Data Structures and Algorithms Using JAVA

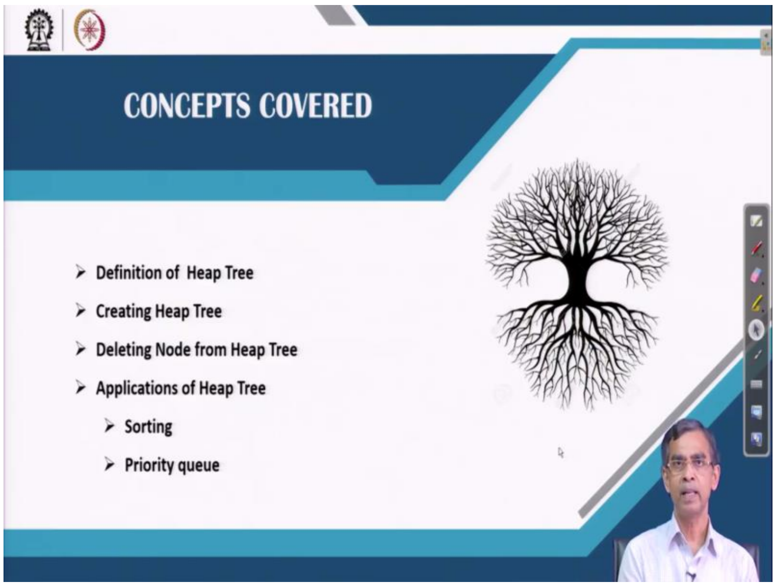
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture: 32

Heap Tree

We are discussing about Binary trees and we have mentioned once that there are different Binary trees having different properties. Binary search tree is one of them, now today we will discuss about another very important Binary tree, its name is Heap Tree. Like Binary search tree, it has many applications, so in this lecture we will try to understand when a Binary tree will be termed as Heap tree.

(Refer Slide Time: 1:24)



CONCEPTS COVERED

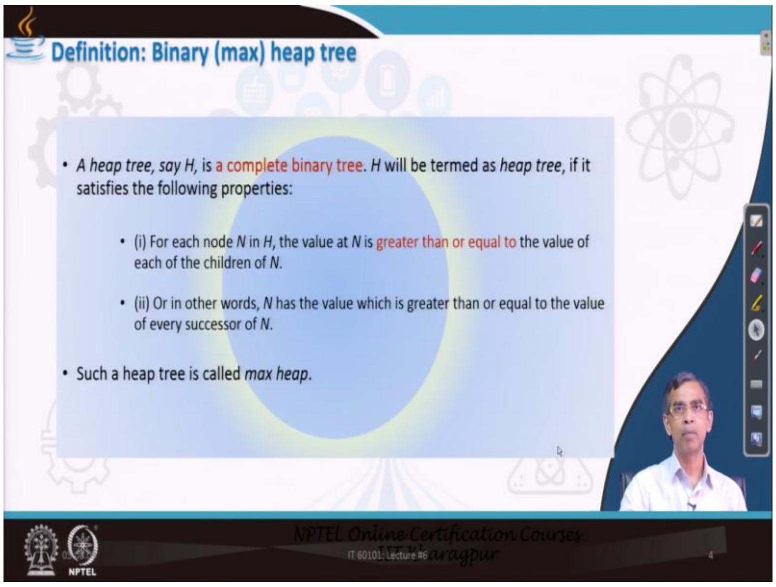
- Definition of Heap Tree
- Creating Heap Tree
- Deleting Node from Heap Tree
- Applications of Heap Tree
  - Sorting
  - Priority queue

So, the property that Binary search tree should satisfy as a Heap tree and then how a heap tree can be created? So, building a heap tree and then will quickly check about how the different operations on Heap tree can be; that means insertion, deletion, like this one. So, mainly deletion is an important operation. We will see how exactly it can be performed and then the lecture will be concluded with a short, brief discussion about its applications. So, it has two very important application, in the sorting of elements or numbers and then priority queue.

(Refer Slide Time: 1:47)

Definition of Heap Tree

NPTEL Online Certification Courses
IIT Kharagpur



Definition: Binary (max) heap tree

- *A heap tree, say H*, is a complete binary tree. *H* will be termed as *heap tree*, if it satisfies the following properties:
  - (i) For each node *N* in *H*, the value at *N* is greater than or equal to the value of each of the children of *N*.
  - (ii) Or in other words, *N* has the value which is greater than or equal to the value of every successor of *N*.
- Such a heap tree is called *max heap*.

NPTEL Online Certification Courses
IIT Kharagpur

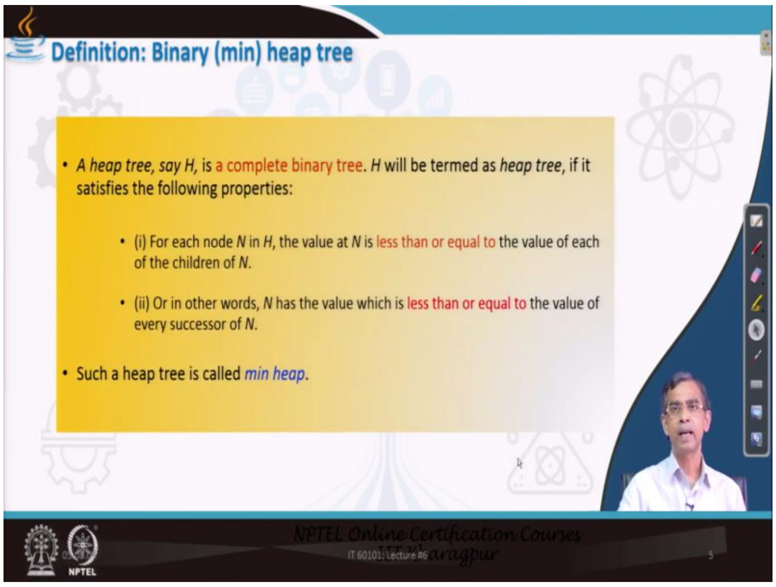
Now let us see, when a Binary Tree will be termed as Heap Tree. Now you can recall, we have defined once that there are two versions of a Binary tree is called the complete Binary tree and full Binary tree. So, a heap tree is basically a complete Binary. Now in addition to this characteristics, property; a heap tree also should satisfy few more properties. Now one property is that, that if each node is, each note is having value greater than, greater than the value of any nodes in either its left subtree or right subtree.

If this property is satisfied for every node, then we can say that it is a heap tree. And here you can check that the root node will contain the value which is basically is the highest value among all the notes in the tree, that means root node is the largest value actually there or maximum value you can say and such a heap is called a max heap.

So, in case of max heap it is a complete Binary Tree and all nodes, the value of each node should be greater than the value of its, any node in its left subtree as well as right subtree. So, this is a concept it is there. .Now likewise this max Heap there is another tree is called the Min heap. It is just opposite to max heap, so in case of max heap; the value is greater than the any value in either of its left subtree or right subtree.

It is basically less than but otherwise it is same complete Binary Tree and each node has the value, has value which is less than the value of any node in its left subtree or right subtree. This means that root node contains the lowest value or minimum value and such a heap tree is called a min heap.

(Refer Slide Time: 4:07)



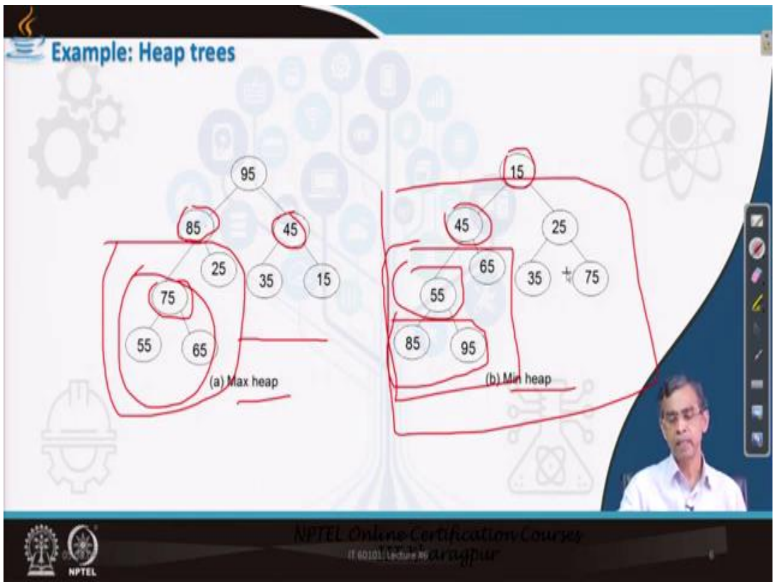
Definition: Binary (min) heap tree

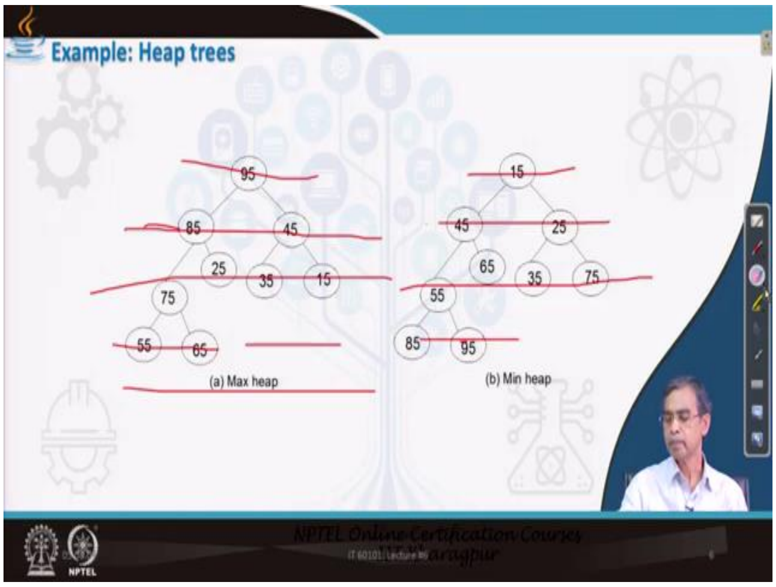
- *A heap tree, say H*, is a complete binary tree. *H* will be termed as *heap tree*, if it satisfies the following properties:
  - (i) For each node *N* in *H*, the value at *N* is less than or equal to the value of each of the children of *N*.
  - (ii) Or in other words, *N* has the value which is less than or equal to the value of every successor of *N*.
- Such a heap tree is called *min heap*.

NPTEL Online Certification Courses
IIT Kharagpur

So, in the heap tree family there are two properties which is possible, max heap and min heap. So, min heap property as it is mentioned here.

(Refer Slide Time: 4:17)


Example: Heap trees


Example: Heap trees

Now here in this example as you can see; this example also is a max heap. Now you see this is the value which has the greater than any value. Now it is two for every note, if we consider this note. Now it consider left subtree and right subtree, it is there. Similarly, if this part if you see it is like this and for this node also it is there. Now for each node basically, so each node has the value greater than any, value in any nodes either in left or right subtree.

So, this is the concept of max heap. And likewise, it is a min heap and you can see the root node is always the lowest or minimum value than the any node in this tree like. Now, this node also has the same thing. It is like this, it is minimum and this node is also at the same thing like it is at this one. Now this way, so each node satisfy this property and another important thing that you see that this is a complete Binary tree.

Complete Binary tree means, it is filled up by maximum number of node in each level except the last level; so this level has maximum, it is also maximum and this level has maximum. Only the last level is not completely filled. It is also like this maximum and this is, only this level is not filled. So both the trees are complete Binary tree and this one satisfy the maximum, Heap property and this satisfy the minimum heap property.

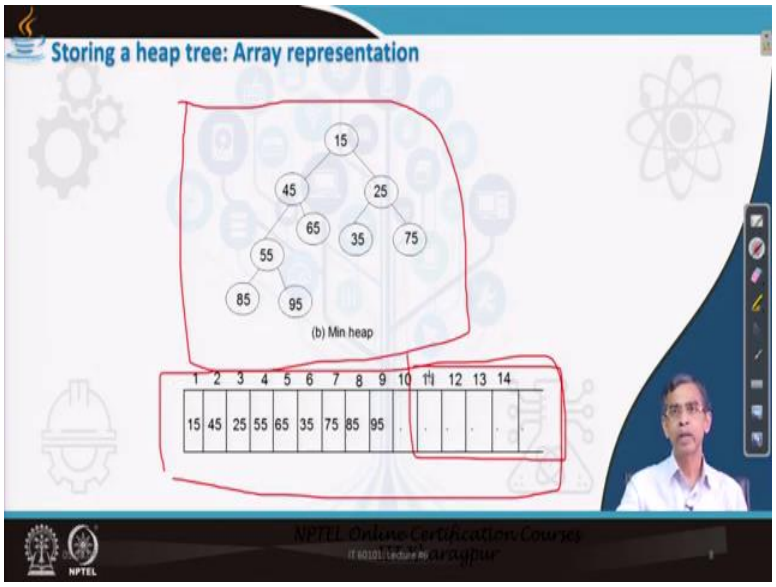
(Refer Slide Time: 5:55)

Storing a Heap Tree

Ok, let us proceed further. Our next task is basically, how a Heap Tree can be stored? Now you can recall the Binary Tree that we have studied can be stored either using an array or using a link representation, that means the pointer.

Now, array representation for a Binary tree or Binary sap trees has many limitations because many space remain vacant if the tree is not balanced or what is called, skewed like. Now heap tree however as it is a complete, so it always store the memory in a most efficient way even using array. So, in order to store a heap tree, an Array is the best data structure that can be considered and usually an array is used to store a heap or heap tree.

(Refer Slide Time: 6:56)


Storing a heap tree: Array representation

Now, here is an example, how array can be used to store a heap. So this basically is a logical view of a heap tree but physically it is stored in an array in this, like this one. Now here you can see in this array, so 1, 2, 3, 4, 7 to 9. Only nine elements are there and nine elements store compact way in the ninth location and we need not use this one.

So, if the nine nodes are there, actually nine elements, nine will be the maximum size of the array is required. So there will be no wastage of memory, it is there. On the other hand, Binary search tree or Binary tree when if you see, there is a lot many memory spaces next remain vacant and wastage of memory therefore.

Now here again we have discussed about how an array can used to represent the Binary tree, if the i'th node is stored in a i'th location, then its left child will be stored in two I location and its right child will be stored in two I plus one location. If the index start from 0, 1 to n or whatever it is there. Now here we can clarify this point, for example, here, so our first node that this is a root node will be start in the first location and then its left subtree will be two I.

That mean two, second location and it right subtree that mean 25 will be in third location. So, it is third location. Now come to 25, 25 is in third location; so it left side child will be sixth location. Now here this one and its right side will be seventy four will be seventh location it is there.

Now this is basically the simple formula to see whether its left subtree, left root, I mean parent of the main node of the left subtree and its right subtree is stored there and (())(9:00) if we follow the same procedure, all nodes will be stored in a compact way in an array. So, this is the mechanism by which a heap can be stored into an array and that is the best or the efficient way that a heap can be considered of storing.

(Refer Slide Time: 9:19)

Creating a Heap Tree

Now let us see, how we can build a heap? What is the formula or mechanism that you can consider? Now we know that a heap will be stored in an array and then we are going to learn about how a heap tree can be created given an array of numbers or a list of elements. Now we will consider in the context of storing the numbers into a heap.

It is usually used for that purpose, but any other type of structure data also be stored in a heap tree. Now for the sake of simplicity in discussion we will consider only storing of number and that is why maximum and minimum come into that way actually. Anyway, so let us see how a heap tree can be built. So, it is called the building a heap tree.

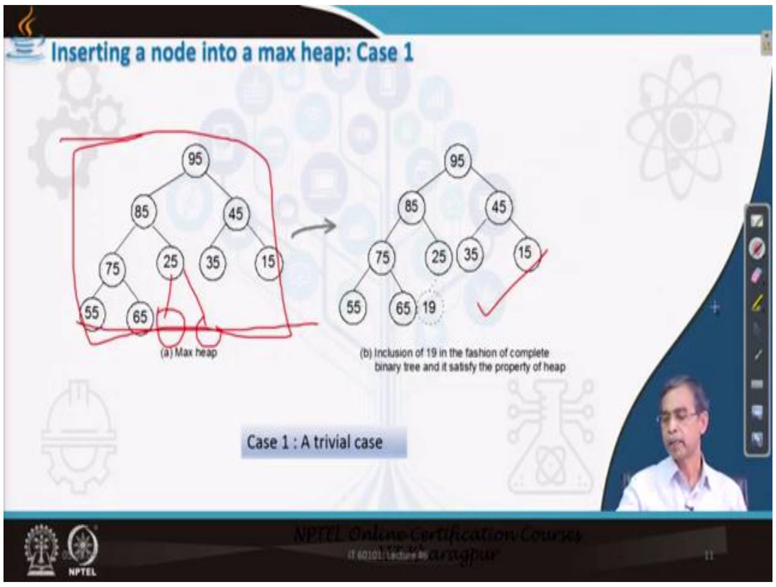
(Refer Slide Time: 10:08)

Insertion Operation

NPTEL Online Certification Courses
IIT Kharagpur

Now what I want to say is that, building heap tree is a very simple procedure. Simple procedure means, we have to insert one node each time starting from the empty tree and it becomes the heap tree actually. And while we insert a node, we have to ensure that insertion of a new note satisfy the heap property of a tree. Either max heap or min heap or whatever it is there.

(Refer Slide Time: 10:39)



Inserting a node into a max heap: Case 1

95
85
45
75
25
35
15
55
65
19
(a) Max heap
(b) Inclusion of 19 in the fashion of complete binary tree and it satisfy the property of heap

Case 1 : A trivial case

NPTEL Online Certification Courses
IIT Kharagpur

Now in the next example we will consider about building a max heap right and at any instant we will consider that. Suppose at any instant, a part of the heap tree is already created. Now, starting from the empty, we first initially inserted some node and then inserted other nodes.

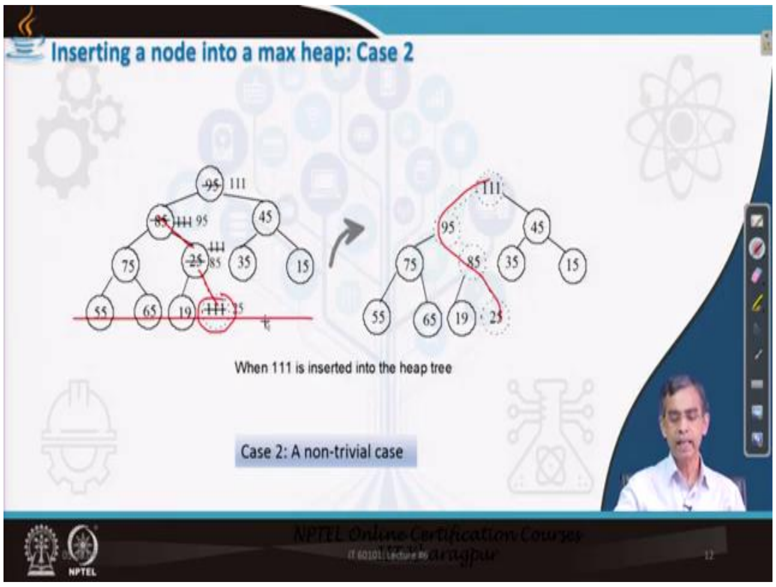
So, this basically whatever the order of elements you have given have shown no issue, but heap tree will be built in that way only, so this is the order of the heap tree.

It is obtained, right? Now, next suppose; we want to insert a new node into this. So, if we insert new node, where we should insert it? That is the question. The question is that heap tree as it is a complete Binary tree, so our next node will be come into as a left subtree of this node. So, our part of new node will come.

After this node is there, next node will come because it is basically tried to fill the level in that way from left to right actually filling. Anyway, so the new node, for example nineteenth will be here, so this nineteen will be inserted as a left child of the node twenty five. So, now let us keep it here but you check. Once we keep it here then whether it satisfy heap property or not?

So, heap property is that this is largest, fine, this is largest of all nodes but we will check that 25 largest than this one because it is there and all are satisfied, so this means that inserting new node does not violate the heap property so it becomes heap tree then. It is a max heap actually. And this is a very simple case, trivial case where we insert and it does not violate any heap property. But always you may not be so lucky to have this kind of case situation.

(Refer Slide Time: 12:31)



Inserting a node into a max heap: Case 2

When 111 is inserted into the heap tree

Case 2: A non-trivial case

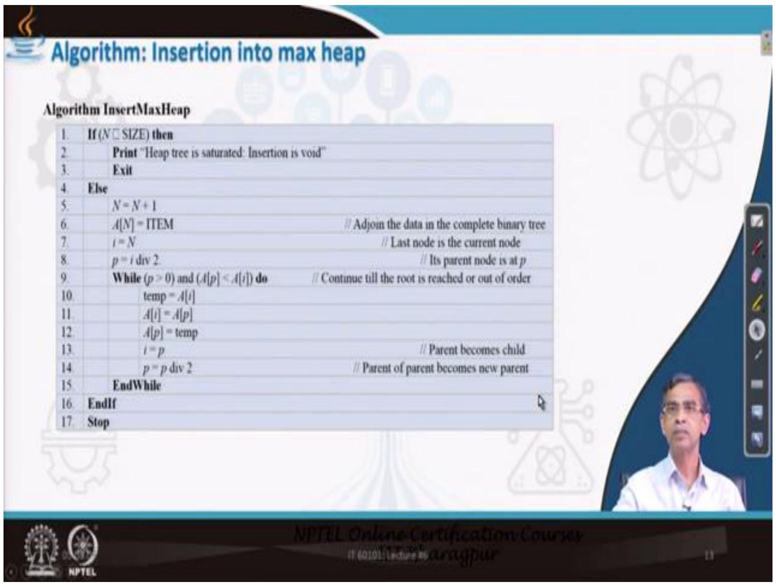
Now here is a one situation you can see, it basically inserting a new node violates the heap property. Now let us consider, this is a initial tree and within this initial tree we want a new node. New node having the value 11, and it will, 111 and it will come this part; because in this portion we have to, so it should come here. Now if we add 11 here and then we can see that this node is basically violate, is not satisfy this property because this node should have highest value of this and this and not there.

So, what is the idea is that; we can just simply replace the two values which basically violates the property. So, 25 should be swapped with 11. That mean, 25 will come here, 111 will go there. But even if we keep 11 here, then we see that this also violates the heap property because it is 85 and it is 100, but here it should be 111 than it should be 85.

So, what they should do is again swap within the 2. After swapping 111 then again it does not satisfy heap property because 95 is not satisfying this one. This means that 95 and 111 to be swapped. Then finally, we reach to root node, so no more, I mean checking are there, so we can stop the process there. Now here if you see, once we insert a node than starting from the node to root node, we have to little bit adjust the value that mean lot of swapping needs to be carried out in order to satisfy the heap property.

And this is basically the idea about, it is called the Re-heap. That means heap tree is there, inserting a node disturbs the heap property so it becomes re-heaping the property and creating a heap starting from empty heap with a given set of elements it is called the heapifying. So, we are here doing max heapifying and while we are heapifying, we just see that some insertion can disturb the heap property therefore, we have to go re-heaping in the process and this is the case that we have discussed about how the re-heaping can be done.

(Refer Slide Time: 14:51)



Algorithm: Insertion into max heap

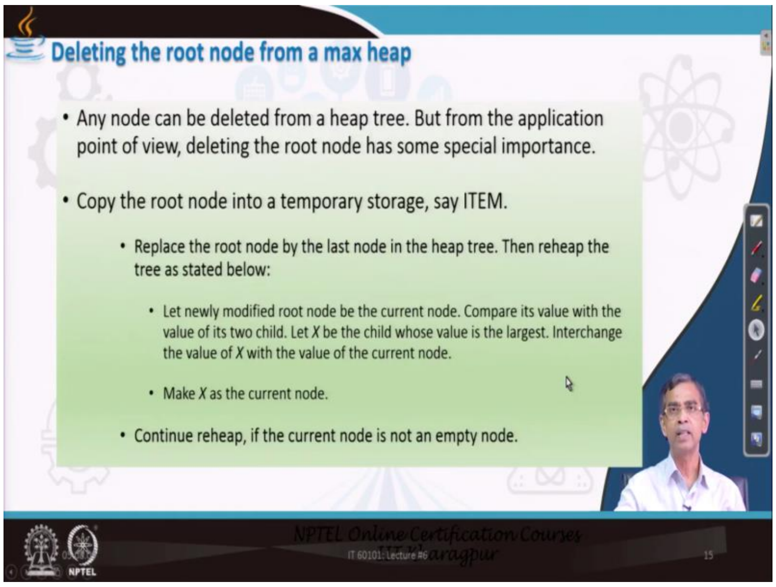
Algorithm InsertMaxHeap

```
1.  If (N □ SIZE) then
2.      Print "Heap tree is saturated: Insertion is void"
3.      Exit
4.  Else
5.      N = N + 1
6.      A[N] = ITEM                       // Adjoin the data in the complete binary tree
7.      i = N                             // Last node is the current node
8.      p = i div 2                       // Its parent node is at p
9.      While (p > 0) and (A[p] < A[i]) do   // Continue till the root is reached or out of order
10.         temp = A[i]
11.         A[i] = A[p]
12.         A[p] = temp
13.         i = p                         // Parent becomes child
14.         p = p div 2                   // Parent of parent becomes new parent
15.     EndWhile
16. EndIf
17. Stop
```

And this is the algorithm that you can consider the re-heap as a procedure for the heap tree.

(Refer Slide Time: 14:56)

Deletion Operation



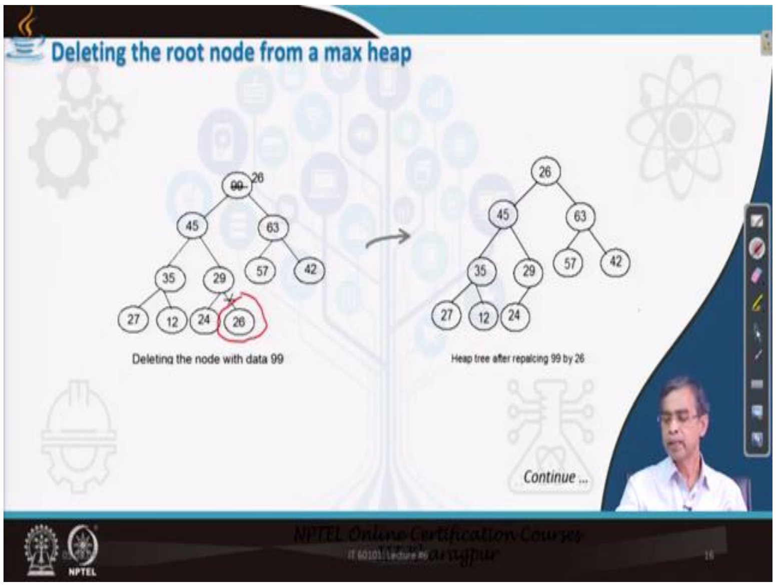
Deleting the root node from a max heap

- Any node can be deleted from a heap tree. But from the application point of view, deleting the root node has some special importance.
- Copy the root node into a temporary storage, say ITEM.
  - Replace the root node by the last node in the heap tree. Then reheap the tree as stated below:
    - Let newly modified root node be the current node. Compare its value with the value of its two child. Let *X* be the child whose value is the largest. Interchange the value of *X* with the value of the current node.
    - Make *X* as the current node.
  - Continue reheap, if the current node is not an empty node.

Now let us consider the delete operation. Delete operation is basically, is a, is not a very common operations in case of heap tree. This because delete operation is not from any location of the heap tree, only delete operation in heap tree matters, deleting a root node actually. For deleting other nodes although it is possible, but usually it is not, it does not have any practical importance for this slide.

So, we will consider about how the delete of a root is there? Now, once we delete a node, it basically again disturb the heap property that mean deleting a node like inserting a node needs re-heaping. Here also deleting a node also needs re-heaping.

(Refer Slide Time: 15:45)



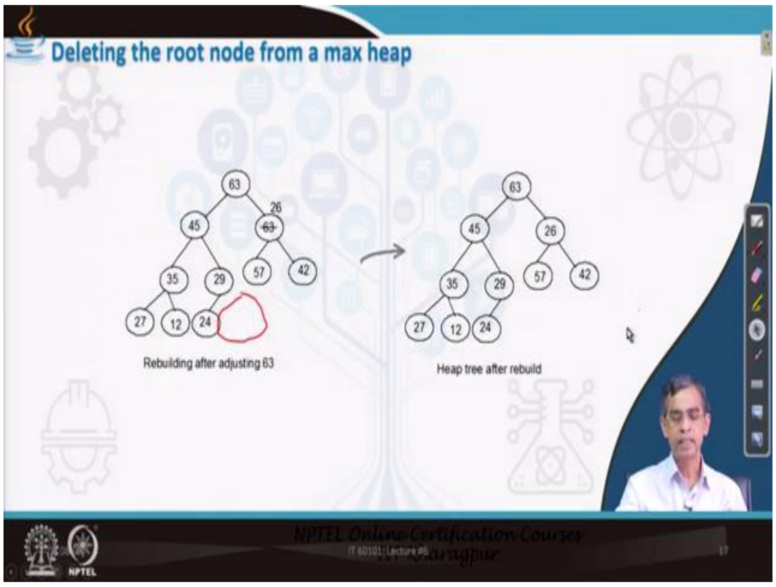
Deleting the root node from a max heap
~~99~~ 26
45
63
35
29
57
42
27
12
24
26
Deleting the node with data 99
26
45
63
35
29
57
42
27
12
24
Heap tree after repalcing 99 by 26
Continue ...
NPTEL Online Certification Courses
IIT Kharagpur

Now, let us see the procedure of delete in, delete the root node. Now suppose this node is a root node in this binary, in this heap tree and we want to remove this node. Now removal is that; now you can node, we should remove in such a way that it should satisfy complete Binary Tree. So, if we remove then better is that the last node that is there in the last level should be removed actually. Now this means that, we can remove it, but we can remove in the sense that we can replace this root node by this node and then 99 will be deleted this way.

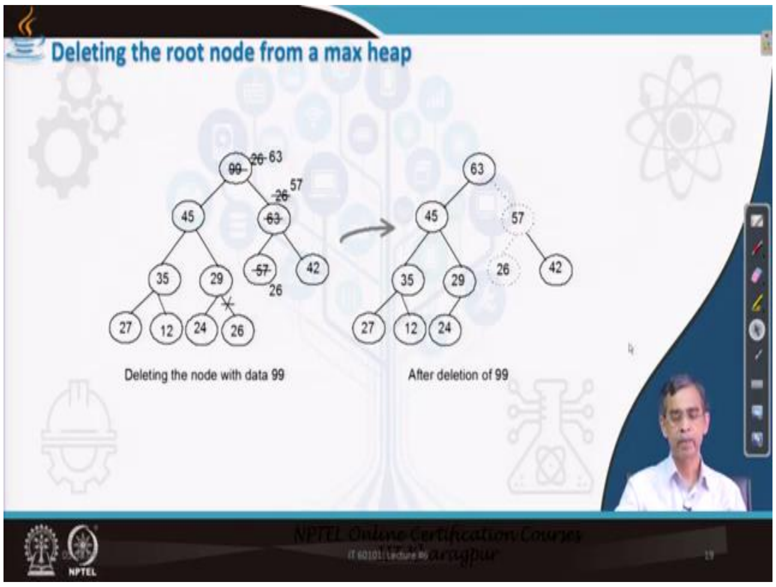
So, replacing the root node by the last node in the last level is basically the first step towards the deleting. So, if we do it, then if we check so the for max heap is concerned this procedure, in fact, disturb the heap property because it is twenty six which has to be the largest of, among the all values in the rest of the tree, but not and therefore we need re-heaping.

Now re-heaping is very simple here, we have to check the swap. Swap but which one? Actually out of the two, I mean child nodes; we have to swap these value with the largest value because this is mix heap. So, 26 should be compared with 45 and 63 and which is 63 is largest, so we will swap 63 and 26. This means that 63 will come here and 26 will go there. Again we check whether it satisfy or not? Now, again 26. but it does not satisfy then 57 and 42; so 57 and 26 will be swapped, finally 26 will come.

(Refer Slide Time: 17:27)


Deleting the root node from a max heap

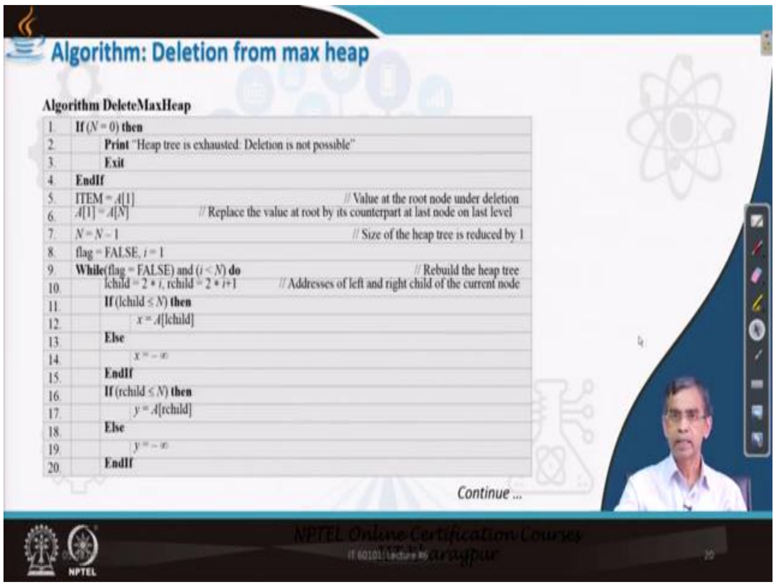
Rebuilding after adjusting 63
Heap tree after rebuild
NPTEL Online Certification Courses
IIT Kharagpur


Deleting the root node from a max heap

Deleting the node with data 99
After deletion of 99
NPTEL Online Certification Courses
IIT Kharagpur

So, after the re-heaping we can see the node that we can get it, it basically shown here. This is basically, so this is the procedure will go on and finally it is here. So, this is the basically we re-heaping the tree and final tree is available here. So, this is the procedure that you can follow and you can just simply delete and this is the case of deletion in fact. So, these are the two fundamental operation that is required for the case of deletion.

(Refer Slide Time: 17:56)



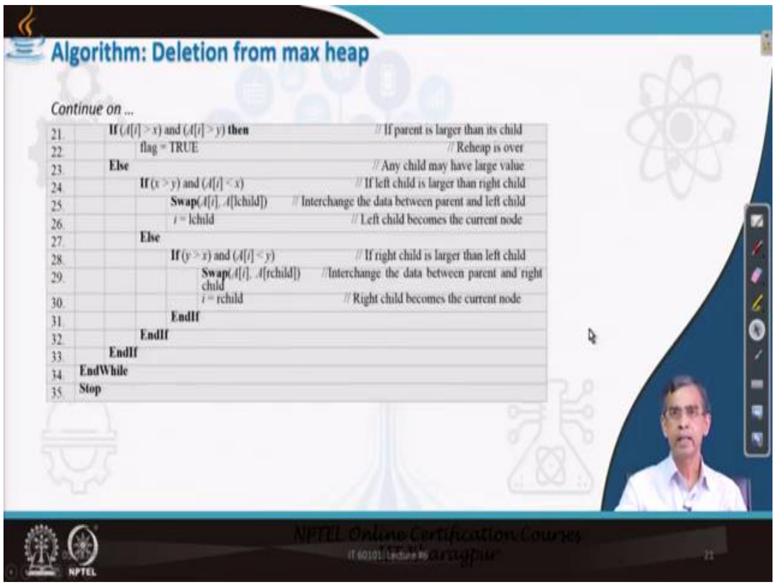
Algorithm: Deletion from max heap

Algorithm DeleteMaxHeap

```
1.  If (N = 0) then
2.          Print "Heap tree is exhausted: Deletion is not possible"
3.          Exit
4.  EndIf
5.  ITEM = A[1]                                  // Value at the root node under deletion
6.  A[1] = A[N]          // Replace the value at root by its counterpart at last node on last level
7.  N = N – 1                                    // Size of the heap tree is reduced by 1
8.  flag = FALSE, i = 1
9.  While(flag = FALSE) and (i < N) do                           // Rebuild the heap tree
10.         lchild = 2 * i, rchild = 2 * i+1   // Addresses of left and right child of the current node
11.         If (lchild ≤ N) then
12.                 x = A[lchild]
13.         Else
14.                 x = – ∞
15.         EndIf
16.         If (rchild ≤ N) then
17.                 y = A[rchild]
18.         Else
19.                 y = – ∞
20.         EndIf
```

Continue ...

NPTEL Online Certification Courses
IIT Kharagpur



Algorithm: Deletion from max heap

Continue on ...

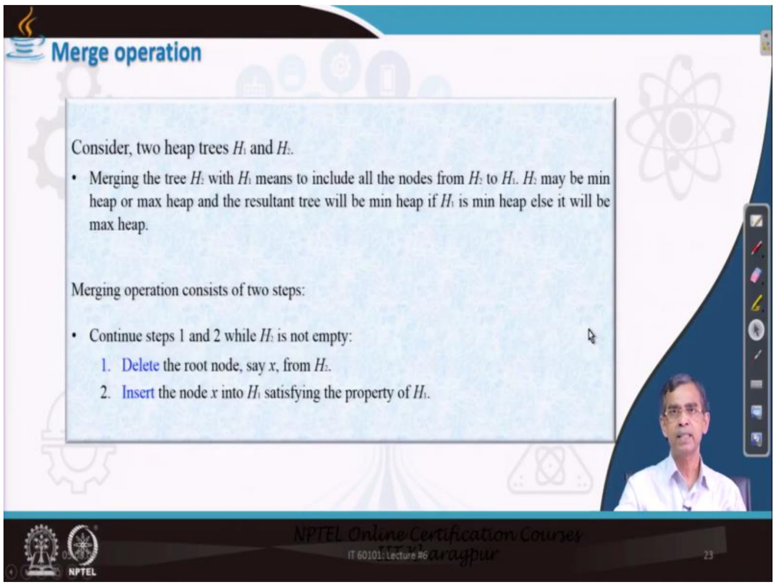
```
21.         If (A[i] > x) and (A[i] > y) then              // If parent is larger than its child
22.                 flag = TRUE                                          // Reheap is over
23.         Else                                                // Any child may have large value
24.                 If (x > y) and (A[i] < x)            // If left child is larger than right child
25.                         Swap(A[i], A[lchild])    // Interchange the data between parent and left child
26.                         i = lchild                          // Left child becomes the current node
27.                 Else
28.                         If (y > x) and (A[i] < y)    // If right child is larger than left child
29.                                 Swap(A[i], A[rchild])    //Interchange the data between parent and right child
30.                                 i = rchild                 // Right child becomes the current node
31.                         EndIf
32.                 EndIf
33.         EndIf
34. EndWhile
35. Stop
```

NPTEL Online Certification Courses
IIT Kharagpur

Now this is algorithm that you can follow while you are writing the programme for heapifying the tree and then re-heaping the tree with deletion operation.

(Refer Slide Time: 18:04)

Merge Operation


Merge operation

Consider, two heap trees H_1 and H_2.

- Merging the tree H_2 with H_1 means to include all the nodes from H_2 to H_1. H_2 may be min heap or max heap and the resultant tree will be min heap if H_1 is min heap else it will be max heap.

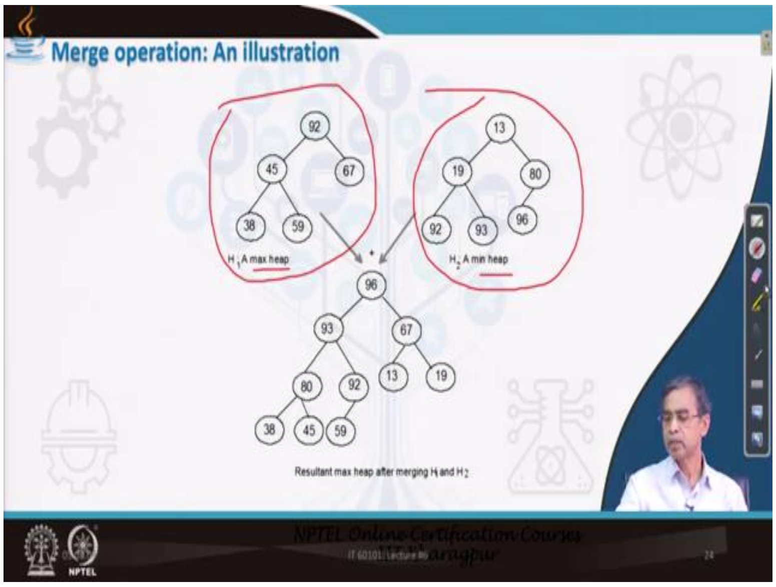
Merging operation consists of two steps:

- Continue steps 1 and 2 while H_2 is not empty:
    1. Delete the root node, say x, from H_2.
    2. Insert the node x into H_1 satisfying the property of H_1.

Now merge operation is the operation on very common, in case of heap tree. Here basically given two heaps and we have to concatenate the two heaps into single heap. The procedure or the logic is very simple; the idea is that H1 and H2 are the two heaps. Both are same max heap, if anyone is max heap other min heap that is also no issue. So it is depends on which heap you want to build?

So, suppose you want to max heap. So, start with anyone which is max heap, whatever it is there and then what we can do is that; we will delete node from one tree and insert this node into other tree and that procedure will continue until all nodes in the second heap is removed and inserted into the first heap. So this is the procedure actually, the trivial procedure it is we can follow it here.

(Refer Slide Time: 19:03)


Merge operation: An illustration
92
45
67
38
59
H_1: A max heap
13
19
80
92
93
96
H_2: A min heap
+
96
93
67
80
92
13
19
38
45
59
Resultant max heap after merging H_1 and H_2
NPTEL Online Certification Courses
IIT Kharagpur
IT 60101: Lecture 86
24

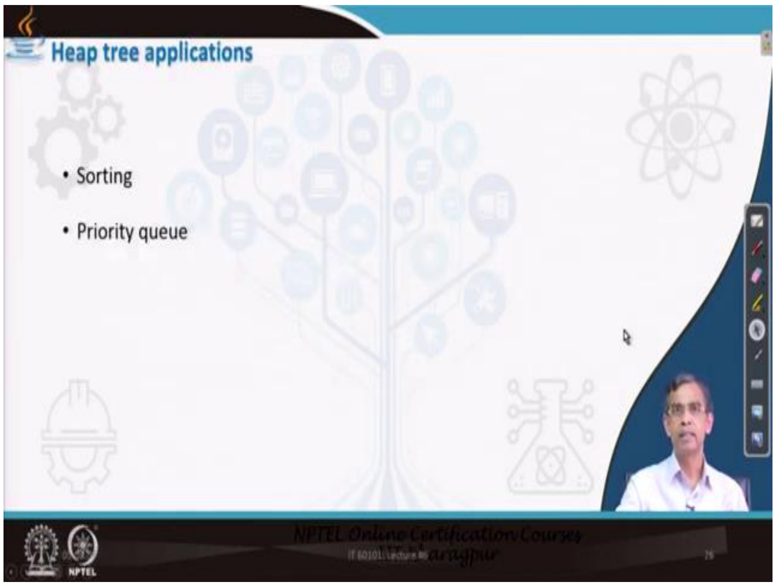
In this example, this is the one heap tree and this is another heap tree. And in this case if you see this is max heap and this is min heap, no issue. We will suppose, want to have the max heap, so we can start that and this is the target heap and this is basically the source heap. It means that, we will delete root node from this node and then insert it and then when we delete again, we revealed it and when we send it also, we revealed it.

So, re-heaping both things are there. Now on the other hand if we want to build a min, merging the two heap this max and min; so we can say that this is the target heap and this is the source. This means that we delete the node one at a time and then insert the node into this heap at a time and each time both the heap will be revealed or re-heaping and then finally this will be appeared as a min heap, where two heaps are merged. So, this is the procedure that we can consider for heapifying and then merging operation rather and there is an algorithm again; you can follow the book where the algorithm will find it.

(Refer Slide Time: 20.27)

Applications of Heap Tree

NPTEL Online Certification Courses
IIT Kharagpur



Heap tree applications

- Sorting
- Priority queue

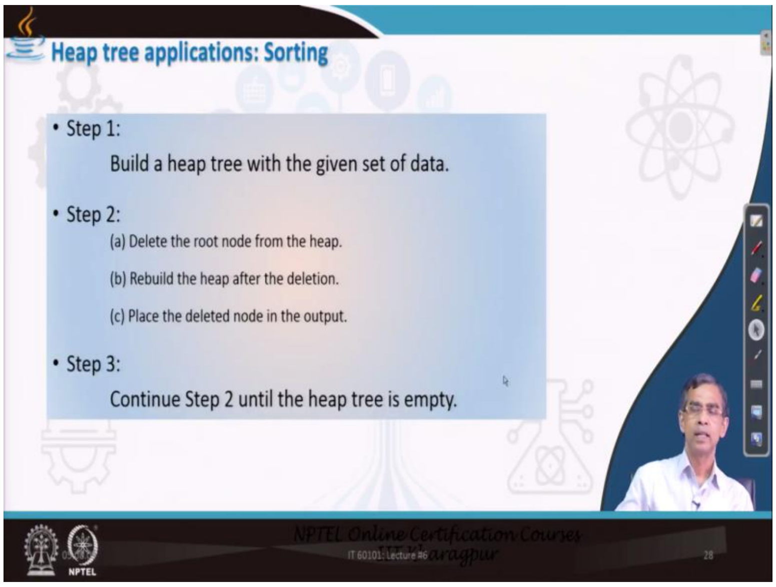
NPTEL Online Certification Courses
IIT Kharagpur

Now let us come to the application of heap tree, there are two applications; the sorting and then priority implementation.

(Refer Slide Time: 20:22)

Sorting Using Heap Tree

NPTEL Online Certification Courses
IIT Kharagpur



Heap tree applications: Sorting

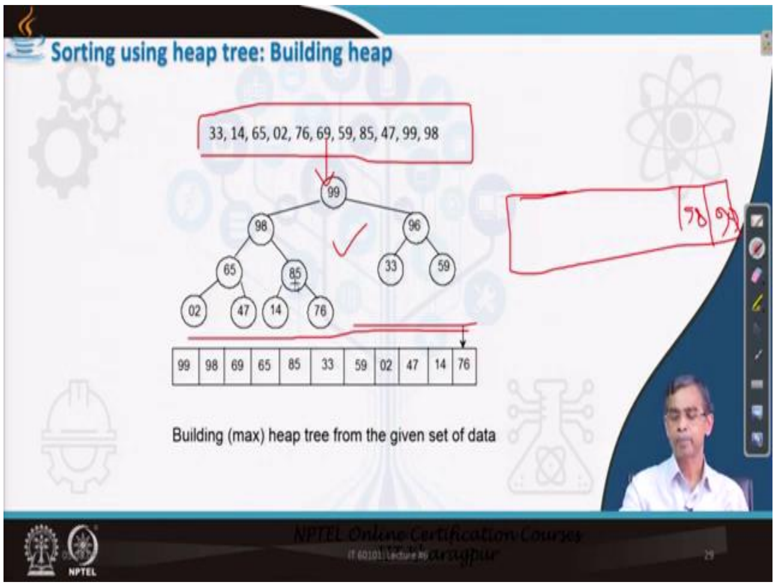
- Step 1:
  Build a heap tree with the given set of data.
- Step 2:
  (a) Delete the root node from the heap.
  (b) Rebuild the heap after the deletion.
  (c) Place the deleted node in the output.
- Step 3:
  Continue Step 2 until the heap tree is empty.

NPTEL Online Certification Courses
IIT Kharagpur
IT 60101: Lecture #6
28

Now let us come to the application of heap tree, there are two applications; the sorting and then priority implementation. Let us first consider the sort. Now you can start with mean heap or max heap, whatever it is there. Now if it is the max heap suppose, then the sorting procedure is very simple. What is the idea is that, each time you have to remove the root node and keep this deleted node into the output list at the end.

If we procedure the same procedure then that means delete re-node, root node each time and then deleted node should be placed in the output list from right to left. Then ultimately all nodes whenever deleted from the heap tree, it will produce the output list in an ascending order of the number.

(Refer Slide Time: 21.18)


Sorting using heap tree: Building heap

33, 14, 65, 02, 76, 69, 59, 85, 47, 99, 98

99	98	69	65	85	33	59	02	47	14	76

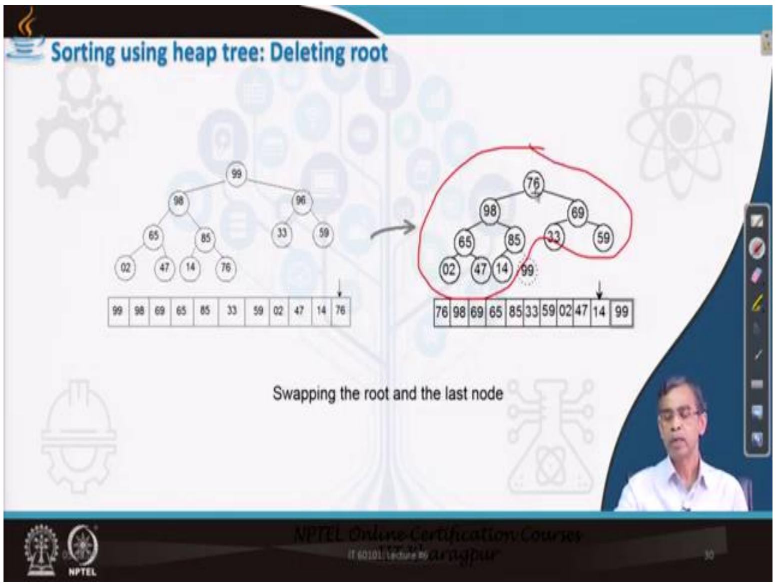
Building (max) heap tree from the given set of data

So, this is the procedure. Let us example, let us illustrate this procedure with an example. Now here is the list of elements that is given to you and we want to sort this list. So, what we should do is that first we create a heap given this list. So this is the heap tree that we got it there. So, this is the input list and then from the input list, we create a heap tree. Let it be the max heap and this basically shows the max heap from the input list.

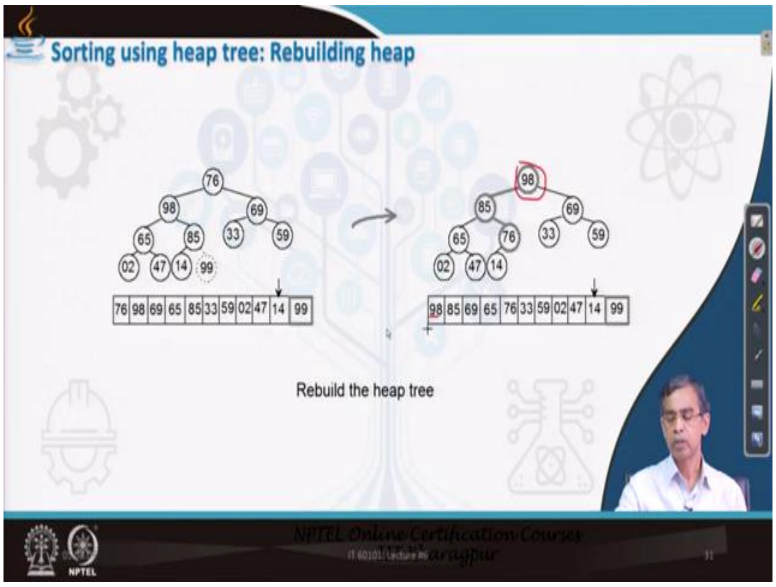
Then our next procedure of sorting is basically delete the root node, and then re-heap this, and then store this deleted node in the output list, so that means 99 will be removed, and 99 will be placed. There may be you can output list again, so this basically the output list. So, 99 will be placed here. Then we have to do, go for re-heaping. After reheaping, again you will see the largest node will come into the root node and let 98 will be the largest node, so 98 again delete from the, should be removed from the list and 99, 98 will come here right?

Again re-heaping and this procedure continue, how long? Till this heap tree is empty. So, this is the procedure of sorting actually. The logic, or technique or algorithm is very simple actually. It is basically delete, re-heaping, delete, re-heaping until the heap is empty and then storing after node is deleted into the output list.

(Refer Slide Time: 22.52)


Sorting using heap tree: Deleting root

Swapping the root and the last node


Sorting using heap tree: Rebuilding heap

Rebuild the heap tree

Now so, this is the example that you can say as I explained it that this 99 is removed. After removing ninety nine, we have to just simply store in the output list, but here I am telling you, storing in an output list although in the last example I gave that separate arrays is required but it is actually not required.

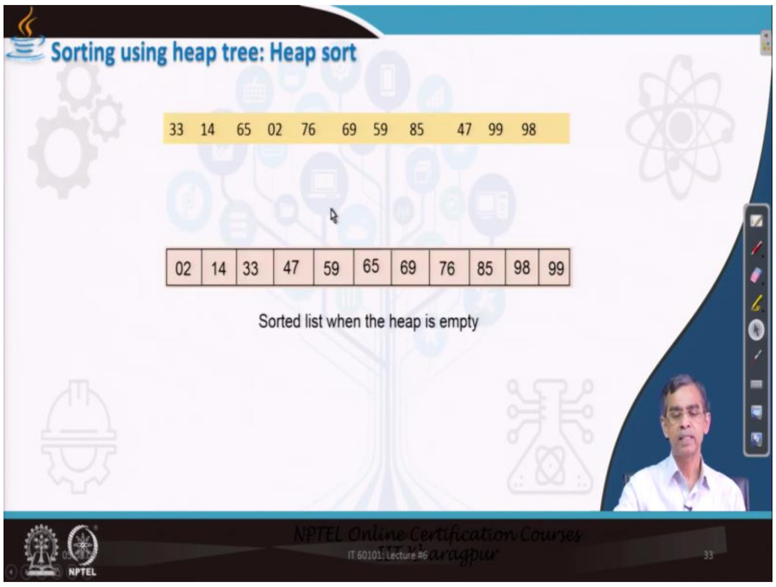
In the same array, where the heap is stored can be store the output array also. The idea is that, see 99 needs to be removed. So, what we can say is that this 99 should take place that place because 76 in the same order, it is come in the last place actually. So, what you can see is that this removing 99 is basically replacing 76. I mean 76 and 99 will be replaced; so what exactly the 99 will come to this place and 76 will go to the root place.

So 76 coming to the root place and 99, where it was in the root initially in the root goes to the last place, so this way it is there. Now, this become the idea about, then after ninety nine the rest of the part will be basically heapifying, so this part needs to be heapifying. So once 99 goes and 76, up this tree is to be heapifying.

So here 99 is removed from this tree and this basically the tree, which basically violates the heap property. So, again we have to go for re-heaping this property. Once we go for re-heaping, we will see that after the reheaping it become the heap it is. So, after the seventy six, 99 is removed; we need to re-heaping so re-heap is basically there. Now again in this re-heap we see this become the largest node so it coming to the root.

So, now again remove it, so 14 is the last node in this series. So, 14 should be placed in the place of 98 where it is there and then finally you can go for re-heaping and so on. So, this way the procedure will be continued and then ultimately we will be landed with the sorting list of the element. That mean heap which is stored in an array, that store is basically, this array stores the sorted elements in the list.

(Refer Slide Time: 25:06)



Sorting using heap tree: Heap sort

33	14	65	02	76	69	59	85	47	99	98

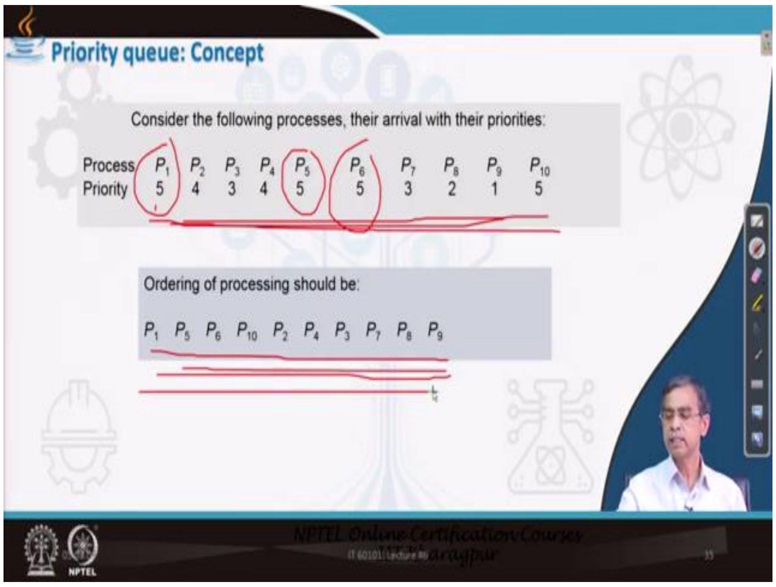
02	14	33	47	59	65	69	76	85	98	99

Sorted list when the heap is empty

So, this procedure is like this, so give this is the input and then finally after this procedure of heap tree application then it becomes sorting.

(Refer Slide Time: 25:1)

Priority Queue Using Heap Tree


Priority queue: Concept

Consider the following processes, their arrival with their priorities:

Process	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
Priority	5	4	3	4	5	5	3	2	1	5

Ordering of processing should be:

P_1 P_5 P_6 P_{10} P_2 P_4 P_3 P_7 P_8 P_9

Now let us quickly come to the discussion of priority queue, once we learn about the priority queue in the process seedling, the idea of the priority is that there may be many process can come into the queue and we have to sharp that process which has the highest priority. In fact, it is invalid, it is not exactly obeying the heap property or queue property called the last in-first out, first in-first out.

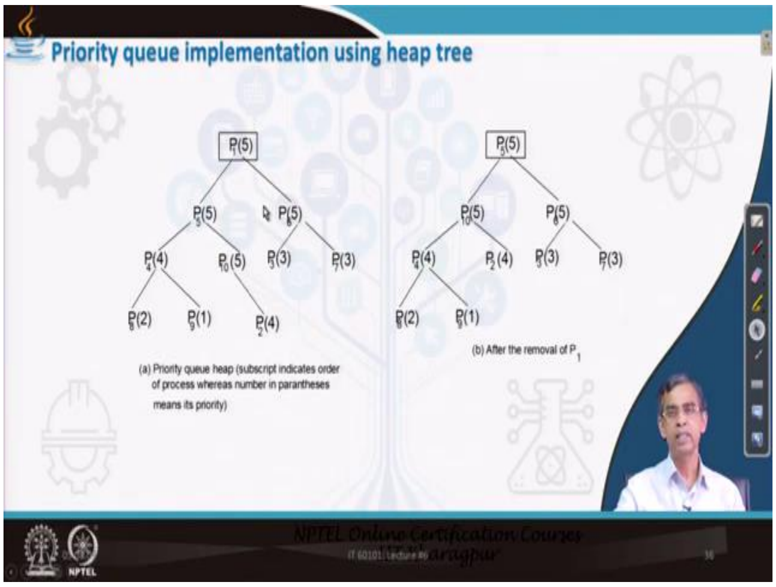
That is the queue property, it is there, but it does satisfy this first in-first out property but it basically satisfy that, the process or the element with the highest value should be processed fast that mean, it should be given the top priority actually. That is why the priority queue here, for example, in this example, here we see these are the different elements having their priority values like this one and they come in this order.

So, it is basically first come in and then this order. So, now out of these, this has the highest priority, this means that P1 will be served fast. Next to P1, although P2 arrives but they will not serve, rather P5 will be served and then P6 will be served and so on. So, the ordering in which process, the process will be summed, served, although they arrive in this order; it is shown here. This is basically, is the arrival of the process where as this is the order of servicing the processes and here you can see P9 will be processed last, whatever it is there.

Now this is the things that we have to follow that means this protocol has to be followed and then using the heap, we can follow this one. So, whenever the process are arriving, we can just adding them into the heap and then heap needs to be adjusted because after adding a process who is having highest priority needs to be adjusted again.

And if we go on adjusting then always the process which has the highest value will be in the root. Then the process which is at the root is to be processed fast and this way the process with highest priority gets service always fast and then this ordering can be enforced. So, this is the idea about the priority key implementation and that is the implementation using heap tree.

(Refer Slide Time: 27:44)



Priority queue implementation using heap tree

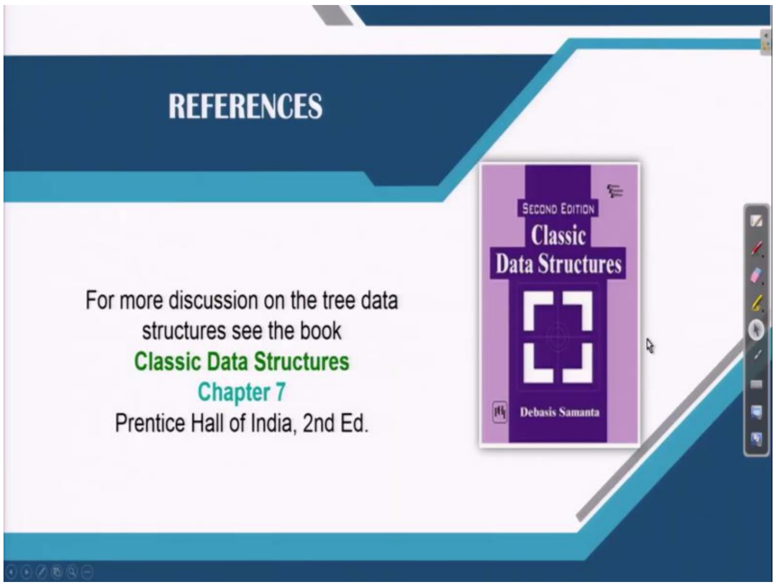
(a) Priority queue heap (subscript indicates order of process whereas number in parantheses means its priority)

(b) After the removal of P_1

So, here basically you see whenever the elements are coming; so elements are added into the heap and when we add the elements into the heap, they can violate the heap properties. So, just simply adjust the heap till it violates the property and this way, all the time new nodes, new process comes, it added into the queue and queue added in the heap and heap always maintains in the order in which they should served and that is the heap application that is there.

So, basic idea it is that; all the process should be stored in the form of a heap according to their priority values actually and then we will serve the process starting from the process which is at the root. Once the process is served, we have to go for reheapifying and for this purpose we will always consider max heap.

(Refer Slide Time: 28:40)



REFERENCES

For more discussion on the tree data structures see the book
Classic Data Structures
Chapter 7
Prentice Hall of India, 2nd Ed.

So, this is the property and for further algorithms and discussion you can follow this book. And in our next video lectures we will have a programming experience about this heap tree. How a heap tree can be stored and how heapifying can be revealed or, heapify and re-heaping can also be done and both max heap and min heap will be considered. So, we will have some programming experience and programming experience for heap tree only will be discussed in our next video. Thank you very much.