will basically talk about heap sort, but heap can be totally independent from heap sort. So, basically heap is nothing do with heap sort, but we use this heap I mean basically max heap to have a sorting algorithm and that is called heap sort.

(Refer Slide Time: 00:57)



Heap

Priority queue

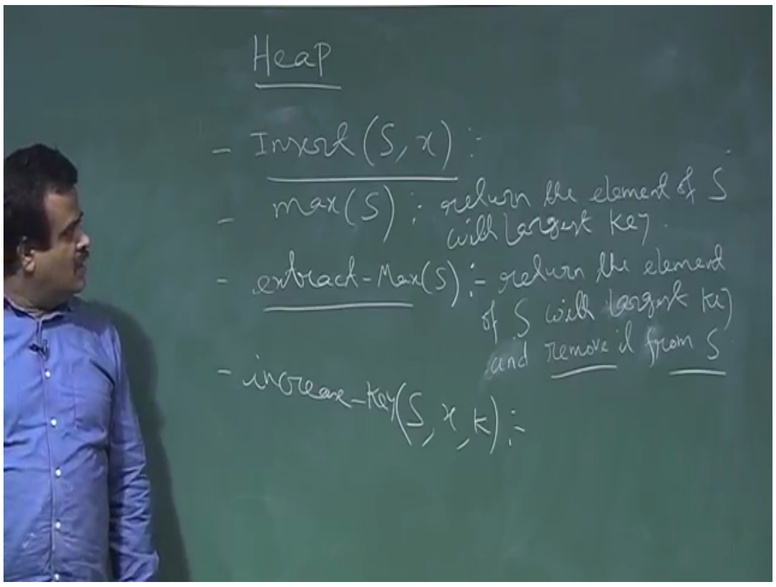
A data structure implementing a set S, each associated with a key, supporting the following operation

But heap can be taught totally separate way like for specially it is a priority queue implementation. So, basically what is a priority queue?

So, basically a data structure implementation; implementing a set S which is associated with a key each element in S is associated with a key and it should support the following operation, supporting the following operation. So, basically we have a dynamic set S and this S set and each element in the S set is associated with the key. So, key is basically we can say if S is record stage student record, student roll number.

So, roll number could be the. So, if this is a record. So, this is the roll number of the student. So, this could be the unique key of x. So, each of this data or we can say record is associated with a key. So, key is the unique representation of that set of that. So, each element of S is associated with a key and it should able to perform the following operation like. So, this is S set the operations are basically what operations are basically insert we should able to insert an element in the S. So, this are the operation insert.

(Refer Slide Time: 02:51)



Heap

- Insert(S, x) :
- max(S) : return the element of S with largest key.
- extract-Max(S) : return the element of S with largest key and remove it from S
- increase-key(S, x, k) :

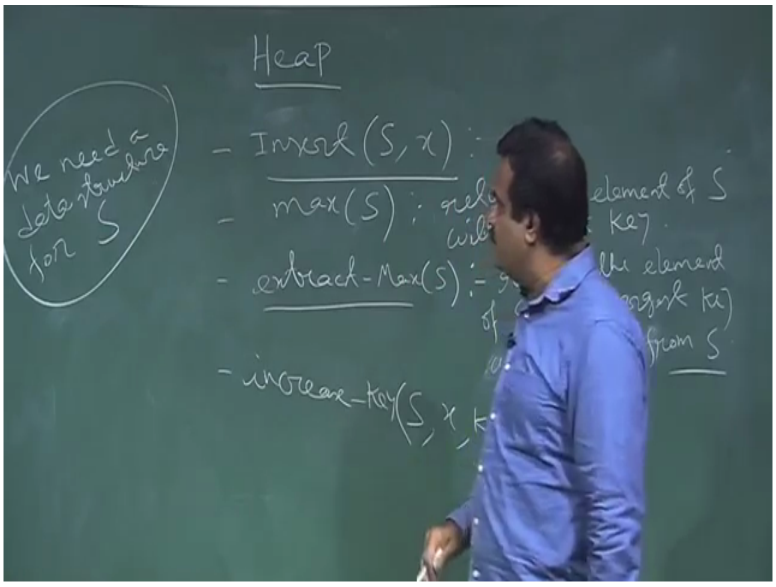
So, this is basically we insert x into f. So, we insert x a new record if the x is the key of that record into S new record with some key value in x this is the insert operation and there is a max operation max of S. So, this is basically the return the element whose key value is maximum return the element of S with largest key value. So, we are looking for maximum element from this set. So, S is a dynamic set every time.

So, we should able to insert a new record new key in this S and we should able to find out this is another operation max operation we should able to get the maximum of this set maximum; maximum element from a this is the largest key value and we should able to do this another operation is called extract; extract max it is telling us we should return a maximum and delete from the q delete from S we should return.

We should return the element return the element of S with maximum key with largest key value and remove it from S that is also important. So, extract mean is extract max is we are extracting the maximum element that is the max S and we should able to remove that element from S remove means delete and remove it from S. So, this is also important.

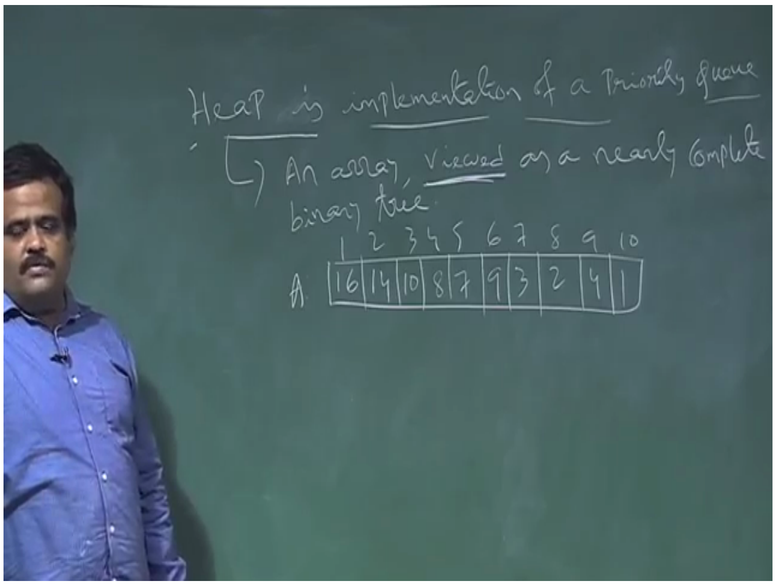
So, extract max means we should able to get the maximum; maximum element maximum we should able to get the element whose the key value is maximum and we should able to remove that element from this set S and another operation is increase key. So, it should able to. So, this is telling us we should able to increase the. So, we take a key we take a x and we should able to replace its key value by the new key value k. So, that is the increase operation. So, it should go to that particular value and it should able to increase it this operation.

(Refer Slide Time: 06:10)



Now, if this data if suppose we have a simple. So, we need a data structure for this. So, we need a data structure for S. So, that is our goal. So, we need a data this is called priority queue implementation. So, we need a data structure for this S so, that we can perform this type of query so, for that heap is the data structures which are going to introduce for this purpose. So, heap is basically it is basically an array, but it is viewed as a completely binary tree nearly complete binary tree.

(Refer Slide Time: 06:57)



Heap is implementation of a Priority Queue

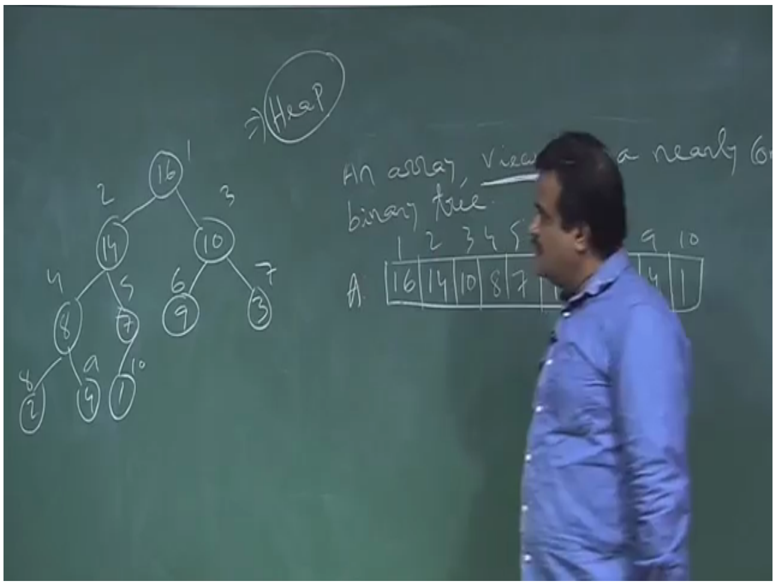
↳ An array, viewed as a nearly Complete binary tree.

	1	2	3	4	5	6	7	8	9	10
A	16	14	10	8	7	9	3	2	4	1

So, we just. So, heap is basically. So, heap is basically implementation of a priority queue of a priority queue. So, that that S, so, heap is use to, it is basically an array which is viewed as a viewed or visualize as A, this is the viewed or visualize as a nearly complete binary tree how we will explain complete binary tree. So, basically it is an array, but we are viewing this array as a binary tree how we can view a array as a binary tree.

So, that we have to explain. So, suppose we have given a array. So, say we have array. So, this is 16, these are the element these are the key value 10 8 7 9 3 2 4 1. So, how many element one 2 3 4 5 6 7 8 9 10, so, we have a array of 10 element now this array we want to view as a we want to visualize this array as a tree.

(Refer Slide Time: 08:48)



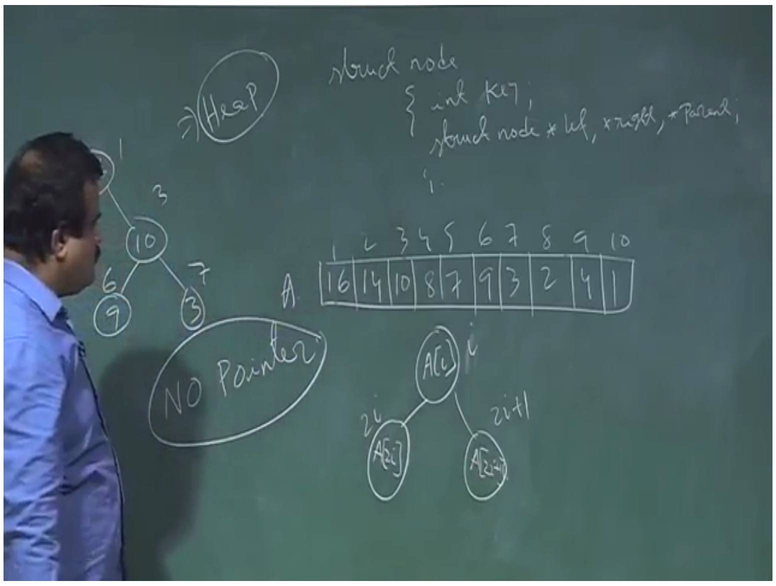
So, how we can do that so, one way to do this is we start with this as a root. So, we start with this as a root.

So, this is A 1 and then A 2, A 3, then A 4, A 5, A 6, A 7 then A 8 A 9 and then we have a 10 because array is up to 10. So, this is the way we view this array as a tree. So, this is basically A 1; A 1 is 16. So, we just put this value this is the key values 10 8 7 9 3 2 4 one. So, this is the visualization of this array as a tree and this is nearly complete binary tree.

Because here number is 10 if the number is say 15 if we have another one 2 3 another 4 element in the array if the array size is 1 to 15 then it is a complete binary tree, but anyway this height of this tree is log n if there are n element. So, that is why it is called nearly complete binary tree. So, depending on the value of n we will have complete or it is a nearly complete. So, this is a visualization of this array as a tree and this is called heap this visualization this is called heap.

But this is just a visualization in reality we do not have this tree. So, in reality if we have to have the; if we need to do the tree implementation what we need to do suppose in c we want to implement implemented tree.

(Refer Slide Time: 10:41)

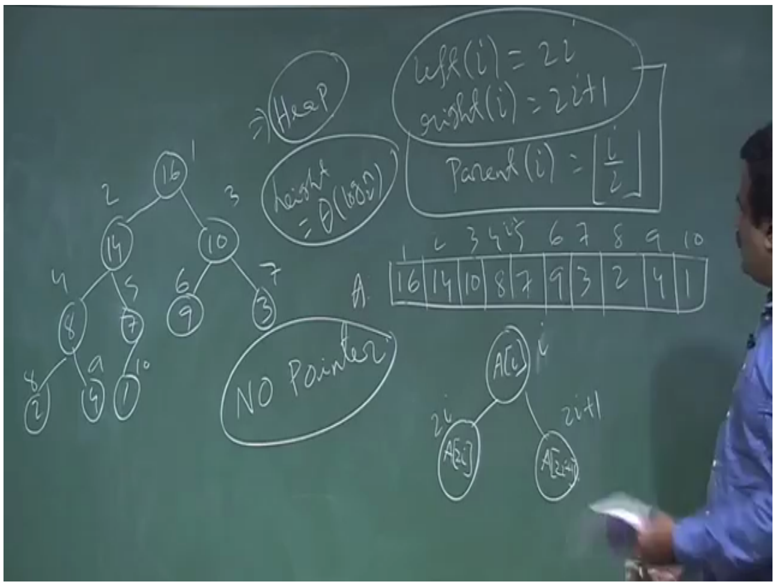


So, what we need to do we need to define a structure struct note then we need to have the data or the key value and the we need to have the pointers struct note star left star right may be star parent something like that.

So, this way we just define just implement a tree in a in a c language to have a this type, but here we do not need that. So, no pointer is required this is the advantage no pointer is required to have this tree because we know if a note is say I. So, if this 3, 3 note, who are the child for 3 6 and 7; that means, if a note is I, this is a I, this is I, if a note is a I then we know the child's are at A 2 I A 2 I plus 1 this is basically 2 I 2 I plus 1.

So, this is the formula we use to build a to visualize that tree this is this tree is our visualization it is not there actually we are not having this tree we are not having the pointed to have this tree we are just viewing this tree basically we have a array, but we are viewing this tree that this 10 is the child of 10 is the parent of 9 and 3 like this. So, basically we have a array, but we are viewing this as a array as a tree and this is basically called heap data structure. So, this is the A 2 I and A 2 I plus 1. So, this is the formula we use for visualizing the tree.

(Refer Slide Time: 12:47)

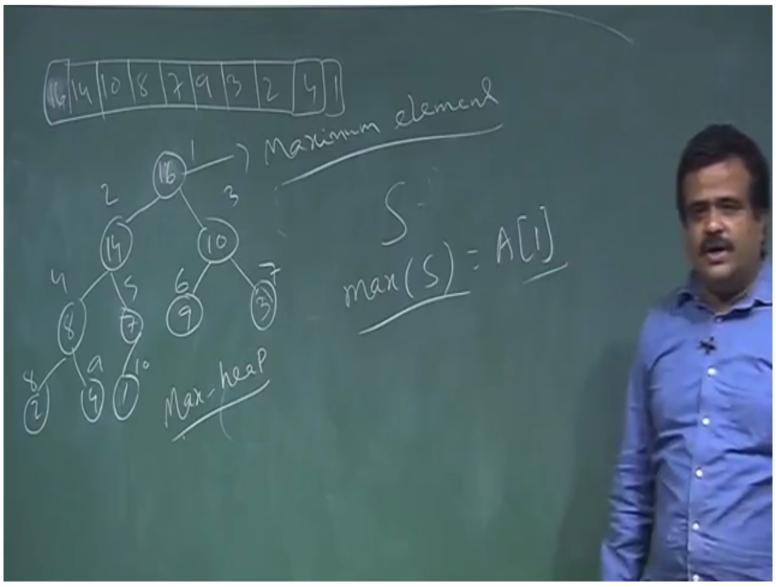


So, no pointer is required. So, we know where at the child. So, basically left child of left of I is 2 I and right of I is 2 I plus 1. So, this is the formula will give us the tree and the parent of I is basically I by 2. So, if by using this formula we will just view this as a tree. So, this is called heap this array is called heap where we are viewing this array as a binary tree nearly complete binary. So, if there are n element height of the tree is.

So, height is basically order of log n. So, that is why it is a good tree it is a balance tree we will talk about balance tree. So, balance means the height is order of log n. So, that is why it is a good tree. So, we can view this as a tree. So, this is called heap this data structure is called heap now when we call this heap as a max heap. So, we call this heap. So, this is the way we just have child. So, so if we take this is as a I then we know that the children are basically 2 I plus 1, 2 I there is no 2 I plus 1.

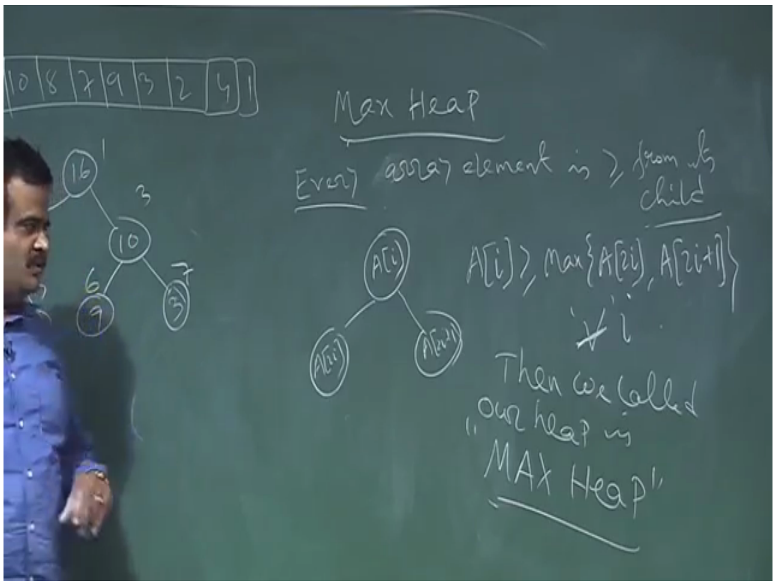
So, this 7 has only one child which is one. So, this is basically the formula to get this to view this tree. So, now, we call this heap to be max heap. So, when we say. So, this is 3 let us have this. So, this is basically the visualization actually we have this array.

(Refer Slide Time: 14:47)



So, let us have the array. So, the array is 16 14 10 then 8 7 9 3 2 4 1 16 14 8 7 9 3 2 4 one this is the array.

(Refer Slide Time: 15:17)


Max Heap
Every array element is ≥ from its child
A[i] ≥ max{A[2i], A[2i+1]} ∀i
Then we called our heap is "MAX Heap"

So, when a heap is called max heap the definition of a max heap is if each of the elements is greater than from his child we take an element this is the element say ith element if it is greater than from his child. So, for 8 the children are basically 2 I 2 I plus 1. So, this 2, if every element if every element every array element is greater than or equal to from his child every element and if this is true for every element so; that means, if this is a i.

So, if this is A i it h element then the child is basically where A 2 i and A 2 i plus 1 so; that means, if A i is greater than equal to maximum of A 2 i and A2 i plus 1 for all I and if this is true for all I then every element is greater than from his child then we call and if this is true for all I this is this is the symbol for all if this is true for all I then we call called our heap is max heap this the definition.

So, for max heap this property should be there every element should be greater than from his child. So, is this a max heap array? So, you can verify that. So, you have to check every element whether it is greater than from his child. So, if we take this; this is having no child. So, this element is greater than from his child this element is this element is greater than from his child this is also greater than from his child this is also greater than from his child. So, this is a max heap array.

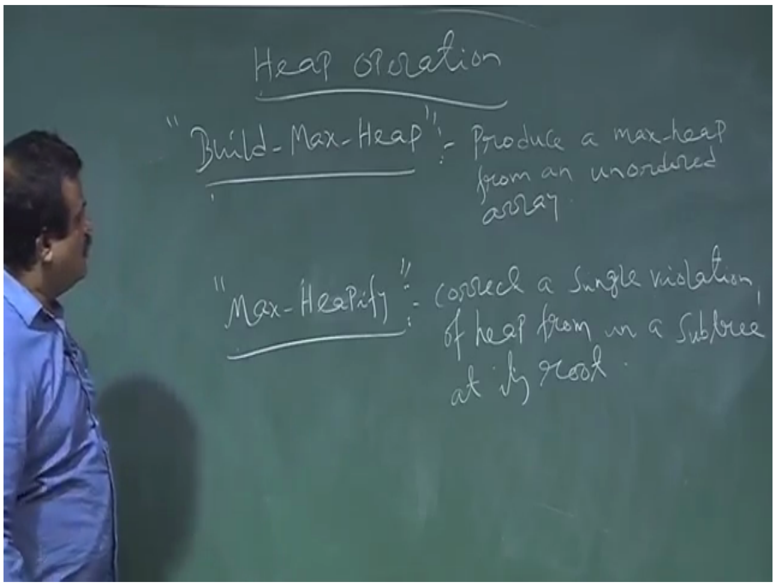
So, this array is a max heap array. So, so this is a max heap array now question is given a array. So, in general we may not have a max heap array. So, given a array suppose we take a input say suppose instead of this say this is 14. So, instead of 14 suppose we have 9 already we have eleven we can put eleven somewhere here. So, instead of 16 suppose this is eleven if this is eleven then this is not a max heap array because this element is not greater than from this.

So, now the question is given a array how we can make a array to be a max heap array. So, what is the advantage of having max heap array? So, advantage is if we have a array which is max heap array suppose this one then we know root content the maximum element. So, that max operation we can easily find out. So, we should able to perform those square is max square is. So, max of S. So, this set is basically S. So, max of S is basically maximum element.

So, this is basically root this is the basically A 1 if the array is a max heap array similarly we can have min heap array in that case the minimum is our priority, main priority. So, we will talk about that. So, if we have main priority then we have the min, min heap in the min heap the property will be the all the elements will be less than from his child. So, that will be the min heap property. So, now, the question is. So, in general we have a unordered array we have given any array how we can make the array to be max heap array or after how we can extract the max.

I mean how we can delete it from this array and again make the array to max heap array. So, those operations we have to do. So, we have to perform few heap operations. So, those are basically max heapify and build max heap. So, these are basically heap operation.

(Refer Slide Time: 20:22)



Heap operation

"Build-Max-Heap": - Produce a max-heap from an unordered array.

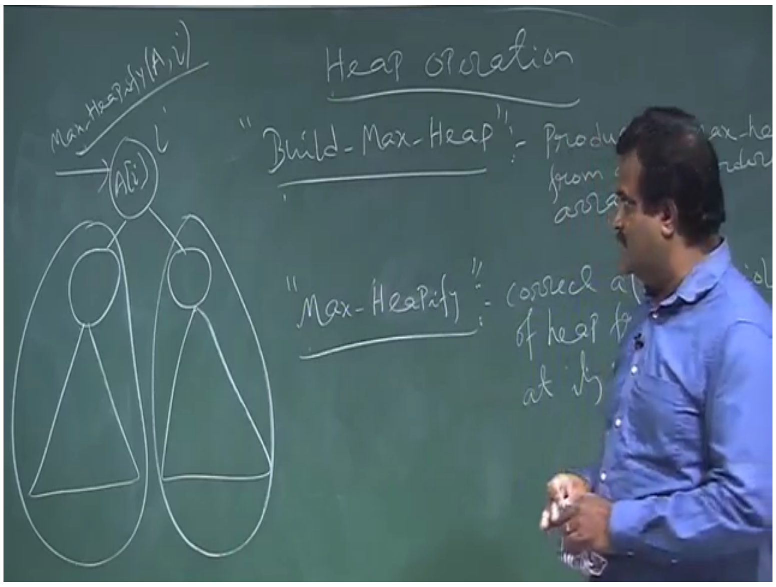
"Max-Heapify": - correct a single violation, of heap from in a subtree at its root.

So, basically build max heap. So, this is basically the function which will produce the produce a max heap array from a unordered array.

We have given a array and we have to make it max heap array and that will be done by this sub routine build max heap by this function. So, build max heap is the function or the sub routine which will take the input as any array and it will give it will it will produce the max heap array will talk about that this function this sub routine and in this build for build max heap we need to take help of another function another sub routine which is called max heapify. So, write small or big anyway it should be consistent max heapify.

So, max heapify means it should able to correct only one violation correct a single violation I will explain violation of heap property specially max heap we are talking about it could be min heap also heap property in a sub tree at its root. So, this is a; this is a sub routine this is a sub routine which we are going to call in this build max heap and this max heapify is basically. So, this is say i th element.

(Refer Slide Time: 22:32)

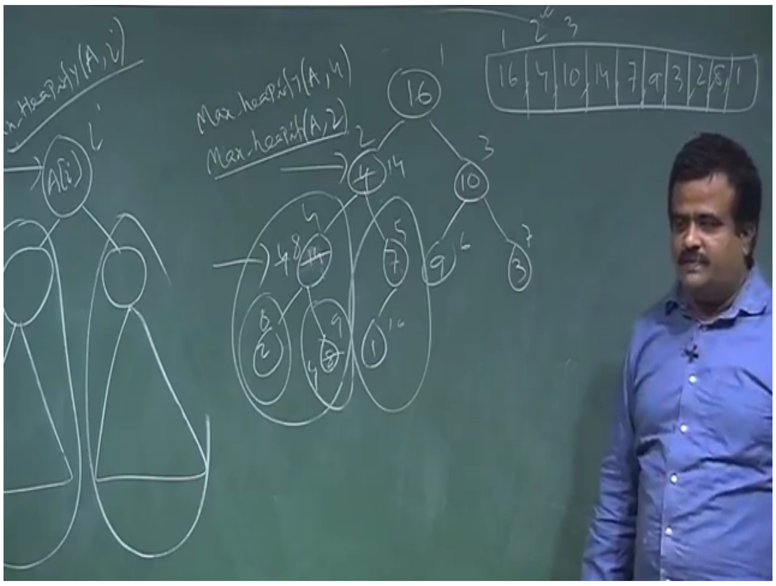


So, maybe we have to call this max heapify of the array at the point i. So, this is the call. So, we will talk about the code for this. So, we will see how this will looks like. So, this is the max heapify. So, we call this max heapify at the point. So, for that what we need to have. So, this is the left sub left element left child this is the right child. So, so this is the left sub array this is the right sub array. So, for this max heapify we assume everything is over here and everything is over here. So, this is this sub array is already max heapify and this sub array is also max heapify only we may have a violation at this point.

So, that is why it can only handle the single violation. So, max heapify can only handle the single violation; that means, everything is fine over here everything is fine over here only we can we may have a problem with this point. So, then only we can call this max heapify this assumption is very much required to call this max heapify. So, whenever we call the max heapify at that point this assumption should be there that we have to check that whether everything is in the left sub array everything is in the right sub array then only we will call max heapify say this max heapify can handle single violation.

So, that is very important. So, will talk about max heapify how it will handle this violation through an example?

(Refer Slide Time: 24:24)



Suppose we have this is suppose a given input say. So, we have 4 10. So, 14 7 9 3 and say we have 2 8 and 1. So, what is the array; array is 16 4 10 then 16 4 10 14 7 9 3 2 8 1. So, this is the given array this is the array this is the array and this is the 1 2 3; these are the indexes 4 5 6 7 8 9 10.

So, that there are 10 elements is this a max heap array we can just check whether each element is greater than from his child this is greater than from his child this is greater than from his child this is greater than from his child this is, but this is not greater than from his child. So, we have a problem over here. So, we have to call this max heapify and for calling the max heapify we have to check whether everything is over here everything is over here this is the left sub tree this is the right sub tree.

If we just check this is already max heapify this is already max sub max heap sub array this also max heap sub array. So, then only we can call the max heapify at this point. So, that is that is the assumption we need to have. So, we call the max heapify max heapify on at 2 I is equal to two. So, we call this max heapify at this point this is 1 2 3. So, this is the point we call the max heapify remember this tree we are just visualizing mainly we have this array and we can viewing this array as a tree.

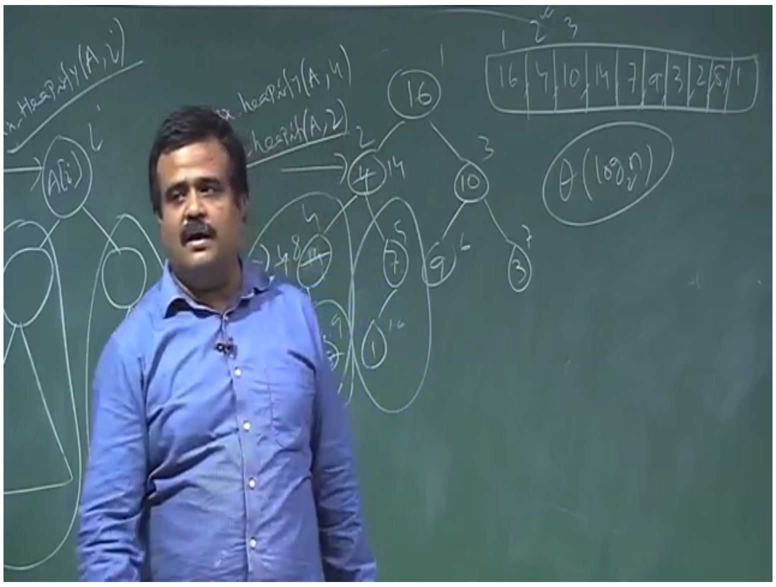
So, now we how we can fix that max heapify sub routine is work like this. So, this has a problem. So, what we do we replace this by the maximum of this two. So, these are the 2 child. So, among this 14 is the maximum. So, we replace 14 with this 4. So, this 14 will.

So, this is 14 of 4 will come here and the 14 will go here now once we put this 4 then it may violate. So, again we have to call the max heapify over here.

So, we call this max heapify max heapify it may violate a and where it is going it is going to 4 we call the max heap here at the point 4 now again we check this is already max heapify this is already max heapify. So, we may have single violation yes there is a violation because this is less than 8. So, what we do again we just replace this by maximum of this 2. So, this is maximum. So, this will come 4 and this will be 8. So, now, this is the leaf note. So, we stop, so this is this is the max heapify operation max heapify sub routine.

So, we keep on doing until we reach to a leaf note or it is fixed so, but every time we have4 to remember this is can handle only single violation. So, every time we need to check whether left part is or right part is we may have a violation in that in that corresponding route. So, this is very much required to have and send then what is the time complexity.

(Refer Slide Time: 28:16)



So, it will; it may go up to the height. So, if it is if there are n nodes and if we starting from the route then the time complexity may be the log n for the max heapify.

So, it may go up to the height. So, will use this max heapify to build a max heap array from unordered array. So, that will be the build max heap will talk about in the next class.

Thank you.