

**Data Structures and Algorithms using Java**  
**Professor. Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture No. 51**  
**Improved Sorting Algorithms**

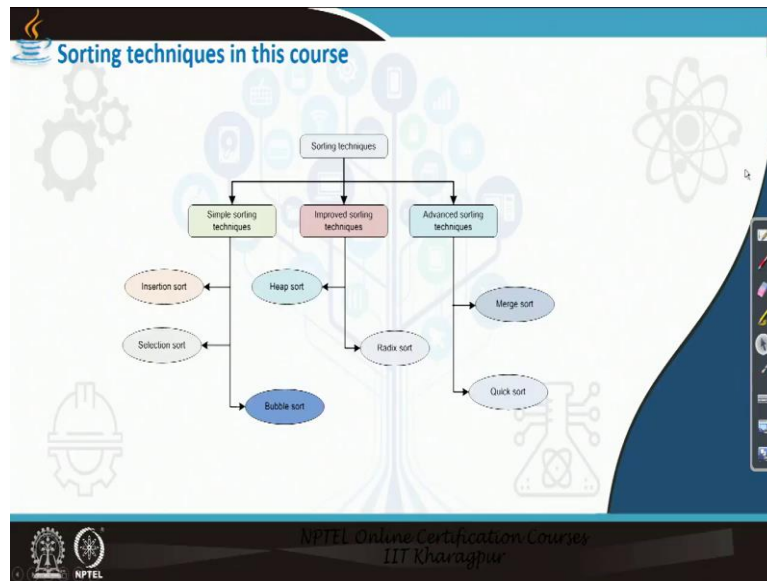
Right now, we are in module fourteenth and this lecture is in continuation with our previous discussion.

(Refer Slide Time: 0:35)



So, today we will cover a few more sorting algorithms, mainly two sorting algorithms, the Heap Sort and Radix Sort we will cover. The heap sort is very unique because it is a nonlinear sorting techniques, but it store the data on an array. Radix sort is again a totally unique sorting algorithms, where actually we do not have to do any key comparison. However, it is a memory hungry sorting technique, but it is very fast, one fastest sorting technique we can tell. So, today's topic includes these two sorting algorithms in details.

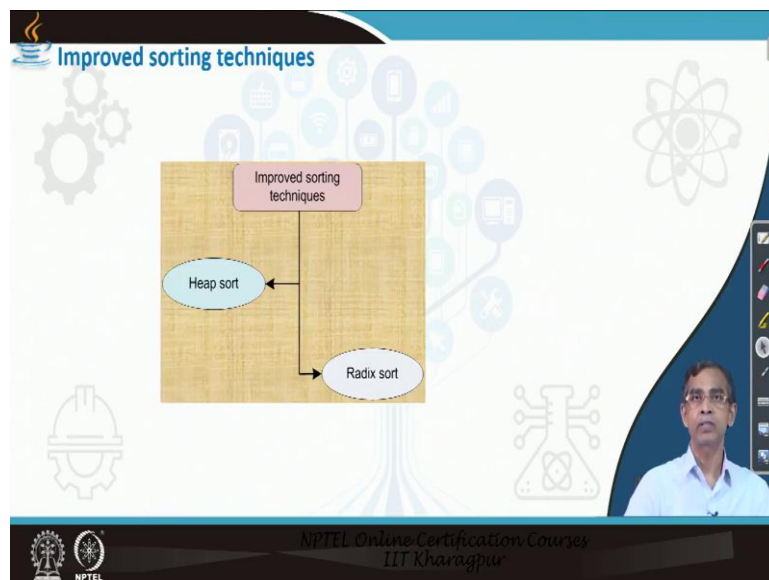
(Refer Slide Time: 1:17)



Now, so, we have discussed about insertion sort, selection sort and bubble sort in the last video lectures. So, today video lectures includes these two sorting algorithm and other two sorting algorithms we will be discuss in next part of the lecture. So, these two techniques is, we termed as improved sorting algorithms, because somehow it works better than the previously discussed sorting technique like insertion sort, selection sort or bubble sort.

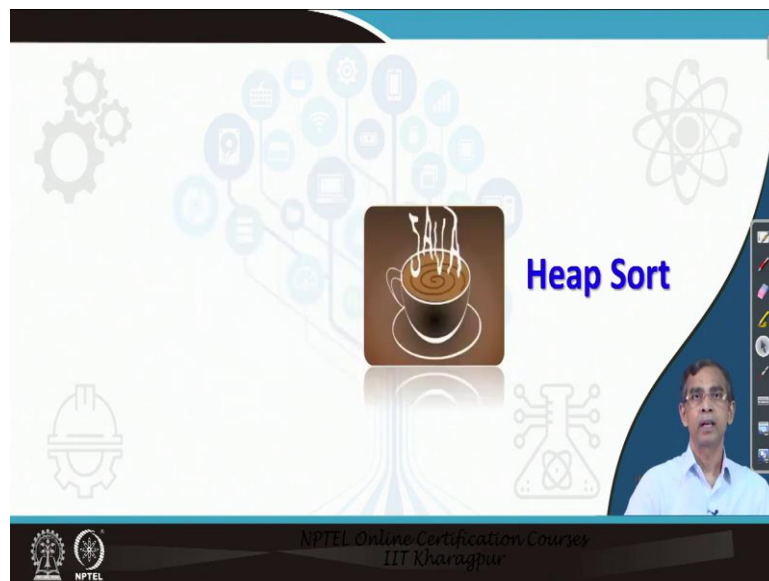
However, the (previo) those are the sorting technique that is called a simple sorting technique is the sorting techniques which known at the beginning and still it is basically an interesting sorting technique for the students, because they can exercise their programming skill while they try to implement this sorting algorithm. And they are simple because only simple data structure that is used that is why they are called simple sorting techniques.

(Refer Slide Time: 2:22)



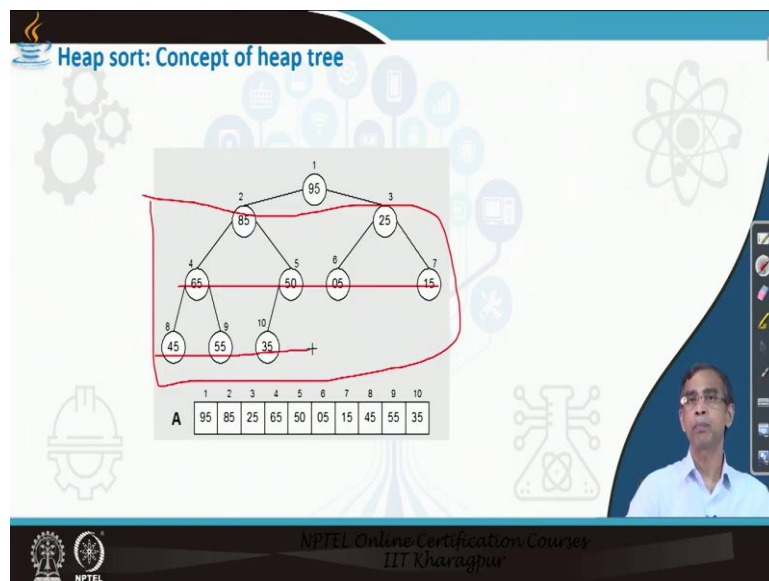
Now, let us consider the next version of the sorting techniques comparatively better than the previously discussed sorting techniques.

(Refer Slide Time: 2:34)



First we will discuss about the heap sort. Now, heap sort we have discussed about while we are discussing bit about heap data that is the heap tree a binary tree, it is called the heap tree.

(Refer Slide Time: 2:47)



And, we know a heap tree is a complete binary tree and it, the values that the different nodes stores, it basically satisfies certain properties and there are two versions of heap tree, one is called the max heap and another is called min heap. Now, this figure shows one heap tree, it is called the max heap tree, a heap tree is called max heap tree, if each node satisfy the property that the value of that node is greater than the value of any node in any of its subtrees.

Now, if we see starting from this, this is the node, the value is 95. And if we see its subtree the value is greater than any value in the, in any of the subtrees actually. And another important property that the heap tree satisfies is that it is a complete binary tree that means before completing the final last level, it should complete its the last part one level. So, this is come, this is basically it contains the maximum number of nodes that it in this level it is possible before but this last level possibly maybe not fully complete.

So, this is a property that it should satisfy. And this property actually is an advantageous to store a heap tree not using link structure rather storing heap tree on an array and then indexing formula for from a parent node to its left child and right child that I have already discussed. And another advantage is that in case of ordinary binary tree you can traverse from parent to left but opposite way traversal is not possible. Whereas, if you store a tree using an array and using that simple indexing formula from a given children, you will be able to reach to its parents.

So, this is why this concept is very advantageous and array, it is the most memory efficient on data structure, because we have to store data only anything else we do not have to store, on the other end heap tree, you can manage the heap tree store any value, integer, string, any object, user defined data type, whatever it is there. So, this way, this is the one best one what is your data structure, which we can consider and that is why, the computer scientists gave enough, what is called effort to develop a better sorting algorithm.

And, another advantage of this sorting algorithm is that it is efficient compared to the all other sorting algorithm because the all sorting algorithm those are created as good sorting algorithm, usually having the time complexity order of  $n \log n$ . So, this sorting algorithm is also order of  $n \log n$  and that is to in all cases best case, average case or worst case whatever it is there. Now, let us start discussing about how the sorting algorithm works.

(Refer Slide Time: 6:18)

**Heap sort: Concept of sorting using heap tree**

- Create heap**
  - Create the initial heap tree with  $n$  elements stored in the array  $A$ .
- For  $i = n$  down to 1 do
  - Remove max**
    - Select the value in the root node (this is the maximum value in the heap). Swap the values (that is  $A[1]$ ) and value at the  $i$ -th location in  $A$ .
  - Rebuild heap**
    - Rebuild the heap tree for elements  $A[1, 2, \dots, i-1]$ .

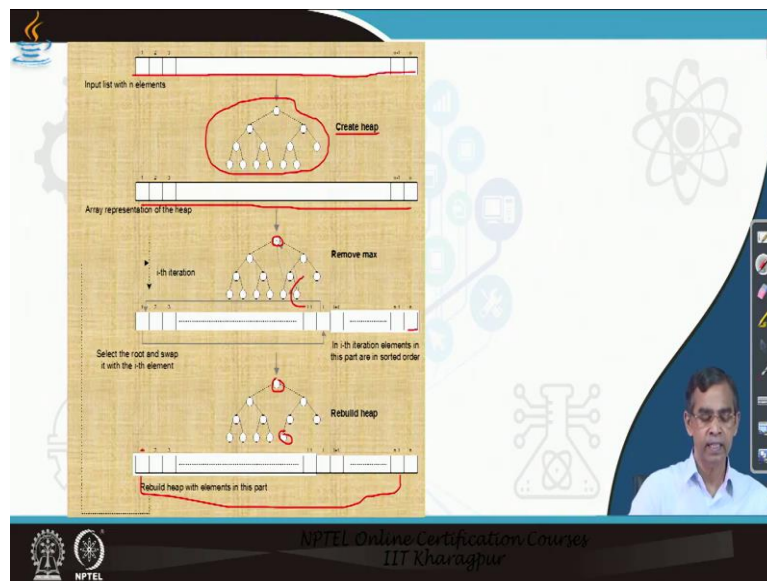
NPTEL Online Certification Courses  
IIT Kharagpur

And obviously, this requires the concept of heap tree which you have already discussed. I do not want to discuss the heap tree concept again. So, far the sorting is concerned your input, a set of numbers, let the number  $n$ ,  $n$  elements are given to you. So, what is the first step is that we have to first create a heap tree with  $n$  number of elements. Now, that creating heap tree we have already discussed, while we are discussing about heap binary tree. Then next part that is basically an iterative steps, we have to start from 1 to  $n$ . So,  $n$  number of loop, I mean loop should roll for  $n$  times, so  $i$  equals 1 to  $n$ .

Now, in each loop in each iteration, the procedure is that you have to delete the root node. Now, how we can delete the root node that also we have discussed while we are discussing heap tree. Anyway, so heap node, the root node is deleted from the heap tree and this node which is deleted from the heap tree is placed into the output buffer that is output list, then once that root node is deleted from the heap tree, the second step that is required in the, in each loop is that we have to rebuild the heap tree.

So, there are 3 major steps the first is that initially create the heap with the given elements, then remove maximum elements, if it is max heap or remove minimum elements if it is min heap which is a root node and then rebuild the heap. So, these are the 3 major steps those are involved. And you will check that list that means input list and output list we do not require two different arrays, in the same array where the heap originally stores will be able to store output list there also. So, this way it is also memory advantage that no memory over it is there. So, in case of memory constant application, this is really very good sorting technique.

(Refer Slide Time: 8:34)



Now, let us consider the idea that I have told you. So, this is the input list, list of  $n$  number suppose and we just create heap. So, heap is a complete binary tree satisfying the heap property that is either max heap or min heap whatever it is there. So, now this heap again, as you know it stored in the form of array that we have already discussed. So, our next step is that from this heap, we remove the root node. Now, how the root node will be removed, the last node should be swapped between the root node and this node.

Now, so, this last node in the last level actually, that is what I should say. Now actually swapping, this means that if you see in this array, the last node in the last level this element. So, swapping this means, the root node which is there will come to here. So, this way if you continue, and then once you swap this node that means delete the root node, then it may disturb the heap property. So that is why, what we require is that we have to rebuild the heap.

Now, the procedure will continue considering that this node is sorted, then again rebuild, so this is a rebuild then again the max node, this max node will be replaced by the location of the last node that means this, this and then this is the maximum will be swapped and this will continue and this will continue until the heap tree contains single node, whenever it contains single node, this is a only node and this place is already fixed.



(Refer Slide Time: 10:35)

```
1. Read  $n$  elements and stored in the array  $A$ .
2. CreateHeap( $A$ ) // Create the heap tree for the list of elements in  $A$ 
3. For  $i = n$  down to 2 do // Repeat  $n-1$  times
4.   RemoveMax( $B, i$ ) // Remove the element at the root and swap it with the  $i$ -th
5.   RebuildHeap( $B, i-1$ ) // Rebuild the heap with the elements  $B[1, 2, \dots, (i-1)], i-1$ 
6. EndFor
7. Stop
```

So, this is a procedure that we have to follow and for which the different, the algorithm is I have already given an algorithm and here is a algorithm, this algorithm is the heap sort algorithm. Now, as we see that there are two, I mean three procedure rather we can say create heap from the list of elements which is stored in an array. So, in argument is an array and if you pass this array to this procedure, it will build the heap tree that means, it will store the elements into an array following heap property.

So, this is the input array, where is the random numbers and create heap will return the same array, but elements are stored according to the heap structure, then the loop, the removed the root and then rebuild heap when we remove root, it store the element in the output list in the order, ascending order, descending order if you want to do it is in a min heap, then you can, you can store the elements in descending order that later on can be changed into ascending order whatever it is, there are little bit modification is that if we place a whatever it is there, so, we will be able to do that.

So, now we will give an example about programming that how mean heap can be used to do that, anyway. So, max heap is the convention, so that the elements should be automatically stored in an ascending order. So, this is the procedure. Now, we will discuss about this procedure that create heap remove root, remove node and then rebuild heap.



(Refer Slide Time: 12:20)

```
1 i = 1 // Initially, the heap tree (B) is empty and start with first element in A
2 While (i < n) do // Repeat for all elements in the array A
3   x = A[i] // Select the i-th element from the list A
4   B[i] = x // Add the element at the i-th place in the array B
5   j = i // j is the current location of the element in B
6   While (j > 1) do // Continue until the root is checked
7     If B[j] > B[j/2] then // It violates the heap (max) property
8       temp = B[j] // Swap the elements
9       B[j] = B[j/2]
10      B[j/2] = temp
11      j = j/2 // Go to the parent node
12   Else
13     j = j - 1 // Satisfy heap property, terminates this inner loop
14   EndIf
15 EndWhile
16 EndWhile // Select the next element from the input list
17 Stop
```

Let us discuss the different algorithm that we have given here. So, this is the algorithm that is from a given list of numbers, how we can create a heap tree I have given the algorithm if you understand the logic how it works, then understanding this algorithm is easy. And then once this algorithm is understandable, then you will be able to write the program with your own style. So, I always recommend that you follow the algorithm once your logic is clear, and then try to write the code of your own, so that you can understand it better.

(Refer Slide Time: 12:56)

1 2 3 4 5 6 7 8 9 10  
A 15 35 55 75 05 95 85 65 45 25

1 2 3 4 5 6 7 8 9 10  
B 95 85 25 65 50 05 15 45 55 35

Create heap: input array A to heap B

Continue...

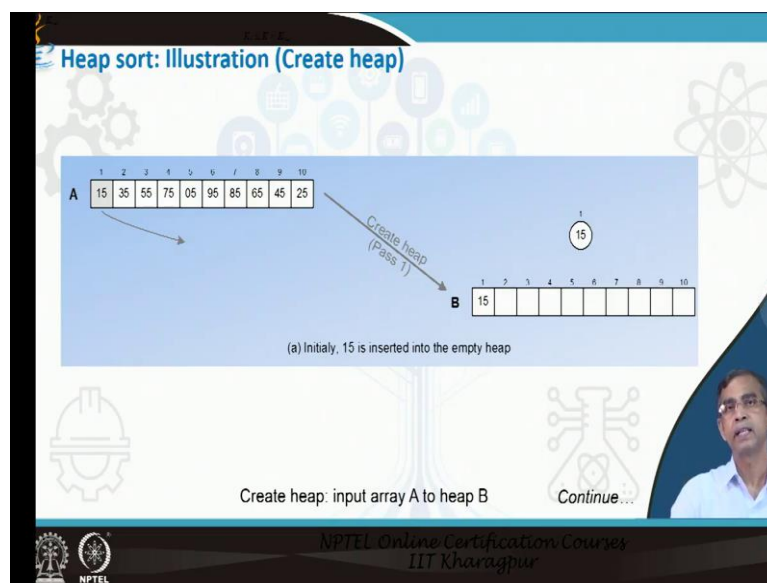
Now, so I should illustrate the example of how we can create a heap, that understanding is important. So, this is an input list. So, this is stored in an array, actually this is the array, the tree structure can be set like this. So, this is a logically looks like, is a tree but physically it is

stored in the form of an array. So, this is the input array initially, we should forget about this one not required. Now, from this array, we want to create heap. So, these store the same array, but the ordering of the elements you see in a different manner and this ordering is a logical structure of this heap tree.

So, root, then left child, then right child, left child, right child and each left child, right child according to this indexing formula  $2 \star i$ , if the root is an  $i$ , then its left child is  $2 \star i$ , its right child is  $2 \star i + 1$ , this is the formula and for a node, its parent is  $i$  by  $2 \left( \left( \right) \right)$  (14:03). So, this concept is basically used to, I mean move from a root node to, from a node to any of its child or from your child to its parent, whatever it is.

Anyway, so this is the idea about that heap. Now, let us have the few steps, so that you can understand how this heap is created initially starting from an empty heap tree.

(Refer Slide Time: 14:26)



So, this you can consider 1st, this is the input list, 1st we will consider 15. And initially heap was empty, so we just add these empty, we do not have to do anything. So initially, the 15 goes to in this array location. So, this is the only node and this is a root node, it does not have any child, or whatever it is there. So, this is a first step that the fifteenth is added. Next, we will see how 35 is inserted into heap. And so is, creating heap means take one element from the input list, insert the element into the heap whatever, this procedure will continue until the total, all elements are covered.

(Refer Slide Time: 15:08)

Heap sort: Illustration (Create heap)

A 1 2 3 4 5 6 7 8 9 10  
15 35 55 75 95 95 85 65 45 25

1 2 3  
55 15 35

(c) 55 is inserted and moved to root

Create heap: input array A to heap B Continue...

NPTEL Online Certification Courses  
IIT Kharagpur

Now, let us consider the insertion of other nodes. So, we have added 15. Now, let us consider adding 35, if we add here 35. So, 35 usually come here, but as they disturb the heap property, so, what you should do is that we have to swap. Now, let us come to the 55. So, 55 will be added here because it is in that portion. Now 55, if you add it, then it disturb the heap property then we have to swap, so if you swap then this one. So, swapping this and this gives you the heap with first 3 elements. Now, this procedure can be continued to further elements.

(Refer Slide Time: 15:56)

Heap sort: Illustration (Create heap)

A 1 2 3 4 5 6 7 8 9 10  
15 35 55 75 95 95 85 65 45 25

1 2 3  
75 55 35

15

(d) 75 is inserted and moved to root

Create heap: input array A to heap B Continue...

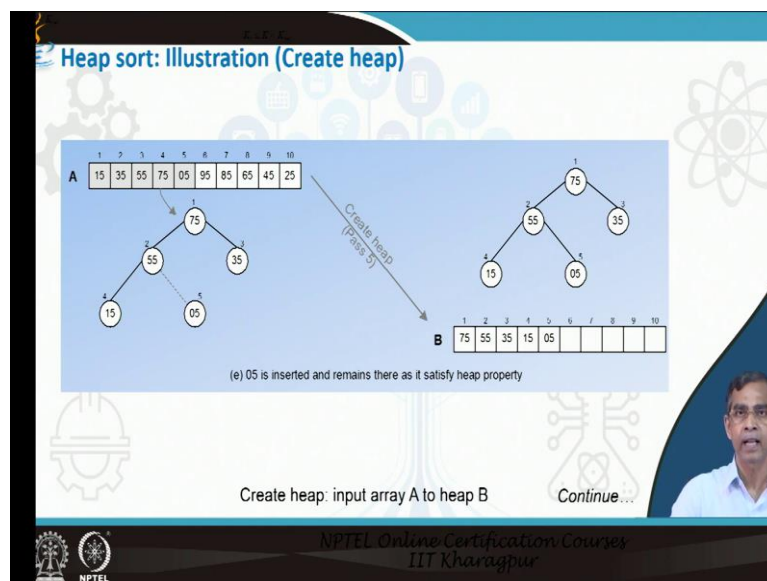
NPTEL Online Certification Courses  
IIT Kharagpur

Let us proceed on few more steps, so that we can understand it better. Now, let next element 75. So, 75 should come because this level is free, I mean full, so 75 will come here. Now, we should compare this node with its parent nodes, if we see that parent node is greater, then we

just go for swapping and this swap will continue till we reach to heap node, if require. For example, here 75 and 15 we see that they are not in order. So, it should be swapped 15 will come here 75 here, so initially 55 anyway, so this is going on and then 75 if we place here and then initially it was 55, so it says that, it also does not satisfy the heap property. So, 55 and 75 should be swap.

So, from the point of insertion, if we follow the path from that node to the root node, then only on that path only swapping if it is required till we see that the heap is not in order. So, this gives you the heap structure this one and you see this follows the indexing formula to store the elements into an array and this is the indexing according to that indexing the elements are stored in this array. So, this is the logical structure and this is physically the heap is stored in the array. So, this is, so if we continue this process again and again, I will not be able to discuss all the steps in details.

(Refer Slide Time: 17:30)



Now here again 5, as you can see 5 is already in order, so no need to do anything because it is root or its parent rather its parent is greater than the node itself, so nothing to do in this case.

(Refer Slide Time: 17:43)

**Heap sort: Illustration (Create heap)**

(f) 95 is inserted and moved to the root

Create heap: input array A to heap B      Continue...

NPTEL Online Certification Courses  
IIT Kharagpur

Next whenever 95 comes we have to again check and from this node point of insertion we have to go to the root and ultimately 95 will be placed at the root actually this is the largest node in this case.

(Refer Slide Time: 17:56)

**Heap sort: Illustration (Create heap)**

(g) 75 is inserted and moved to location 3

Create heap: input array A to heap B      Continue...

NPTEL Online Certification Courses  
IIT Kharagpur

### Heap sort: Illustration (Create heap)

(h) 65 is inserted and moved to location 2

Create heap: input array A to heap B      Continue...

NPTEL Online Certification Courses  
IIT Kharagpur

### Heap sort: Illustration (Create heap)

(i) 45 is inserted and remains there as it satisfy the heap property

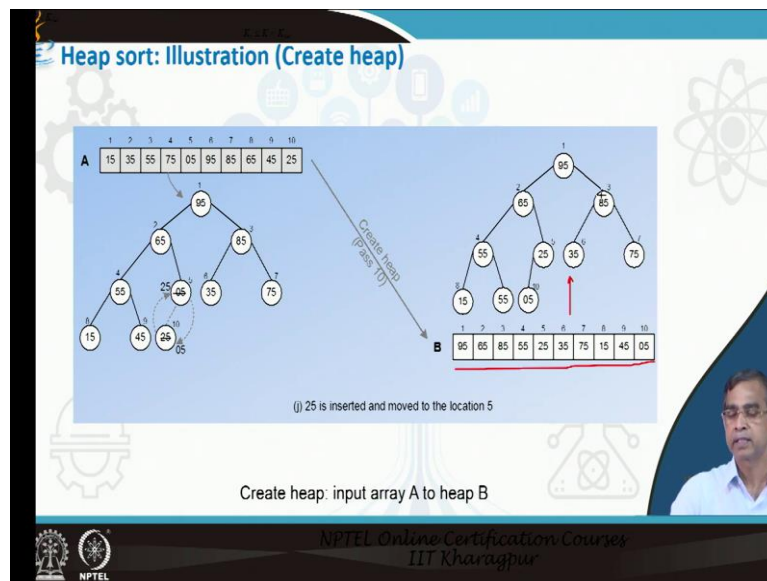
Create heap: input array A to heap B      Continue...

NPTEL Online Certification Courses  
IIT Kharagpur

Now, again 85 same procedure, next 65 we will go to the next level again the procedure is continued till we find that the elements are in order, that means its parent is greater than any of its child and this is the intermediate heap. And finally, 45 is added 45 is added after 45 added it placed. So, this is a heap structure.



(Refer Slide Time: 18:25)

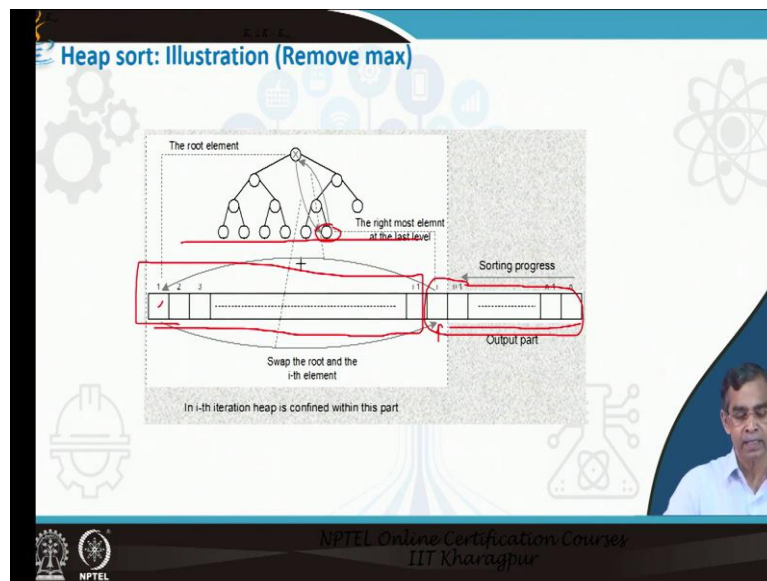


And finally 25, 25 we added and this gives you the final heap. This is the logically the heap look like and all the indexing formula I have given you for your understanding, so that he can understand. So, 1, 2, 3 this is its node, its left child, its right child, its node, left child, right child,  $2 \star i$ ,  $2 \star i + 1$ , it is  $i$ th position then left child is  $2 \star i$  position and it is 2 for any values appear.

Now here, one thing is you can note that the number of nodes which is required and size of the array that is required is almost same we do not require any extra space or there is no wasted or vacant space in the array, that is why heap is most compact I mean it stored the data in an array in a more compact way we can say, no wastage of memory space. So, this is about the, how we can create a heap.



(Refer Slide Time: 19:30)



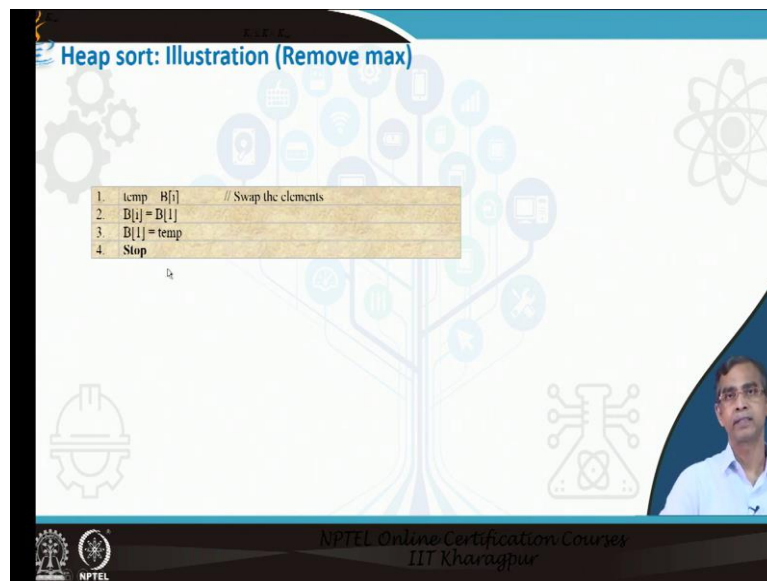
Now, our next procedure that is the next step is remove the root node, it is we can say remove max, it is a max heap, otherwise if it is a min heap, then remove min heap, anyway. So, remove max that means removing the root node. So, idea is that removing root node is swapping the root node with the last node in the last level. So, this is the last level and the last node. So, what we can do is that? We just swap this node this one. So, as node last level, if we, suppose up to this part we have already sorted, then these location of the last node in the last level and then we just swap this is the location of the root node.

So, swap the element this one and this we automatically see up to this part list becomes sorted. Then, for this part again we have to go for what is called rebuild the heap because whenever we place the last node here, it may not satisfy the heap property. So, that is why the heap rebuilding is required.

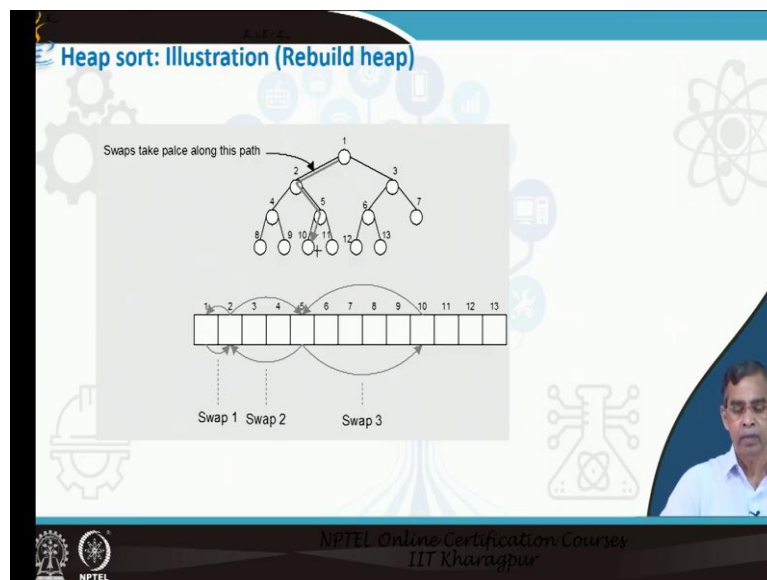
(Refer Slide Time: 20:41)

### Heap sort: Illustration (Remove max)

```
1. temp = B[i] // Swap the elements
2. B[i] = B[1]
3. B[1] = temp
4. Stop
```



### Heap sort: Illustration (Rebuild heap)



Now, let us consider how heap rebuilding is there now, this is a very simple swap procedure, just okay, we have to, just induction that procedure is nothing to be discussed there. Now, next is rebuild heap. Now, rebuild heap as I told you, so that from the position where you have inserted the node, it just change the check that whether the parent is greater than or not, if it is not greater than then swap the elements between the parent and that child and it will continue until we see that parent is greater or we can, we reach to the root node.

So, ultimately this process will lead to you to placing the next largest node at the root location. So, this way it tell how the swapping from current node to another node to go it is just, in the chain order, so it is easy.

(Refer Slide Time: 21:37)

```
1  If (i = 1) then
2  Exit // No rebuild with single element in the list
3  j = 1 // Else start with the root node
4  flag = TRUE // Rebuild is required
5  While (flag = TRUE) do
6  leftChild = 2*j, rightChild = 2*j+1
   /* Check if the right child is within the range of heap or not */
   /* Note: If right child is within the range then also left child
7  If (rightChild < i) then
8  /* Compare whether left or right child will move to up or not */
9  If (B[j] < B[leftChild] AND B[leftChild] < B[rightChild]) then
10 Swap(B[j], B[leftChild]) // Parent and left violate heap property
11 j = leftChild // Move down to node at the next level
12 Else
13 If (B[j] < B[rightChild] AND B[rightChild] < B[leftChild]) then
14 Swap(B[j], B[rightChild]) // Parent and right violate heap property
15 j = rightChild // Move down to node at the next level
16 Else // Heap property is not violated
17 flag = FALSE
18 EndIf
19 EndWhile
20 Else // Check if the left child is within the range of heap or not
```

Continue...

NPTEL Online Certification Courses  
IIT Kharagpur

And, the algorithm that you can follow for this process I have given it algorithm is relatively tricky, but if you follow it, you will be able to do it very quickly. 1st you should understand the logic, then you will be able to understand the algorithm that will be better and then finally, you can go for writing code. So, this is the algorithm that I have given you. So, you can take your own time to understand this algorithm.

(Refer Slide Time: 22:01)

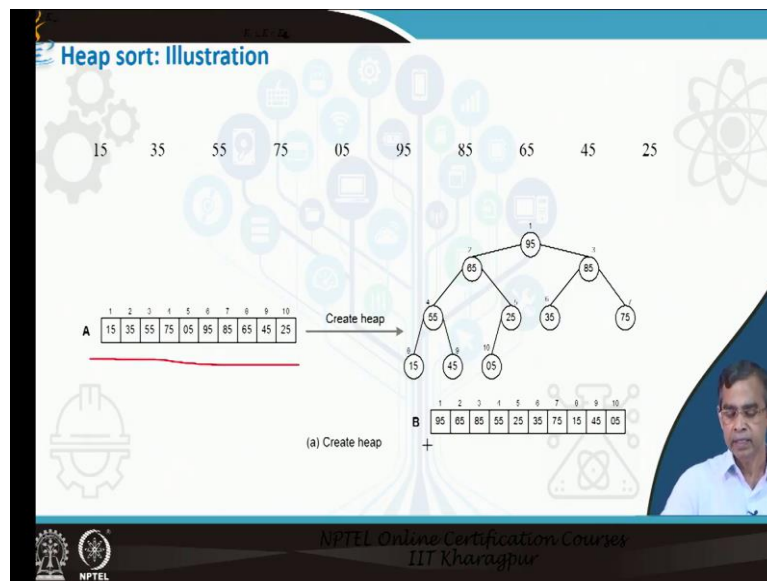
```
21 If (leftChild < i) then
22 If (B[j] < B[leftChild]) then // Parent and left violate heap property
23 Swap(B[j], B[leftChild]) // Swap parent and left child
24 j = leftChild // Move down to node at the next level
25 Else // Heap property is not violated
26 flag = FALSE
27 EndIf
28 EndWhile
29 EndIf
30 EndWhile
31 Stop
```

Continue...

NPTEL Online Certification Courses  
IIT Kharagpur

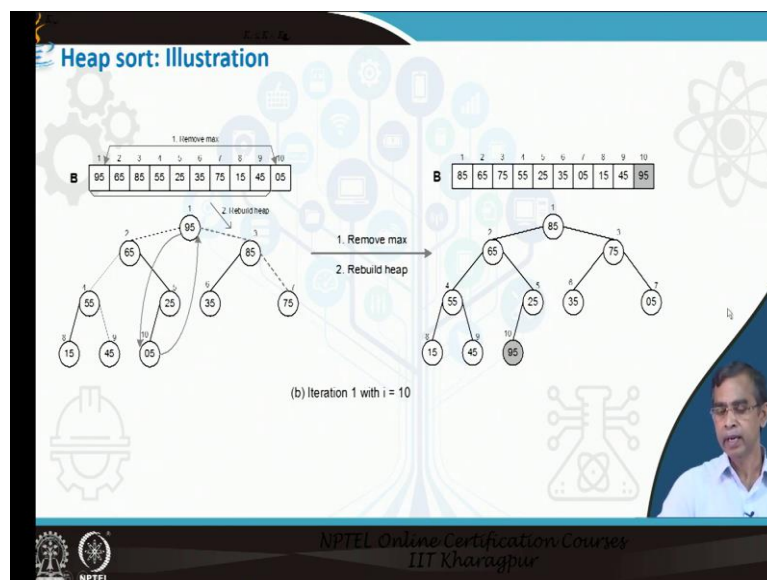
And when finally, I can illustrate the rebuild heap also, that procedure is simple.

(Refer Slide Time: 22:04)



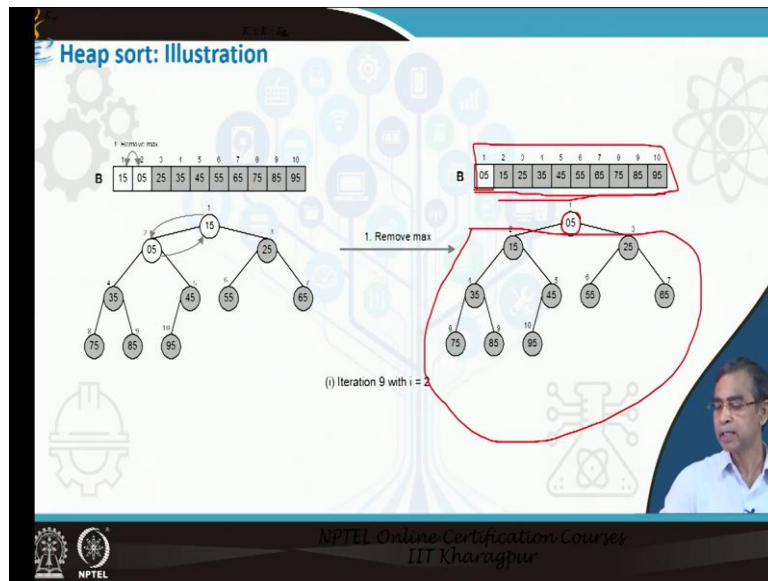
Here, I gave an example where it gives a full illustration that means heap sorting, starting from very beginning that we input list create heap, then remove and then we will repeat the procedures last two steps. Now, here let us consider this example again this is the input array, this is the create heap and after these create heap is there, we just remove the max.

(Refer Slide Time: 22:31)



Here remove max it comes to the location and it will continue then every time you see the nodes are getting sorted and then they are placing the array and if we repeat these things for continue.

(Refer Slide Time: 22:43)



Finally, all the nodes will be in a sorted fashion as you see here. And finally, the last node is only node because all the nodes have been covered, they are virtually deleted actually, and then this node is all placed in its location. So, finally, this gives you the output list which the elements in sorted order in ascending order, if you consider remove, if you consider the max heap. So, this procedure is the simple procedure that is there. Now, let us come to the discussion of the complexity of the sorting algorithm.

(Refer Slide Time: 23:26)

Case	Run time, T(n)	Complexity	Remark
Case 1	$T(n) = 3(n-1) + \frac{1}{2} \log_2(n+1)!$	$T(n) = O(n \log_2 n)$	Best case
Case 2	$T(n) = 4n + 2 \log_2(n+1) - 1$	$T(n) = O(n \log_2 n)$	Best case
Case 3	$T(n) = 3(n-1) + \frac{1}{2} \log_2(n+1)!$	$T(n) = O(n \log_2 n)$	Worst case

Handwritten notes on the right side of the slide:

$$\sum_{i=1}^n \log_2 i$$

$$= n \log_2 n$$

$$+ n \log_2 n$$

$$= 2n \log_2 n$$

Handwritten note at the bottom left:

$$O\left(\frac{n \log_2 n}{2}\right)$$

Now, complexity of the sorting algorithm as you see there are, first of all we have to create the heap. So, how much time it is required. So, first only 1 node, so only 1, second is 2, so only 2, third is 3. Now if you see in each time the number of comparison that is required in

order to make the heap is, in the  $i$ th to be placed in  $\log i$ . So, it is  $\log 1$  plus  $\log 2$  plus  $\log 3$  plus dot dot  $\log n$ , if  $n$  number of elements have to be placed there. So, the number of I mean the time complexity that is required to create a heap is we can say; is this formula that you can consider  $\log i$ , where  $i$  equals to 1 to  $n$  and simplification of this formula can give you  $n \log n$ . So, this is the time for creating heap in the first task.

Now each time if you remove that node from the heap and placing it, then again rebuilding this procedure, again if you see it will take and  $i$ th loop the  $\log i$  again because each time the  $i$  number of elements are stored. So this, so second step is also the same as  $n \log n$ . So, putting these two together, so total time that is required is  $2 n \log n$ , and so the time complexity that we can tell for the sorting algorithm is  $n \log n$ .

So, this is the time complexity and this is very good, I mean time complexity, so far the efficiency of algorithm is concerned, it is created as good sorting algorithm and whatever be the case, base case for example, the element is already given to you in sorted order. So, best case also whatever the step it is there we have to do it. Now, the average case also you can see that it is also same, average case it is and then worst case also it is like, worst case may be that you have to sort in ascending order by input is given in descending order whatever it is there. Average case means when the elements are given to you in random order. So, this is about the algorithm that is heap sort algorithm.

(Refer Slide Time: 26:16)





### Radix sort: Concept

- A sorting technique which is based on *radix* or *base* of constituent elements in keys is called radix sort. The radices and bases in few important number systems are listed in the table given below.

Number system	Radix	Base	Example
Binary	0 and 1	2	0100, 1101, 1111
Decimal	0,1,2,3,4,5,6,7,8,9	10	326, 12345, 1508
Alphabetic	a, b, ..., z A, B, ..., Z	26	Ananya, McGraw
Alphanumeric	A, ..., Z, A, ..., Z, and 0...9	36	06116014 05CSS201

NPTEL Online Certification Courses  
IIT Kharagpur

Now, let us come to the discussion of another sorting technique, the radix sort. Now, first we should understand what we mean about radix, so every number system has its own radix. For example, the simple most number system is called a binary number system, where only two elements are there that is called the 0 and 1 and it is binary. On the other hand, decimal number system, the number of digits, it is called, it is 10, that is why it is called decimal, so 0 to 9. This way the octal, the number of elements is 8, 0 to 8, 0 to 7 and is the octal system. The hexadecimal system 0 to 9 then a b c d e f, so it is a 16 symbols are used to represent that number according to that number system.

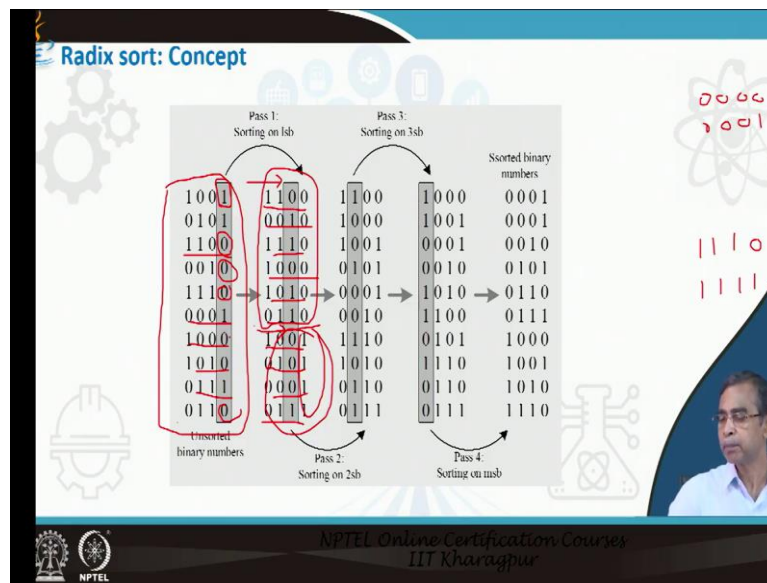
Now, if you can come to the screen, then there may be any alphabets is possible A to Z, either small letter or capital letter. So, we can say that it has the total 26 different symbols are used if we consider that case is not important. On the other hand, alphanumeric both stick both numeric symbols in decimal system as well as alphabets concerned together total a number of symbols that is equal to represent any elements any number or any elements we can say 36, so that these called, number of symbols and in (differ) different systems, the radix is there. So, radix may be 2 for binary system, 10 for decimal, 8 for octal, 16 for hexadecimal and like this one.

Now, so this concept is required. So, that we can write use this concept to I mean sort the numbers or any elements according to a particular system like. Now, the idea is that, so, position of each, for example, in case of decimal system, unit position, then the tenth level position, 100 level position, then 1000 level position these kinds of positions are there. So, it will go up to the different; what is the size of the element numbers that you want to consider, is a four-digit number or three-digit number whatever it is there.



Now, see the same is true for the binary number system also the first bit, it is called the least significant bit and then the highest bit is called the most significant bit. So, that bit is starting from LSB to MSB. Now, this actually the consideration is that if we take a simple unit from the system, and then we can consider whether the system will be placed higher. Now, they are actually the consideration is that we can plan according to the number system, so many brackets then each bracket will consider to store the elements according to the order of the position of the elements. Now, let us consider an example so that we can consider about this.

(Refer Slide Time: 29:38)

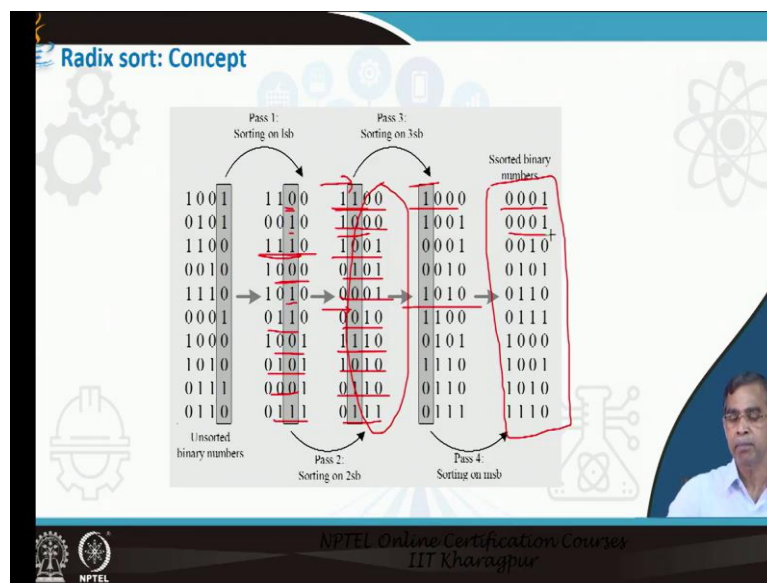


First we should consider a very simple numbering system, binary system that means, we want to sort the binary numbers. So, this is first this is consider, this is a list of input numbers and this is in random order. And binary number system, so the first number is the 0 0 0 this is and then the 0 0 0 1 and it will continue 1 1 0. And this is the ordering of the binary number system, so this is 0 and this is 16 or 15 you can say, this is 15, 0 to 15 actually, these are numbers, according to the decimal form, anyway.

Now, here you see first, we will consider only two brackets we can consider, but I just want to give an idea about how we can sort the numbers, and then they can be placed into their corresponding brackets, I can consider two brackets, like here I am considering only one bracket for an example. Now, what is the consideration is that, first of all, we will consider the least significant it will go to that bracket depending on each value, if 1 then it goes to the bracket 1 now in this binary system, only two bracket is required actually.

So, here we are sending this is the bracket start from here for 1 and this is a bracket start from here for 0, let us consider this concept. Now, so this is there, now this is of last this bit is 1. So, it goes to this bracket, so we place it here. Now, this is also 1 we go to the next, in the same bracket, here only. Next this is the 0, so it goes to the bracket this one, so it is coming here this is 0 it goes here again this is 0, so it goes there again, then again 1 this come here, and this 0 this goes, this come here, this 0, this come here, and this 1 again this come here, and this 0 it comes there. So, this is all numbers, if you see partially sorted according to that first bit actually and here also all elements are sorted according to the first bit.

(Refer Slide Time: 32:17)



Now, again, if we can repeat the procedure, but this time with respect to the second bits. What is idea? Idea is that now we will consider this position of the elements. Now 0, so we will start with 0 brackets here. And, we can say that 1 bracket start from here. So, we can actually take the two different locations for the two brackets, I am considering non location for example, and so this is a 0, so it goes here and this 1 it comes here, then this is 1 again, so these elements, so, these elements come here and this is 0 this means these elements come there as you see it is coming there.

So, this one it comes here and this one also it comes here. Now, these 0, 1 0 0 1 it will come here, this again 1 0 0 1 will come there, and this is also 0, 0 0 1 comes here and this is finally here, it comes here. So, now, we can say that whenever we can consider the second position of the bits, then all the elements are sorted according to this order. So, 0 0 0 0 0 1 they are in this order actually the binary fashion. Now, repeating the same procedure, then but this time,

we will consider this bit, then this number will go to the next bracket, this is the next bracket and this is the first bracket and 0 it will go to that this bracket as you see and this one.

So, ultimately the 4 different loops is required, if the total number of bits in the numbers is 4, then this is first 3 and after first 4, we see all the numbers are stored in this order actually. So, this is the smallest number, the next smallest number, the next smallest number and so on. So, this ultimately get the sorting order. Now, the same procedure can be repeated for the decimal number system.

(Refer Slide Time: 34:13)

### Radix sort: Algorithm

```

/* For each base in the number system */
1. For i=1 to c do // c denotes the most significant position
2.   For j=1 to n do // For all elements in the array A
3.     x = Extract(A[j], i) // Get the i-th component in the j-th element
4.     Enqueue(Qi, A[j])
5.   EndFor
6.   /* Combine all elements from all auxiliary arrays to A (assume A is empty) */
7.   For k = 0 to (b-1) do
8.     While Qk is not empty do
9.       y = Dequeue(Qk) // Dequeue of y from the queue Qk
10.      Insert(A, y) // Insert y into A
11.     EndWhile
12.   EndFor
13. Stop
  
```

NPTEL Online Certification Courses  
IIT Kharagpur

### Radix sort: Illustration

(a) Input list

A: 136 187 358 169 570 217 598 639 205 609

(b) Distribution of elements into 10 auxiliary arrays

Q<sub>0</sub>: 570  
 Q<sub>1</sub>:  
 Q<sub>2</sub>:  
 Q<sub>3</sub>:  
 Q<sub>4</sub>:  
 Q<sub>5</sub>: 205  
 Q<sub>6</sub>: 136  
 Q<sub>7</sub>: 487 247  
 Q<sub>8</sub>: 358 598  
 Q<sub>9</sub>: 169 639 609

(c) Combine all elements from auxiliary arrays to A

A: 570 205 136 487 247 358 598 469 639 609

(d) Distribution of elements into 10 auxiliary arrays in pass 2

Q<sub>0</sub>: 205 609  
 Q<sub>1</sub>:  
 Q<sub>2</sub>:  
 Q<sub>3</sub>: 136 639  
 Q<sub>4</sub>: 217  
 Q<sub>5</sub>: 358  
 Q<sub>6</sub>: 469  
 Q<sub>7</sub>: 570  
 Q<sub>8</sub>: 487  
 Q<sub>9</sub>: 598

Continue

NPTEL Online Certification Courses  
IIT Kharagpur

Let us have a look of this system. This is an algorithm that you can follow later on, I am emphasizing on the understanding about the logic algorithm that you can follow accordingly. Now, this is an example that you can consider these are input list as you see and the input list

is in random order. Now, first we consider the first position, so these are the position. Now, according to these position, the we can maintain yet nine brackets. So, this 136 goes to the sixth bracket as you can see here 487 should go to the seventh bracket, 58 goes to the eighth bracket and so on, so on.

Now, this height of the bracket should be considered in such a way that at the worst case, all numbers he wants to go to one bracket then he can accommodate, that means if all numbers have the same bit values, the same number, the same position values, then it is quite possible. For example, if you have given all the numbers who the last digit is on, then it should go to the bracket one like this one. So, this way, all the elements are sorted, you see.

Now, next what you should do is that, again we will consider all the numbers but considering the second position of the bracket, so then 570, so this will go to the fifth one. So, 570 it goes to the seventh brackets, and then 205 will go to the 0s position, then 136, it go to the third bracket, so here. Now, the second if you consider then it will ultimately, then another loop is required, because this is considered second, then third loop that if you consider, and here is a third loop.

(Refer Slide Time: 35:53)

**Radix sort: Illustration**

(i) Combining all elements from auxiliary arrays to A

A:	205	609	136	639	217	358	169	570	187	598
Q <sub>0</sub> :										
Q <sub>1</sub> :	136									
Q <sub>2</sub> :	205	247								
Q <sub>3</sub> :	358									
Q <sub>4</sub> :	169	187								
Q <sub>5</sub> :	570	598								
Q <sub>6</sub> :	609	639								
Q <sub>7</sub> :										
Q <sub>8</sub> :										

(ii) Distribution of elements into 10 auxiliary arrays in Pass 2

A:	136	205	217	358	169	187	570	598	609	639
----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

(iii) Combining all elements from auxiliary arrays to A

NPTEL Online Certification Courses  
IIT Kharagpur

We can see in the third loop, all the elements are right from the second pass already elements are there and if we take bracket wise. In this bracket, there is no element then this is the first element, the next element in this order, and then these elements and so the you see these are sorted order. So, this is the sorted list we can say if we collect all the elements, those are distributed among the different brackets.

So, this one algorithm based on the principle based on the key distribution where no key comparison is required. And we are not comparing key actually, we contrast to other algorithm where key needs to be compared. Now, so this is idea that, watch for the radix sort. Now, let us see how efficient the sorting algorithm is.

(Refer Slide Time: 36:40)

**Radix sort: Time complexity**

Let,  $a$  = time to extract a component from an element.  
 $e$  = time to enqueue an element in an array.  
 $d$  = time to dequeue an element from an array.

Time for distribution operation =  $(a+e)n$  [in Step 2-5]  
 Time for combination =  $(d+e)n$  [in Step 6-12]

Since these two operations are iterated  $c$  times (Step 1-12), the total time of computations is given by

$$T(n) = (a+e)n + (d+e)n \times c$$

$$= (a+d+2e)cn$$

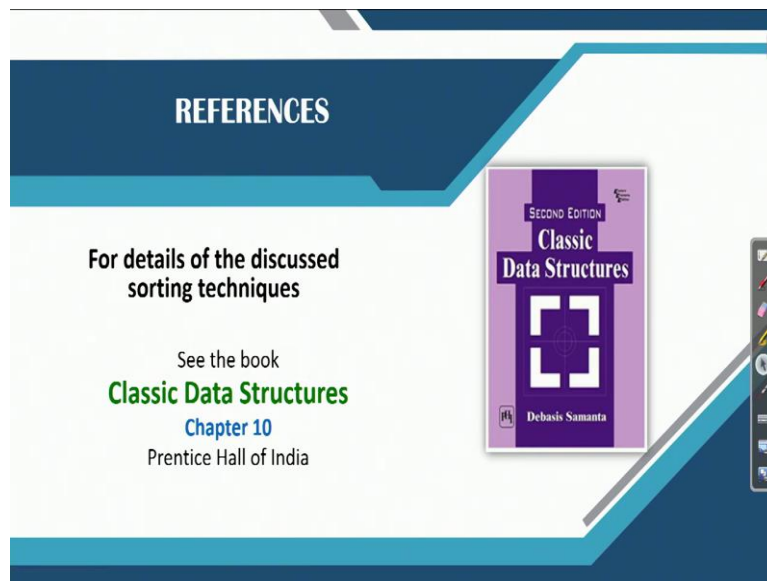
Since,  $a$ ,  $d$ ,  $e$ , and  $c$  all are constants for a number system, we have  $T(n) = O(n)$

*Handwritten notes in red:*  $= O(n) < O(n \log n) < O(n^2)$

NPTEL Online Certification Courses  
 IIT Kharagpur

And this algorithm is the details and analysis of the algorithm you can follow the book classic data structure, which I will refer it, so, the total time that is required to run the algorithm is this one, these are constant actually depending on the total number of bits, and then decimal system whatever it is there. So, this basically says that the complexity is order of  $n$ . Now, this complexity of order of  $n$  is very less or other is less than any sorting algorithm that we have discussed so far is order of  $n \log n$ , and order of  $n \log n$  is place is then order of  $n$  square. So, this is the sorting algorithm which possible with the lowest time complexity and that is the advantage that this algorithm is having.

(Refer Slide Time: 37:37)



Now, so these are the two algorithms that we have discussed about for the further study and few more discussion you, I suggest you to follow these books in the chapter 10, so that you can learn much more about in details according to your own level of understanding.

(Refer Slide Time: 37:50)



So, thank you very much. Thanks.