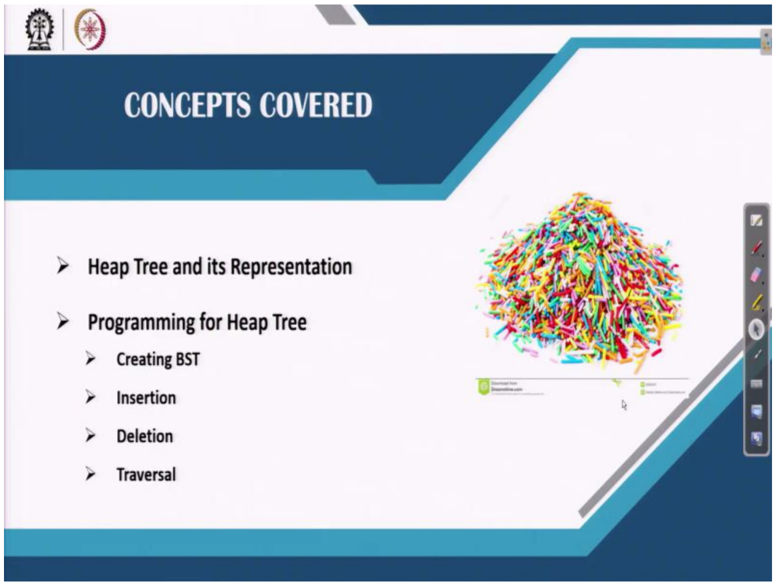
Data Structure and Algorithms using Java
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Lecture: 33
Topic Programming for Heap Tree

In this video lecture, we will discuss how a heap tree can be programmed. Now so far the programming of the Heap tree is concerned, basically what we have to do is that, we have to create a Heap. And Heap is basically a collection, is a collection of elements. So we will see how a Heap can be created. And in our last lecture discussion we mentioned that Heap is always stored in the form of an array. So a collection is an array.

(Refer Slide Time: 1:03)



CONCEPTS COVERED

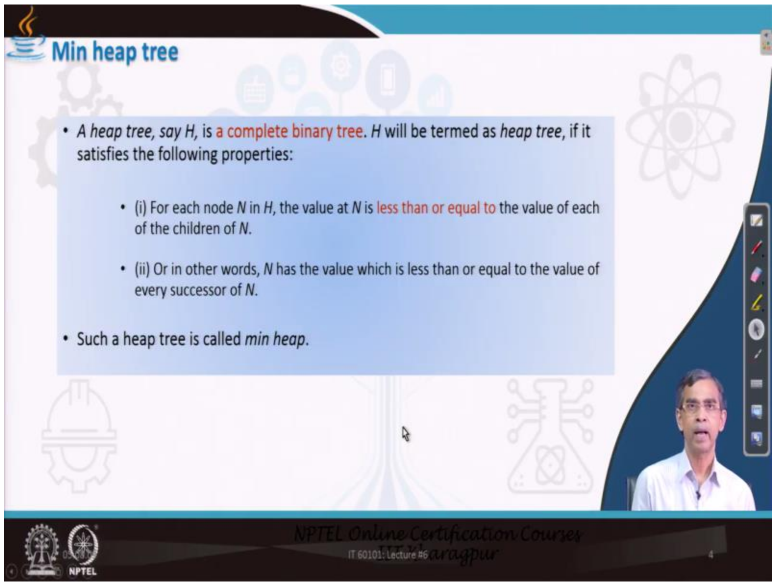
- Heap Tree and its Representation
- Programming for Heap Tree
  - Creating BST
  - Insertion
  - Deletion
  - Traversal

So the first task is basically, how a heap can be created, that mean heap can be stored and then the different operations of heap needs to be implemented. And all those operations are to be added into the class of the heap in the form of methods. The methods are insertion, deletion, traversal, merging like this one. Let us see how all those operations and then data structures that can be considered to write a program so that we can store in the form of a heap.

(Refer Slide Time: 1:37)

Quick Review of Heap Tree

NPTEL Online Certification Courses
IIT Kharagpur


Min heap tree

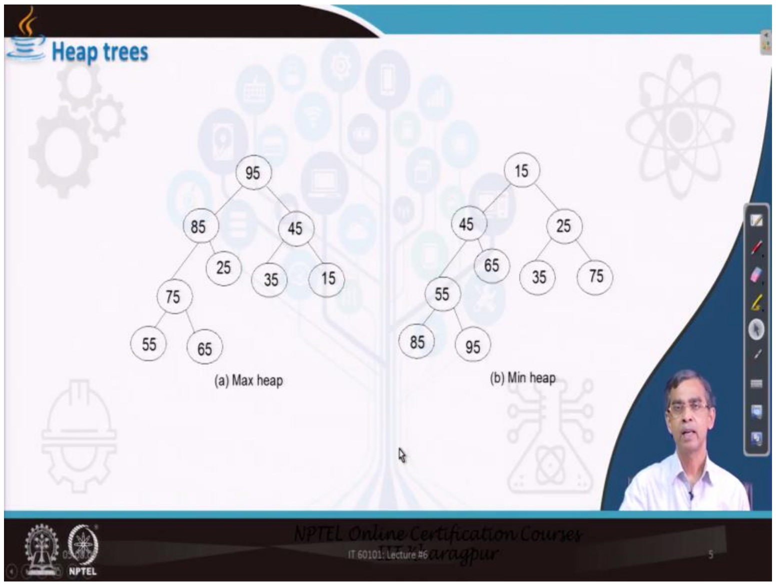
- *A heap tree, say H,* is a complete binary tree. *H* will be termed as *heap tree*, if it satisfies the following properties:
    - (i) For each node *N* in *H*, the value at *N* is less than or equal to the value of each of the children of *N*.
    - (ii) Or in other words, *N* has the value which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

Now we have already learned about two types of Heap, the Max Heap and the Min Heap, so we will consider only the Min heap for the programming here, and you can follow the same procedure that we will discuss in this discussion, and you can repeat this work for the Max heap also.

So it is very easy actually, but what are the changes you have to do, that you can think and then you will be able to do I believe. So you will concentrate on the Min heap only, how the Min heap can be created and then how the different operations for the Minheap can be implemented.

(Refer Slide Time: 2:13)


Heap trees

Now different operations means as you know building a heap, build heap like and another is called re heapifying that means heapifying rather we can say, or reheaping. So this is basically once you insert or delete a node you have to go for heapifying it actually. And so this is a procedure, now this is the Min heap that we have already discussed about it.

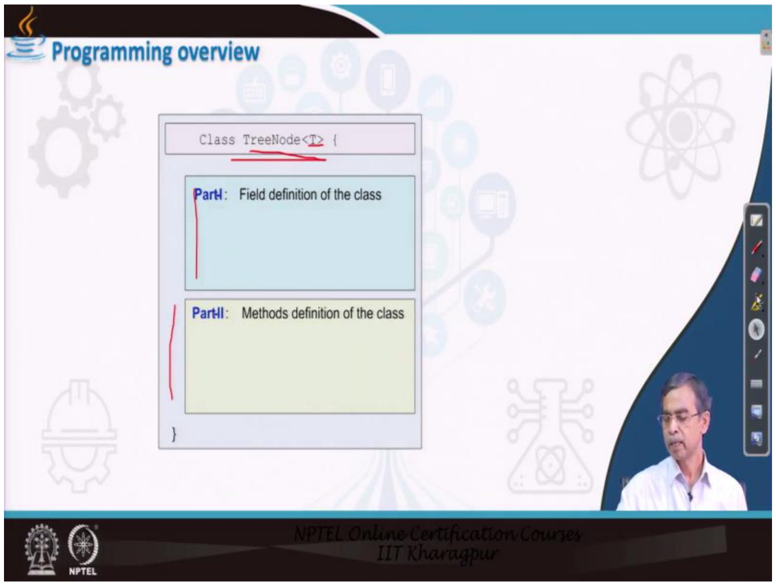
(Refer Slide Time: 2:38)

Programming for Heap Tree

Now, let us come to the discussion of programming. So we have to write a program, and in Java we have to basically declare a class and then declaring a class means the Name, type, what is the other things are there and then defining that class. So far the class definition is concern we have to add the fields and different methods are there.

Again likewise the previous programmings for the different data structure that we have already examined and studied, here also we will consider generic, generic programming that means the heap tree that we want to define it should be generic, this means that the heap can store, any type of Data including numbers and then strings and whatever it is there. But we will limit our discussion to numbers only.

(Refer Slide Time: 3:33)



Programming overview

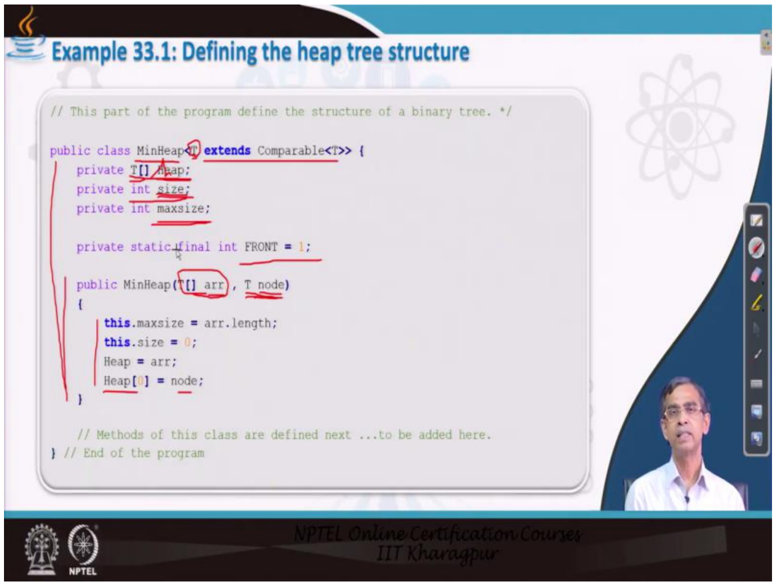
```
Class TreeNode<T> {
    Part-I:  Field definition of the class
    Part-II: Methods definition of the class
}
```

Now, here again this the overview of the class that we are going to build, so it should have the name of the class and as I said it is a generic class. So this is a generic type T, and these are the different fields that we have to consider. And the different methods, that is related to this heap tree data structures to be added here. This is basically same concept that we have also started for the binary search tree, stack queue and everything the same line of the programming actually. And this is the best way of doing programs for data structure actually.

(Refer Slide Time: 4:21)

Defining Structure of Heap Tree

NPTEL Online Certification Courses
IIT Kharagpur



Example 33.1: Defining the heap tree structure

```
// This part of the program define the structure of a binary tree. */

public class MinHeap<T extends Comparable<T>> {
    private T[] Heap;
    private int size;
    private int maxsize;

    private static final int FRONT = 1;

    public MinHeap(T[] arr , T node)
    {
        this.maxsize = arr.length;
        this.size = 0;
        Heap = arr;
        Heap[0] = node;
    }

    // Methods of this class are defined next ...to be added here.
} // End of the program
```

NPTEL Online Certification Courses
IIT Kharagpur

Now let us see, how we can define a class for a heap. So this basically you have to declare the class first and here is a small program that you can think. So this is the first step towards the programming for heap. Now here you can see, we are declaring a class and we give the name of the class as Min heap.

And then T is the type that the heap will store, it can store any numbers including integer long float, sort, and that is why it is as it is. So T, if you want to limit the T right, to numbers only so we just extends comparable T. That means should be of type comparable. All the numeric values are of type comparable, string is also of type comparable.

This means that this T will be valid for numeric any types long, float, sort, Int, doubles and including string because these are the basically comparable type. Now as we mentioned that a heap will be stored in the form of an array, so we have to declare an array. So we declare T is array an and this is the name of the array say heap, and this is basically size, how many elements that you want to store into this array. Let it be size and maximum size is basically, what is the maximum capacity of the heap.

So you can allocate T array heap as a size hundred, but ultimately in your application may be 50 or 30. Elements are to be stored so it is basically size. Size and maxsize is basically, maxsize is a capacity; where size is the total number of elements that is present in the current heap. And we declare a temporary constant called FRONT that is required to check that from where the root will be (())(6:11). So initially it is the starting location of the heap, assuming that it is start from 1.

Anyway, so this is the starting node. Now. we will declare one constructer to initialize a heap. So here is the constructer. For this constructer we will pass an array location that means where this array will be stored. For example, we want to declare a new heap into a new array A, next time you want to declare in the same program, you have to declare another heap into another array B, so this is the name of the array needs to be passed.

And this is the element which basically needs to be passed as a initial value into the heap. So this is basically first value that needs to be added into the heap. So initially our heap will content this element only. So the initial heap will be, I mean, while we are starting, or started, while we start building a heap we will start with a single node actually, so this the value of the node that initial heap will contain.

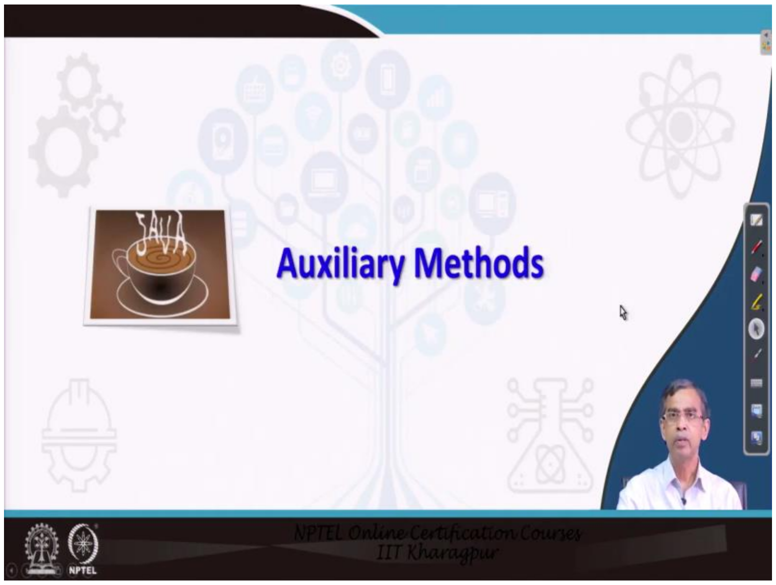
So this is the assumption that we have assumed it, and that is all. These are the other initial value as we have said. So this is basically, and this node is stored at the heap 0, which is basically array in our case. So this declaration you can see, okay, so this basically is the declaration that is required with constructer is basically to create an instance of heap.

So, when heap is created, but now we have to add different methods into this heap structures so that early heap manipulation is possible. So regarding the heap manipulation means we have to delete insert. Now, while we go for deleting it basically involved as a consequence that reheapifying. Similarly, insertion also will be associative with reheapifying. That is a procedure that is okay, and if we use this process of insertion starting from empty heap like starting one node at a time and everything, so it will basically starting, it is basically inserting one node at a time.

And then we have to repeat, we have to reheapfying every time and therefore our heap will grow and then process also while the heap is growing we can again perform any deletion or any other operation as you wish. We will see exactly how the different manipulation of a heap structure can be done using a master program after defining all methods that is required to manipulate heap.

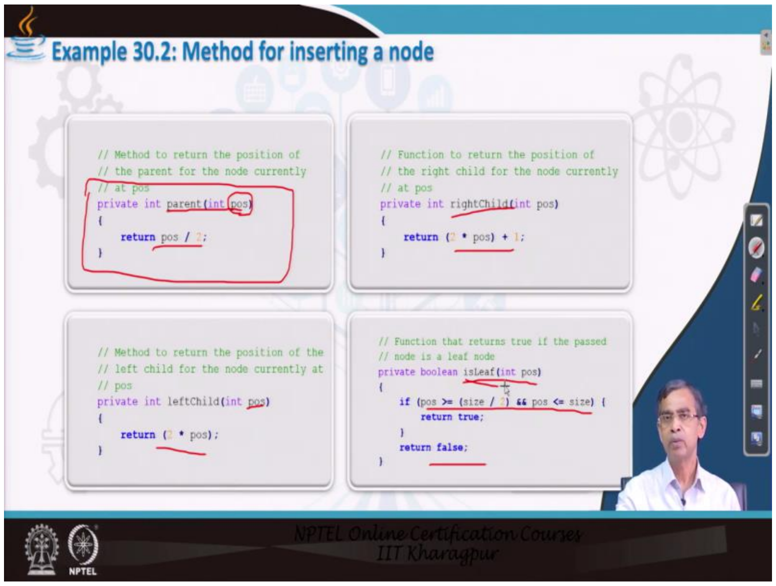
(Refer Slide Time: 9:05)

Defining Methods of Heap Tree



Auxiliary Methods

NPTEL Online Certification Courses
IIT Kharagpur



Example 30.2: Method for inserting a node

```
// Method to return the position of
// the parent for the node currently
// at pos
private int parent(int pos)
{
    return pos / 2;
}
```

```
// Function to return the position of
// the right child for the node currently
// at pos
private int rightChild(int pos)
{
    return (2 * pos) + 1;
}
```

```
// Method to return the position of the
// left child for the node currently at
// pos
private int leftChild(int pos)
{
    return (2 * pos);
}
```

```
// Function that returns true if the passed
// node is a leaf node
private boolean isLeaf(int pos)
{
    if (pos >= (size / 2) && pos <= size) {
        return true;
    }
    return false;
}
```

NPTEL Online Certification Courses
IIT Kharagpur

Now let us define all the methods those are important in the context of programming heap. So there are many methods are involved. We will first discuss simple method which we will call, our main method, so it is a Auxiliary method you can say. First auxiliary method is required is that if a node is given, then we will be able to calculate where its parents is because sometimes we can, we can see whenever we are going for reheapfying we have to go from child node to know its parents node. So given a node as a node then we, it sometimes require that which is the parent node.

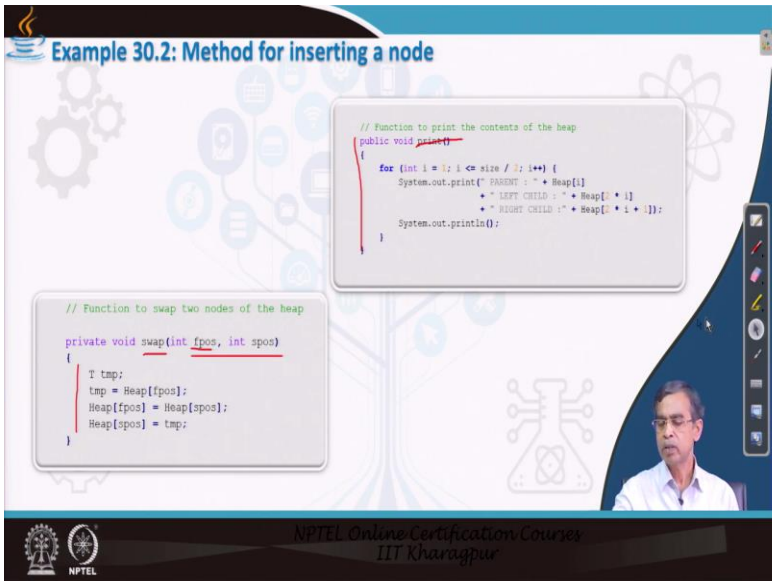
So this formula, this method can be called for a child node. This method can be called for a child node to know that what is the location of each parent node. So, if child node is in the position pos then its root node, I mean its parents will be calculated by this formula, this obvious formula for the storing of elements into an array that is the condition that it is there.

If i'th node is the node, then its two children will be storing as 2i and 2i plus1. Opposite to A if i'th node is the children, then its parent will be available at I by 2, with floor, so this is the concept it is there. So pos deep 2 always give the floor value. Now here again, if a given node is at pos location, then its left child can be calculated using 2 star pos, it is basically I 2 star I like. And then right child also can be calculated using this formula.

Now whether a node is Leaf node or not, that also can be calculated by this formula that say suppose a node is in the pos location, and if this condition is satisfy, then we can say that this is a Leaf node otherwise it is not a Leaf node. So isLeaf sometimes it needs to be checked, because when we want to insert, we want to insert into the last level as a last node as a Leaf node.

So that is why it is required, so that we have to check it and then we can think about it. So these are the methods, which are basically swap methods, so we can, and that can be defined in the class, the Min heap class that we are, we have declared there as a part of the method actually or what else. So this is basically the different methods it is required.

(Refer Slide Time: 11:37)



Example 30.2: Method for inserting a node

```
// Function to print the contents of the heap
public void print()
{
    for (int i = 1; i <= size / 2; i++) {
        System.out.print(" PARENT : " + Heap[i]
                         + " LEFT CHILD : " + Heap[2 * i]
                         + " RIGHT CHILD :" + Heap[2 * i + 1]);
        System.out.println();
    }
}
```

```
// Function to swap two nodes of the heap

private void swap(int fpos, int spos)
{
    T tmp;
    tmp = Heap[fpos];
    Heap[fpos] = Heap[spos];
    Heap[spos] = tmp;
}
```

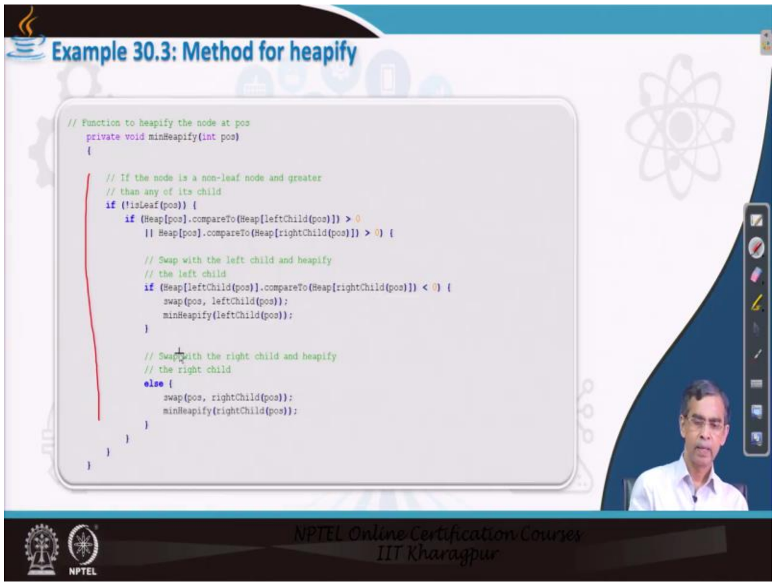
Now, we will come to the Core method. So this is basically the Auxiliary method. Then Core method, okay, there are few more Core method is required also, auxiliary method is required. Here is a swap. You can recall whenever we are re-heapifying, then each time we have to inter change elements from the child to its parents, and in that process will go starting from the point of insertion, to the root node or starting from the point of deletion to the Leaf node whatever it is there.

Now so for this process actually we need to interchange the elements between the parent and child and vise-versa. So for this process we can define another method, this method is called the swap method. So it is basically swap between two position, this is may be parent and this is child, so it is there. So this are swap routine can be implemented like this, this is a usual swap routine that is there.

And then finally we, if we want to print the entire heap at any instant and this is the print method that we have defined can be used to print all the elements in the heap tree actually. And these are the methods as I said that these are the auxiliary methods, which required to define our core method.

(Refer Slide Time: 12:55)

Method for Heapify



Example 30.3: Method for heapify

```
// Function to heapify the node at pos
    private void minHeapify(int pos)
    {

        // If the node is a non-leaf node and greater
        // than any of its child
        if (!isLeaf(pos)) {
            if (Heap[pos].compareTo(Heap[leftChild(pos)]) > 0
                || Heap[pos].compareTo(Heap[rightChild(pos)]) > 0) {

                // Swap with the left child and heapify
                // the left child
                if (Heap[leftChild(pos)].compareTo(Heap[rightChild(pos)]) < 0) {
                    swap(pos, leftChild(pos));
                    minHeapify(leftChild(pos));
                }

                // Swap with the right child and heapify
                // the right child
                else {
                    swap(pos, rightChild(pos));
                    minHeapify(rightChild(pos));
                }
            }
        }
    }
```

Now, let us define our Core method that is actually for Heapify. Now heapify can be called when inserted node into the existing heap or we delete a, delete the root node from the heap. So this is a heapify method and I have written a code for you here, the heapify method and it is in the context of minHeap so this why I gave the name as minHeapify.

Likewise you can little bit change, only minor changes are required to convert this routine, this method into the maxheapify. I left it as an exercise for you, if you understand the concept I am sure that you can follow, this method this code in the minHeapify method, to write your own code as the maxheapify method.

Now, here is the procedure of max heap actually, as I told you if you have to understand the logic first then the algorithm behind this code also you have to have it and that you can find from the book that I have given the reference as well as the link that I have used for study materials in this course. So all this things needs to be consulted and then if we follow this code because unnecessary coding illustration is very difficult.

I have written he code and checked that the code is working and then you can check that whether this code is working according to a particular algorithm and the logic that is required in order to heapify a Min heap tree. And if you follow this thing all this things will be clearly understandable. So, I do not want to discuss in details about the, details about the code for this and in this and then you have to have to go through with your patience, your time so that you can understand it.

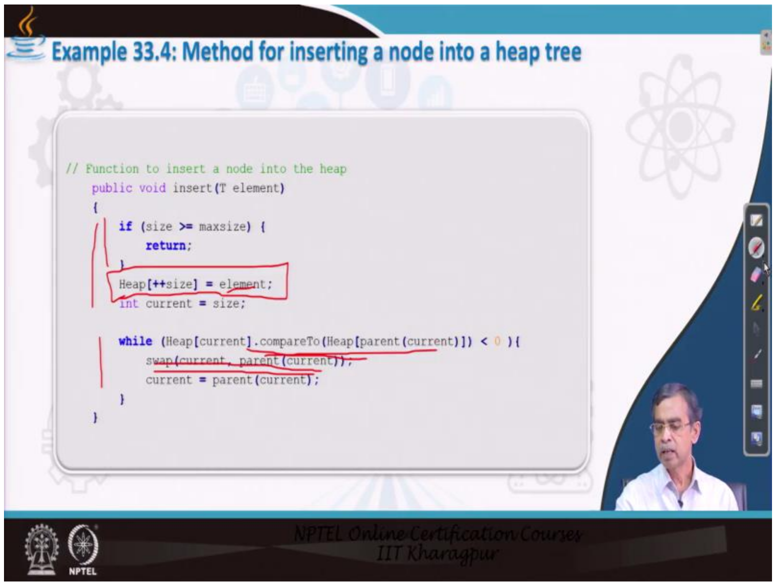
But exactly the thing that you should do is, basically you try to understand the logic first. Then the algorithm that is available here and there. And then try to go through the code. And finally, if you run the programs, then you will be able to understand how it is working. Last step that you should do is that, little bit do some modification according to your own wish.

There are several modifications are possible like; we have built the Min heap so another possible, what it is called the rewriting the program can be Max heap and then Max heapify. Then you can think about that other data structure that we want to create it. Other data structure means, the heap that we have used for the numbers. Now, let us see, whether we can use this heap for the any other user define type or not.

Then the next step that you can think about different application. So, how Heap can be used for sorting, how Heap can be used for implementing, process scheduling, which we have done for the, while we are discussing the queue. The same thing you can do but using queue also. So these are the different what is called the revisions in the programming that you can think for to understand and then later to acquire the skill, in programming, particularly logic, and then data structure and implementing the data structure and algorithm writing your own program. Anyways so these are the basic procedure that, okay yeah my advice for you is to follow it.

(Refer Slide Time: 16:28)

Method for Insertion



Example 33.4: Method for inserting a node into a heap tree

```
// Function to insert a node into the heap
  public void insert(T element)
  {
      if (size >= maxsize) {
          return;
      }
      Heap[++size] = element;
      int current = size;

      while (Heap[current].compareTo(Heap[parent(current)]) < 0 ){
          swap(current, parent(current));
          current = parent(current);
      }
  }
```

Now, let us proceed further to discuss the few more topics is there. And so we have discussed about the reheapifying. But reheapifying is associated with Insertion and Deletion. I will just discuss about the insert. So basically at any instant given an heap tree, it can initially empty whatever it be, so we have to insert a node into heap. Again this insertion I will discuss about the program, concept of min heap.

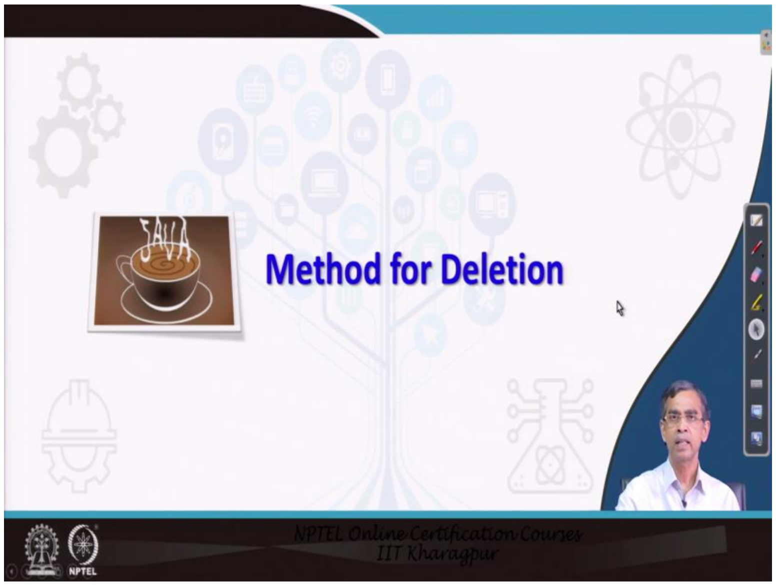
So in case of insertion first of all if we have to say that what is the max size is already achieved the full quota or not, if it is there then insertion is not possible. In that case we can say that overfull like. Otherwise we can go for inserting the element and you see, whenever the new node is inserted, it should be inserted at the last level as a last node. And this is basically, size is the pointer will be used to store that what is the maximum size, the largest element, the last element is basically at the location size actually.

So, it basically the new element, if it is a new element is element, then it should be placed into the size location actually, and after this placing the size will be automatically increased by 1. Now, if size is equals to max size, then that is basically the heap is full. And if the size is greater than max size, you can say that the heap is overfull. That means there is no more space to add any more node.

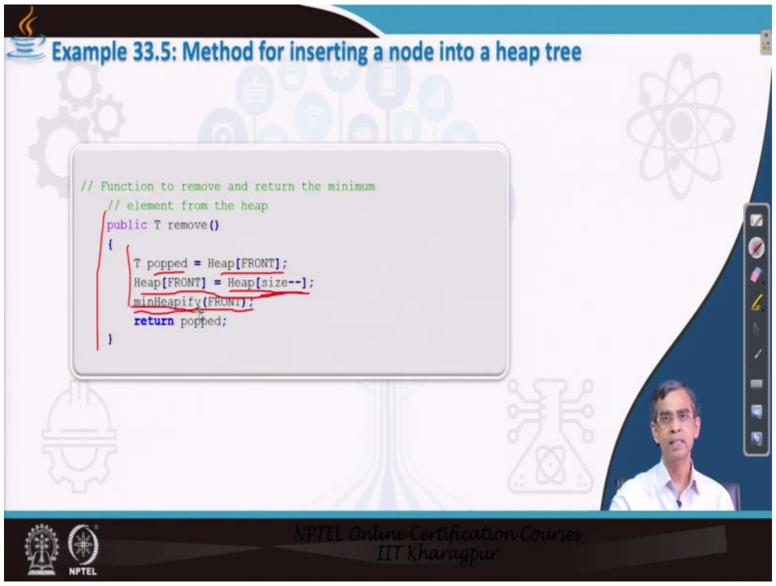
So it is there. And so this is basically check that whether heap is full or not, and if it s not full, then we will go for inserting the node and while we are going to inserting this node you see, we basically swap the node from the child to parent considering if they are comparable. Now here compared to method is called, so it is basically compared to the current position to its parent. If they are not in order we will just go for swapping.

So the swapping is called here after the comparing. So that means we have to compare from the current point of insertion to its parents, then next, parent to its parents' parents and it will continue till we reach to the root node and this way it will continue. So this basically is a process towards the reheapifying we can say. So this is the insertion procedure and insertion is really is very easy in case of heap tree actually. As we say that this is the procedure that we can follow.

(Refer Slide Time: 18:55)


Method for Deletion

NPTEL Online Certification Courses
IIT Kharagpur


Example 33.5: Method for inserting a node into a heap tree

```
// Function to remove and return the minimum
    // element from the heap
    public T remove()
    {
        T popped = Heap[FRONT];
        Heap[FRONT] = Heap[size--];
        minHeapify(FRONT);
        return popped;
    }
```

NPTEL Online Certification Courses
IIT Kharagpur

Now next is deletion, how the deletion operation can be takes place in case of heap tree. And as I told you, deletion of any node is possible from the heap, but from the practical usage point of view, deleting a root node in fact matters. So that is why we will consider deletion of a heap, deletion operation in a heap means we are deleting the root node.

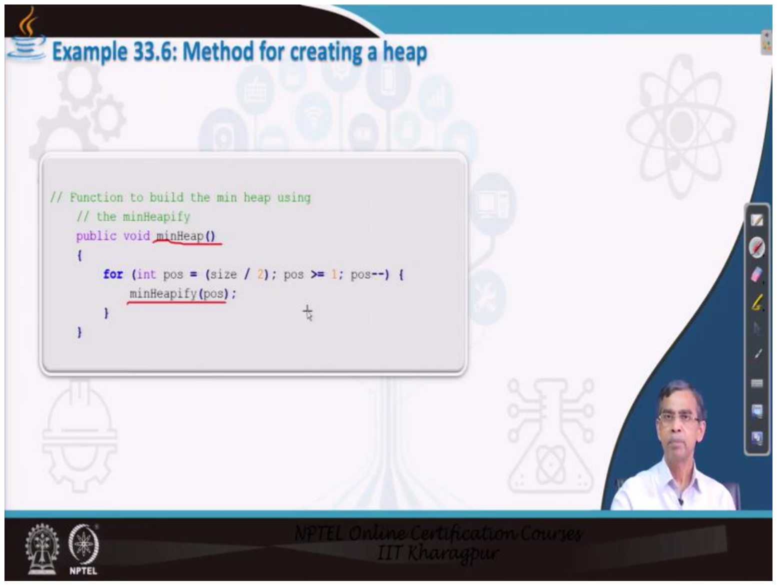
Now, so there is a method again, this basically will be considered about deleting. So we have to delete means we have to delete the, we have to delete the root node. So root node is basically is at the FRONT like, so it is deleted, and once the node is deleted size should be decremented by 1.

And actually here if you see the deletion of a node is basically swapping or basically replacing the root node by the last node in the last level. So it is basically last node in the last level and we just replace it. And this process basically, automatically remove this one. But we have to copy the node first to be used for other purpose, so this basically heap FRONT copied, is basically popped is the location where it will be copied.

And then we have to go for minHeapify, that minHeapify method we have already discussed. Because once we do it, we have to again min call the minHeapify. So this is the method for the minHeapify. And this basically the deletion. So these are fewer lines of code that is required to this one, but it is not exactly fewer lines of code because this code call other codes minHeapify, which is basically we have already discussed earlier and we have to do it. So this is a procedure of deleting a node from a heap.

(Refer Slide Time: 20:50)

Method for Building a (Min) Heap



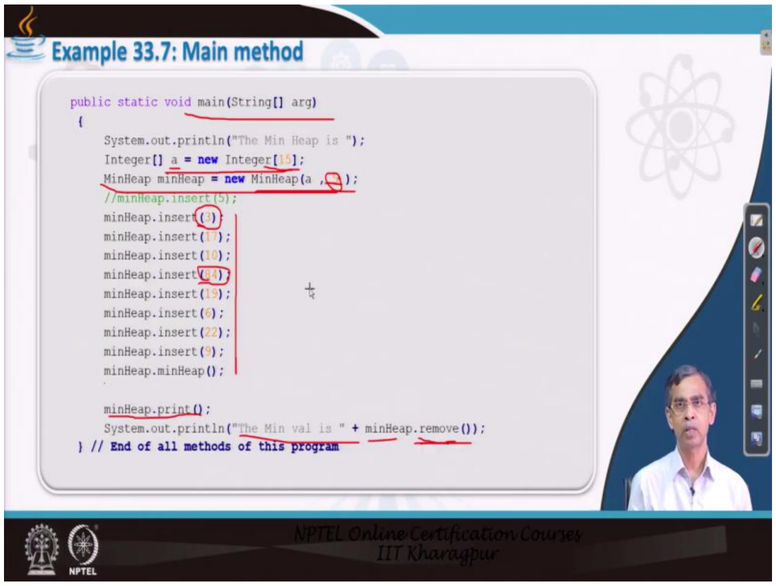
Example 33.6: Method for creating a heap

```
// Function to build the min heap using
    // the minHeapify
    public void minHeap()
    {
        for (int pos = (size / 2); pos >= 1; pos--) {
            minHeapify(pos);
        }
    }
```

And now methods of building a heap it is basically, the idea is that it has given a list, we have add the element into the heap, and satisfying the minHeap property that is why building a heap and for which there is a simple algorithm that you can consider. So it is basically we can the method minHeap, it will build the heap then it is basically go on, starting from the, starting location to each location and calling the minHeapify each time. So we take one element from the input list and call this method and we insert it and in this way heapifying, I mean building heap or heapifying will be takes place.

(Refer Slide Time: 21:38)

Working with Heap Tree



Example 33.7: Main method

```
public static void main(String[] arg)
 {
     System.out.println("The Min Heap is ");
     Integer[] a = new Integer[15];
     MinHeap minHeap = new MinHeap(a ,5);
     //minHeap.insert(5);
     minHeap.insert(3);
     minHeap.insert(17);
     minHeap.insert(10);
     minHeap.insert(84);
     minHeap.insert(19);
     minHeap.insert(6);
     minHeap.insert(22);
     minHeap.insert(9);
     minHeap.minHeap();

     minHeap.print();
     System.out.println("The Min val is " + minHeap.remove());
 } // End of all methods of this program
```

Now, we will, now we have done all methods those are required to define our minHeap class, now we can check whether all operations are working properly or not. So, one master program is required now here is the master program that we can think about it. So this master program demonstrate starting from an empty list. How we can go on adding num, elements into it and in between we can, if we wish any other sort of operation like deletion and all this things are there.

Now here, you just take the code, it is a very easy code actually. This is the main method we are declaring. So, main method can be added into the class declaration that we have discussed. So this basically in continuation of that class, otherwise you can write separate program where main you can define and those class you can import. Now here we declare A is an array, this is the array of input actually, A5 and then we just call the minHeap methods this is basically the constructer, which basically create the heap initially with 5 is the starting element into it. And size of the array A, this is max size we can say is 15.

Now, so initially the heap is created with the first node in it. The first node stored the value 5. And then we can call the ship, a number of couple of insert method here inserting the number in any order into the heap. I have given this order, you need not to follow the same order, you can give in any order, sorted order, descending order, random order, whatever it is there.

And after these things we can just call the print method, here, for example, we can call the print method and this print method will basically print the entire heap at that moment. So this is the idea about it, and there is a remove, actually this a delete method, you can call. Then it basically delete means a we do not have to pass any arguments because delete always takes place deleting the root node. It return the root node that it deletes.

So out of these elements if you see, the largest value 84 and then if you remove it, it will return the, sorry it is Minheap we are discussing, the smallest element in this heap is basically 3, so whenever you remove it, it remove the smallest element that (())(24:19) so this will basically print you the result 3 actually it is there.

Now this again master program can be continued by calling few more insertion method from it and then calling deletion whatever the way, so this means that in your main application you can use your class program that is basically the heap program, and then you can apply this heap in your different... For example, sorting or process scheduling, this one. So now you can call this program or input this implementation into a sorting program and the sorting program can be written.

(Refer Slide Time: 24:58)

REFERENCES

- Classic Data Structures, Debasis Samanta, 2nd Edition, Prentice Hall of India
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html

Now regarding this sorting we will discuss the application of heap while we will discuss the sorting algorithms later on. So for timing I am not discussing the sorting program in this context. Now, so heaps tree, heap tree like by binary search tree is very important structure and I have given the link here. So this is the link that you can think for and this the book where you can find full details about the discussion of heap.

And this is the link that you can think about where all study material are available for you. So I advise you to follow the two links for your understanding and that makes your study complete, and if you have any doubts regarding these things you can write, put your question in the discussion forum for resolving all the doubts or… And thank you, thank you for your attention.