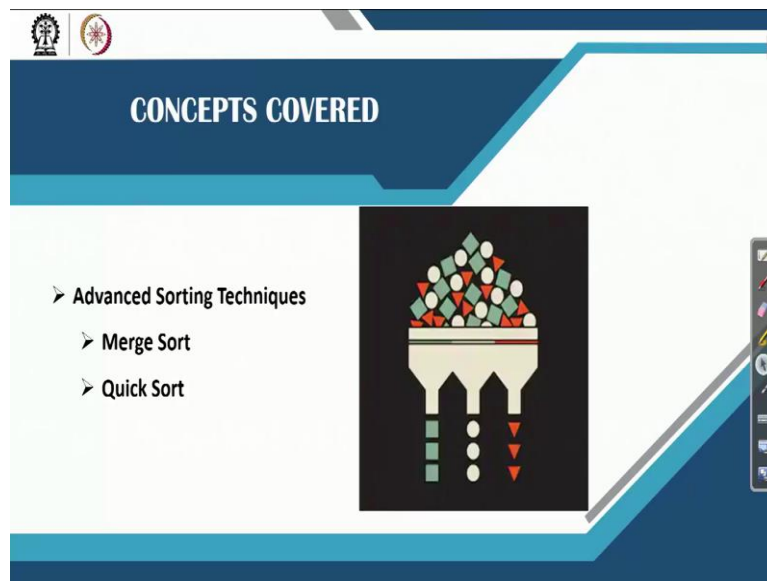


Data Structures and Algorithms using Java
Professor. Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 52
Advanced Sorting Algorithms

Let us discuss two sorting algorithms which people claims, the most advanced sorting algorithm, actually they are advanced sorting algorithm because not only they are performances based, but they are advanced because they have developed later on actually.

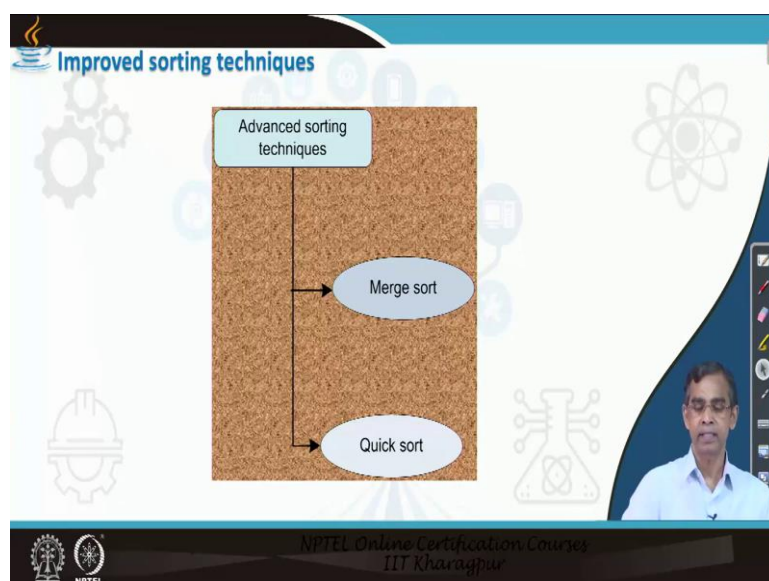
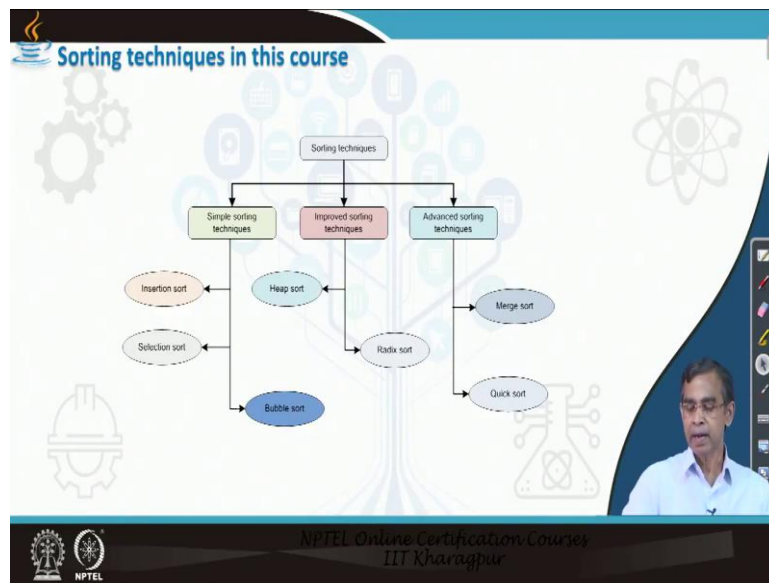
(Refer Slide Time: 0:42)



Now, these two sorting algorithm again have unique property that property is one paradigm actually you can say, based on the paradigm, it is called the divide and conquer. Now, we will learn about what exactly that property called divide and conquer, principle actually it is. And particularly these kind of sorting algorithm that we are going to discuss today it is the merge sort and then quick sort are very efficient sorting algorithm and suitable for sorting very large set of numbers.

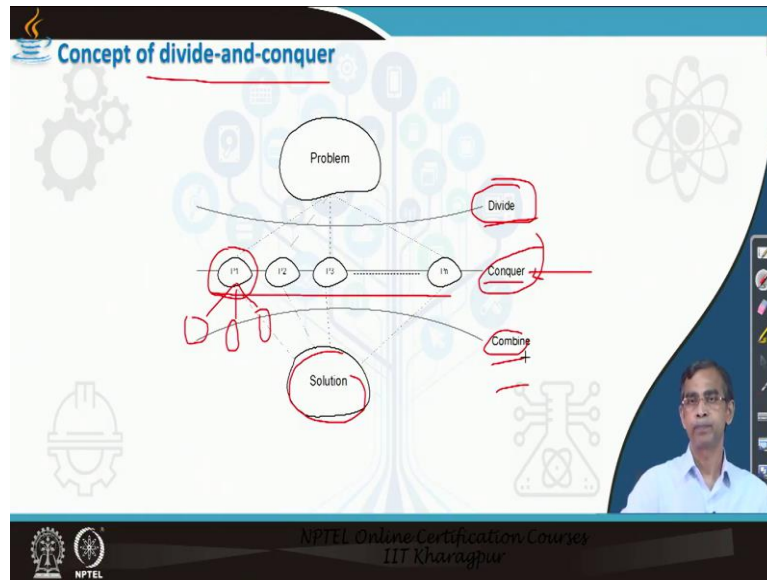
All other sorting algorithm that we have discussed for example, radix sort or heap sort or insertion sort, they are good, they are good, but they are usually mean for sorting a large, not large set of numbers rather very small or moderate set of number. On the other hand, these are the two-sorting algorithm that we are going to discuss are good for large set of elements to be sorted there.

(Refer Slide Time: 1:35)



Now, so these are merge sort and quick sort, we will discuss about. They comes under the banner advanced sorting techniques, as I have already mentioned about it.

(Refer Slide Time: 1:44)



Now they, these two sorting algorithm based on the principle, I said that divide and conquer. This concept comes here whenever you are not able to defeat your enemy I mean group, then what you can see that you can disperse them into smaller group like, then you can fight them and then you will be able to defeat, if you are not able to defeat as a whole. So, this kind of concept is called a divide and conquer. So, their idea is that if the problem size is very large, and it is not possible to solve in real time or it is taking too much time, then let us divide the problem into smaller parts.

Now, here I can tell one example, if your list is very large, then it will take very large amount of time to (solve) sort it then what you should do is that let us divide this list into smaller list and then sort each smaller list and then solve your problem this concept. So, this concept is called the divide and conquer concept. So, a problem needs to be divided into smaller parts as you can see here, this problem can be divided into smaller parts.

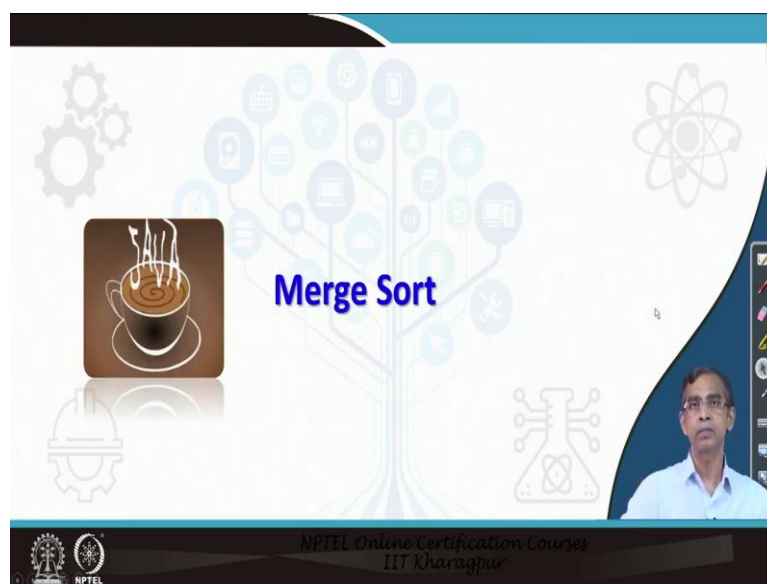
Now, if we see that this part again the smaller part is also very large one not able to solve in an efficient manner, then we can again divide into the smaller smaller, smaller so smaller, smaller parts and this kind of what is called a divide, principle divide can go on until you are comfortable with solving the smallest problem that you have at your hand.

And then finally, all the results that solving all smaller program or smallest problem whatever you can see and accumulating the result ultimately it will give the solution. So, this is a

concept called the divide and then, so divide and solving all small problem is called the conquer and then combining the result is called the combine. So, in that divide and conquer step or paradigm, there are 3 step, divide step, conquer step and combine step.

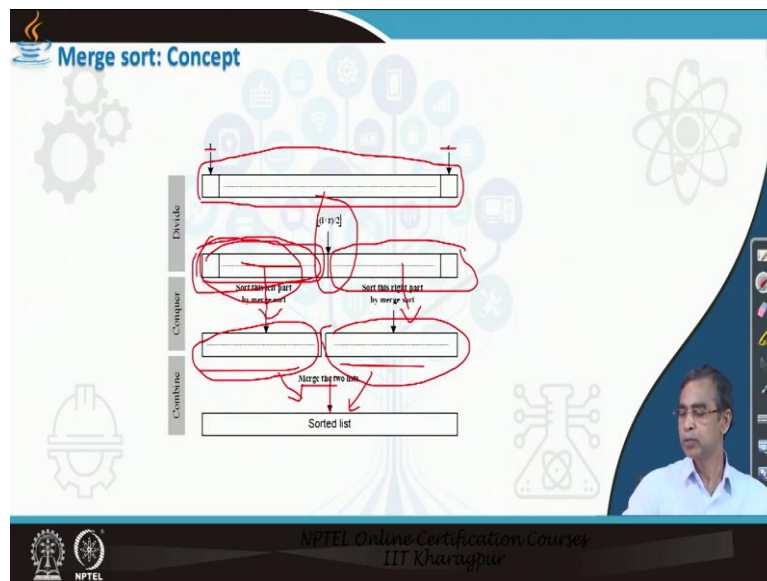
So, divide means you have partition does largest problem into smaller one, conquer means solving small problems, each small problem and then combining means getting the solution from each small problem and then finally, putting the result. So, this principle is followed in the divide and conquer and both merge sort and quick sort are based on these two in the these divide and conquer principle.

(Refer Slide Time: 4:12)



Now, let us consider first merge sort. What is the divide technique? What is the conquer concept and what is the combine that you have to study first and then you will be able to be able to understand how this this helps to sort a number, list of numbers.

(Refer Slide Time: 4:31)

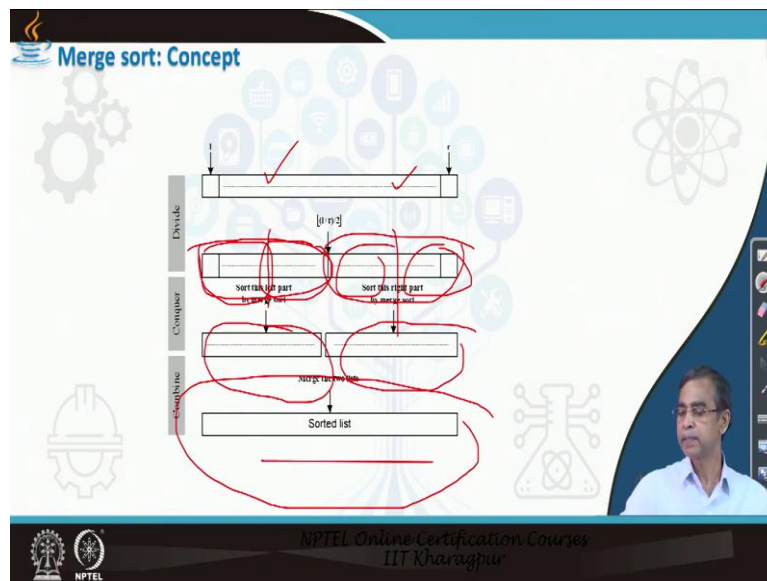


So here is the idea, idea is that let us consider this is the entire list and this is very large list suppose. And this L and R are the two bound, we can say this is the lower bound and largest bound, means you get the this is the first index and this is the last index we can say this way. Now these elements this list we can divide into two equal parts. So, this is the equal part, one part, the left most part and this is the right most part we can say, or left list and right list, then what you do is that, this we can sort and this we can sort and then after sorting, we can then merge the two lists.

So, this list, which is in random order, it is a sorted fashion, this is also random order this gives a sorted one. So, this is a sorted list, sorted list of containing the element the left part and this is one. Now, you cannot match them together because here this is sorted here this is sorted, but when we join them, they may not give the sorted list. So, how do we actually do is that, we have to combine the result into a sorted list. So, this combination process in this merge sort is called a merging operation.

So, here actually here is a divide operation and here you can see sorting each sub list is conquer operation and then combining the result is called a merging operation, these are the three steps are there.

(Refer Slide Time: 6:11)



Now, here again, if you see that this list itself is very large list, then what we should do is that, we can repeat the solving this problem in the same manner, the way we have solved this problem. Similarly, if this is also very large on solving this part is in the same manner the way you have done it. So, this means that initially the problem of size n , now we divide the problem into size n by 2, next time we can again divide the problem into size n by 4 each, n by 4 each and so on.

And in each time whenever n by 4, n by 4 again we merging, merging means it will give you the sorted list of size n by 2, here is the sorted list of size n by 2 and finally, the merging two sorted lists into the final sorted list of size n . So, this concept is followed here and this is the idea about the merge sort.

(Refer Slide Time: 6:59)

Merge sort: Concept

- **Divide**
Partition the list midway, that is, at $\left\lfloor \frac{l+r}{2} \right\rfloor$ into two sub lists with $\frac{n}{2}$ elements in each if n is even or $\left\lfloor \frac{n}{2} \right\rfloor$ and $\left\lfloor \frac{n}{2} \right\rfloor - 1$ elements if n is odd.
- **Conquer**
Sort the two lists **recursively** using the **merge sort**.
- **Combine**
Merge the sorted sub lists to obtain the sorted output.

NPTEL Online Certification Courses
IIT Kharagpur

Merge sort: Operation merge

(a) Input lists A and B and the output list C before the merge operation

A	2	4	6	8	10	12											
B	3	5	7	9	11	13	15	17									
C																	

(b) After the first movement from B to C:

A	2	4	6	8	10	12											
B	3	5	7	9	11	13	15	17									
C	1	2															

NPTEL Online Certification Courses
IIT Kharagpur

Now here, the time that is required to merge. First let us discuss about the process, what is the merging first, how we can merge the two sorted list into a larger sorted list, actually I will discuss about merging, then I will be able to discuss about other, divide is here very simple TBL state because we just simply find the middle location and then decide the left list and then right list. This is the divide, divide is really very simple and conquer is basically we can sort the sub list.

Now, sorting the sub list, you can call any other sorting technique maybe say heap sort, but actually the idea is that why you should solve this one using heap rather we can call the merge sort again. So, it is, the conquer is, repetitive step or is a recursive step. And then

finally, the merging. Merging is the conquer, combining. So, this is the concept it is there. Now, let us first discuss about how we can merge the two sorted lists into a larger sorted list.

So, let us consider as an input, this is a one sorted list and this is another sorted list. So, we can start from the starting points and so, this is the i th pointer for the list A and this is the j th pointer is the list, initially they both point to the first elements in the list. Now, out of these two elements, what we can do is that we will compare and in the output list originally initially at empty, and then we will see exactly who will go to this output list.

Now, it will be decided the smallest number should go to the output list, then I, here 1, it is a smallest number. So, 1 will come here, then once it is moved, so this is empty, so then pointer will move this one and then it will point to the 3 and then this 2, now next again 2 and 3, we have to compare, then who will go then 2 will go, so 2 will be here and then this pointer will come here, then 4 and 3, then 3 will come here, then this pointer will move to this one.

So, this way, once the element is moved to the output list, the pointer automatically, from which list it comes, that pointer will move towards the right and finally, whenever this pointer or this pointer, if say suppose this list is exhausted and this pointer is here, then all the elements at the end should be placed here. So, if this is the list of size n_1 and this is the list of size n_2 and then this is a list of size n this is equals to n_1 plus n_2 . So, this is the merge list, we can say. So finally, this will give the merging operation, and this merging operation is simple.

So, all together merging operation that will take, if the two list are of equal size n and n and then the merge list is of size $2n$ then these complexities order of n , then the n number of comparison that is required. So, this is why this is a very simple algorithm.

(Refer Slide Time: 10:08)

The slide illustrates the merge operation in three stages:

- Initial state:** Array A contains [4, 6, 8, 10] and Array B contains [3, 5, 7, 9, 11, 13, 15, 17]. Array C is empty. Pointers i and j point to the first elements of A and B respectively. Pointer k points to the first position in C.
- Step (c):** The second element from A (6) is moved to C. Array C now contains [6].
- Step (d):** All elements in A are moved to C, and pointer i reaches the end. Array C now contains [6, 8, 10, 12, 14, 16, 18, 20].
- Step (e):** Trailing elements in B are moved to C and finishes the merge operation. The final sorted array C is [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

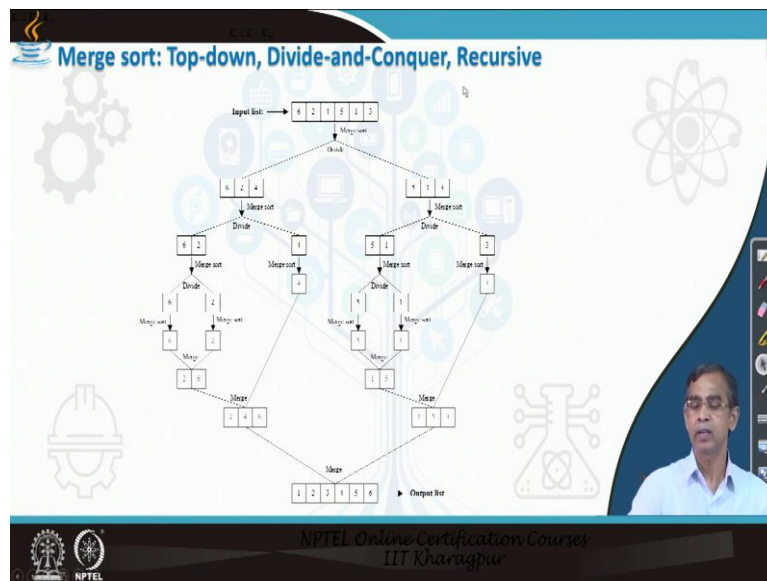
And here we have discussed whatever I told you ultimately it will give you the merge list. So, this is a merge operation and this is the most what is called the costly operation we can say in this type actually and all other operation are very simple.

(Refer Slide Time: 10:23)

```
1 i = 1, j = 1, k = 1 // Three pointers, initially point to first locations
2 While (i <= n1 and j <= n2) do
3   If (A[i] < B[j]) then
4     C[k] = A[i]
5     i = i + 1, k = k + 1
6   Else // A[i] > B[j]
7     C[k] = B[j]
8     j = j + 1, k = k + 1
9   EndIf
10 EndWhile
11 If (i > n1) then // A is fully covered
12   For p = j to n2, do // Move the rest of the elements in B to C
13     C[k] = B[p]
14     p = p + 1, k = k + 1
15   EndFor
16 Else // B is fully covered
17   If (j > n2) then+
18     For p = i to n1, do // Move the rest of the elements in A to C
19       C[k] = A[p]
20       p = p + 1, k = k + 1
21     EndFor
22   EndIf
23 EndIf
24 Stop
```

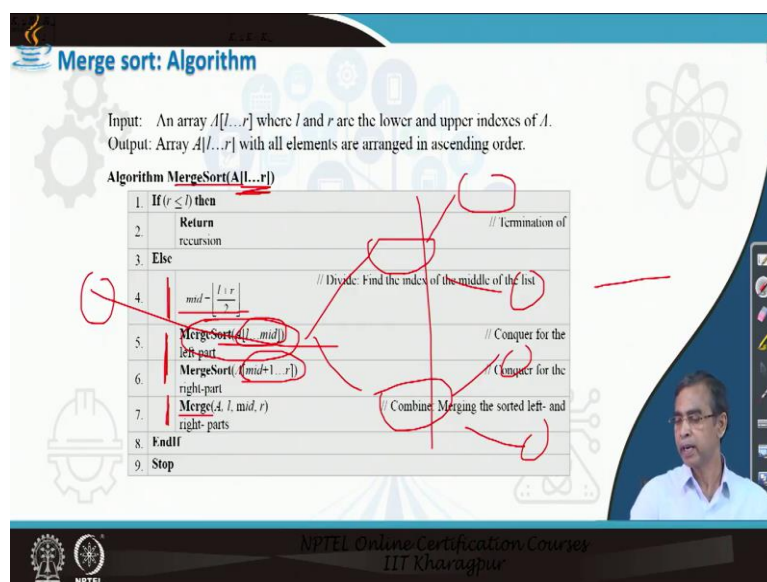
And here, I have given the algorithm how we can write, how we can perform the margin operation. So, once your logic is understandable, then going to this algorithm is very easy. So, margin operation, it is a merging operation is algorithm for the merge operation here.

(Refer Slide Time: 10:46)



And then so, the totally this concept is called top down divide and conquer and recursive procedure that algorithm, quick sort also same, it is a top down divide and conquer recursive, top down means, initially start with a large list then, then go down down down so, it will be called top to down and then divide and conquer each time we divide and conquer, each time we divide and conquer, because each time we divide, solve and conquer, divide, solve and conquer like.

(Refer Slide Time: 11:17)

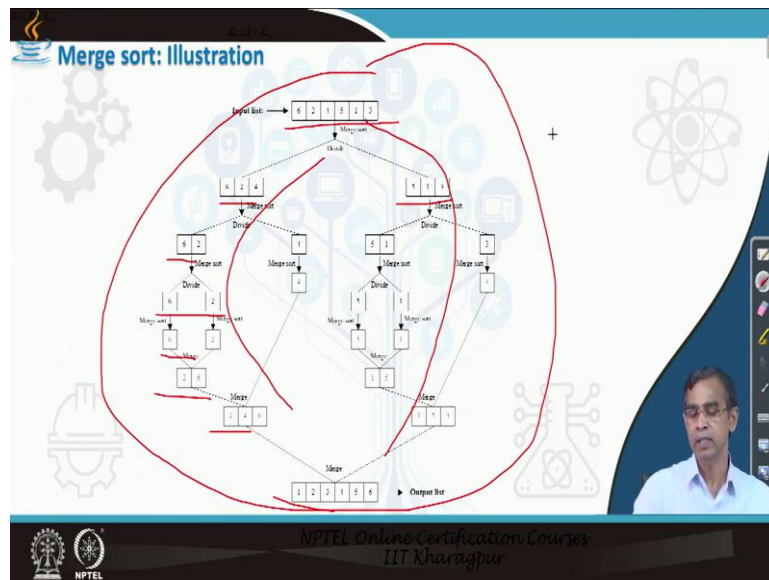


And finally, it is called recursive because it called the same procedure again and again, and that is why this algorithm can be termed like this. So, this is the algorithm merge sort, the first is find a middle location because we have to divide the list and then again call the same

procedure, I mean merge sort, but for the smaller list that is a sub list we can say left part of the list and this is also right part of the list and then we call the merge. So, here this is the divide operation we can see and this is to conquers because we are solving and this is the combine.

So, this is a divide, this is algorithm actually, and you see we just go from top to down this is starting from the whole list, then again divide and then whenever we call it, again this again do the same procedure as it is like this again divide and then again divide it also divide and this is why it is called the top down. So, initially top is this one and then go down down and finally it solved the problem. So, that is why, if we draw the recursion tree that is why recursion tree it is look like this actually.

(Refer Slide Time: 12:36)

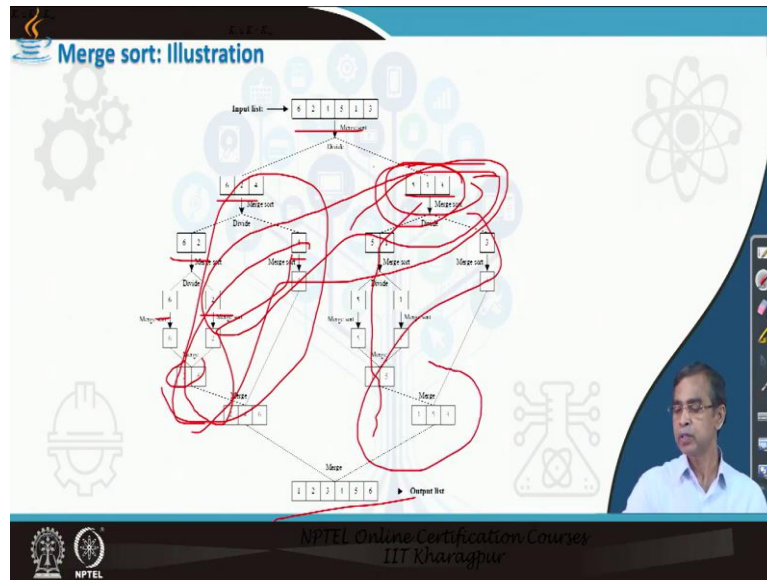


What I told you, that recursion trees look like this, as you can understand from here, this is here, these are initial list. So, sub list divide then again sub list then this complete because we cannot stop because it is a communism condition only the list contain one element, then merging, this is the merging and this one, and finally this one. So, this is a left part similarly, if you can follow the right part it is there. So, this whole thing is called the recurse, recursive tree actually the way it occurs there.

Now, this is for these elements and for the other element, list also you can draw the recursion tree actually, but ultimately, we do not have to do the recursion tree or it is not required it is just made for explanation. But whenever actually the competition takes place, the everything,

taken care like and as it is a recursive procedure implementation. So, each time it follows a stack to store the result.

(Refer Slide Time: 13:32)



For example, this is a initial list then this one, so, stack will be called to push this list to store there, because it will try to solve their whenever solving their stack we will try to call this one then here also stack will try it. So, these elements are putted into the stack as you see here. And whenever, the finish, that means this is a termination condition, then stack will be, so this stack will be open.

So, it is last in first out order, actually and then go there. So that, so this one then again 4 and then this put there, and then, so this put there and finally this put there and it come here, this time also whenever we solve this problem again stack stack stack and then putting there so it will come here. So, this way the procedure works there and this, and this internal stack is required and recursion implementation is the task of the runtime manager who will take care about allocating which call will go to the stack and then which sub procedure will be invoked from the stack all these things the runtime manager will take care, only you have to write the program and writing the program is that algorithm implementation that algorithm actually.

(Refer Slide Time: 14:48)

Merge sort: Algorithm

Input: An array $A[l..r]$ where l and r are the lower and upper indexes of A .
Output: Array $A[l..r]$ with all elements are arranged in ascending order.

Algorithm MergeSort($A[l..r]$)

1	If ($r \leq l$) then	
2	Return	// Termination of recursion
3	Else	
4	$mid = \lfloor \frac{l+r}{2} \rfloor$	// Divide: Find the index of the middle of the list
5	MergeSort($A[l..mid]$)	// Conquer for the left-part
6	MergeSort($A[mid+1..r]$)	// Conquer for the right-part
7	Merge(A , mid , r)	// Combine: Merging the sorted left- and right- parts
8	EndIf	
9	Stop	

The slide includes a video inset of a speaker in the bottom right corner and the NPTEL logo in the bottom left corner. The text 'NPTEL Online Certification Courses IIT Kharagpur' is visible at the bottom.

Writing the program means you have to implement this algorithm, now implementation with this algorithm means we have to define this procedure, this is a recursive one, and we have to define this one, so merge procedure needs to be defined only, and then just write the main code here. And we will give a programming demonstration, once we discuss everything then you will be able to understand, so how it works actually.

So, anyway, so this idea is very simple, actually, this is the algorithm that you have to follow and this merge algorithm that you have to write the code, it is enough for you to solve the problem. So, it is, so simple writing the code is really very simple, if you are little bit efficient or I mean skill about how to write the program in a recursive manner. So, all quick sort, merge sort is a recursive sorting algorithm.

(Refer Slide Time: 15:36)

Merge sort: Time complexity

$$T(n) = c, \quad \text{if } n = 1$$
$$T(n) = D(n) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + C(n), \quad \text{if } n > 1$$
$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{2}\right) = T\left(\frac{n}{2}\right)$$
$$T(n) = c + 2T\left(\frac{n}{2}\right)$$
$$=$$
$$= O(n \log n)$$

NPTEL Online Certification Courses
IIT Kharagpur

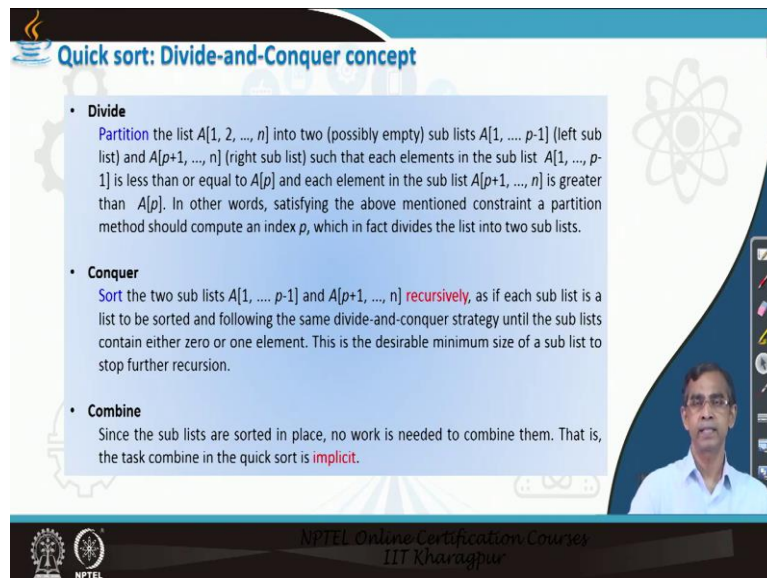
Now, so, this is the sorting algorithm. The time complexity of the sorting algorithm I can tell this way, if T_n this algorithm is given details in a very simple way, I can tell about if T_n is the time complexity to sort using merging operation, I mean merge sort is T_n for the number of elements n , the first is that we have to first divide, divide takes only very small amount of time, let it to c . Then, we have to divide the 2 list, so 2 list and solving the 2 list again in a recursive manner. So, it is $2n$ by 2.

So, this if we continue this recursive explanation, if you write, solve the recurrence relation actually, then ultimately you can check that its complexity is in order on $n \log n$. So, that detailed discussion of the calculation you can find in any standard book including the Classic Data Structure, there also, the details about this steps it is discussed. So, this is the procedure about the merge sort.

(Refer Slide Time: 16:52)



The slide features a central graphic of a tree with various icons (gears, a smartphone, a laptop, a mail icon, a document, a lightbulb, a magnifying glass, a person, a gear, a network, a server, a cloud, a mail, a document, a lightbulb, a magnifying glass, a person, a gear, a network, a server, a cloud) on its branches. To the left is an icon of a coffee cup with steam. The text 'Quick Sort' is prominently displayed in the center. The slide is part of an NPTEL Online Certification Course from IIT Kharagpur, as indicated by the logos and text at the bottom.



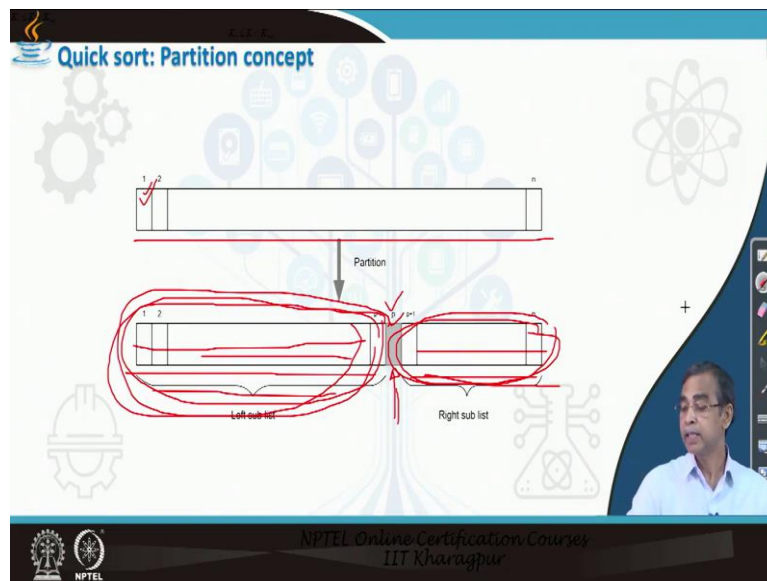
The slide is titled 'Quick sort: Divide-and-Conquer concept'. It contains three bullet points:

- **Divide**
Partition the list $A[1, 2, \dots, n]$ into two (possibly empty) sub lists $A[1, \dots, p-1]$ (left sub list) and $A[p+1, \dots, n]$ (right sub list) such that each element in the sub list $A[1, \dots, p-1]$ is less than or equal to $A[p]$ and each element in the sub list $A[p+1, \dots, n]$ is greater than $A[p]$. In other words, satisfying the above mentioned constraint a partition method should compute an index p , which in fact divides the list into two sub lists.
- **Conquer**
Sort the two sub lists $A[1, \dots, p-1]$ and $A[p+1, \dots, n]$ **recursively**, as if each sub list is a list to be sorted and following the same divide-and-conquer strategy until the sub lists contain either zero or one element. This is the desirable minimum size of a sub list to stop further recursion.
- **Combine**
Since the sub lists are sorted in place, no work is needed to combine them. That is, the task combine in the quick sort is **implicit**.

The slide is part of an NPTEL Online Certification Course from IIT Kharagpur, as indicated by the logos and text at the bottom.

Now, let us discuss about the quick sort. Quick sort again based on the same principle that is divide and conquer, but here divide technique is different, conquer is different and then combine is different. Now, but again like merge sort, it is also top down divide and conquer and recursive, that means recursive written is there. Now, let us discuss in this algorithm, which is the divide step, which is the conquer step and which is finally the combined step.

(Refer Slide Time: 17:24)



In this divide and conquer procedure, the divide step is called partition. In case of merge sort, that divide is simply finding the mid location. So, there divide is very simple, but here divide is more crucial, more critical. On the other hand, in case merge sort, the conquer is recursive, but the combining is there costly because merging operation and here you will see the combining is very fast. So, these are the I mean, the comparison between the two-sorting technique.

Now, in case of merge sort, let us first discuss about this divide and conquer then we will come back to this comparing the two-sorting technique. So, here divide concept is called the partition, the partition concept is like this. So, suppose this is the, this is the input list that is given to you and you have to take any one element, let the first element and this element is called a pivot element. Now, what we have to do is that, we say pivot element as this is a random number, it can be any value.

Now, pivot elements should be placed in its final position, what is the meaning of final position? The meaning of the final position is that it should be placed in this list in such a way, for example, it is placed in here that means, this is a, this element is placed but in such a way that all elements in this part of the list is less than this value pivot and all elements in this list is greater than this pivot element.

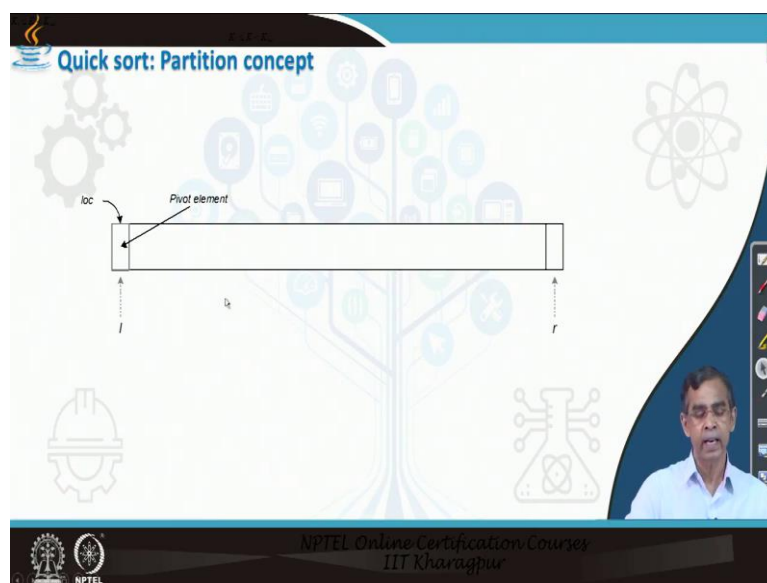
Now, these actually see consequences that meaning of this is that this element placed in final location because it does not, it is not required to move anymore because all elements here they are smaller than this one, all elements here, greater than this one. So, this is already

placed this one. Now, here partition this is called partition. Partition means we have to select anyone number say the first number and this is the pivot number, place it in the list in such a way that it will divide two parts of the list. Now, this this part is called the left list and this is called the right list.

Now, here data not necessarily in sorted order, here also data not necessary in sorted order. Then, what you have to do is that we can again sort this and this sorting again can be called using the quick sort for this but here (n) , but here will be around n by 2 . So, this part also n by 2 , so there is divide and conquer is solving the same, solving the smaller problem using the same procedure these are conquer and finally here there is no combine because you are automatically default combining it because they are already placed here.

So, combine is nothing here. So, here only expensive task or operation is partition. Now, let us see, how this partition operation can be done.

(Refer Slide Time: 20:32)



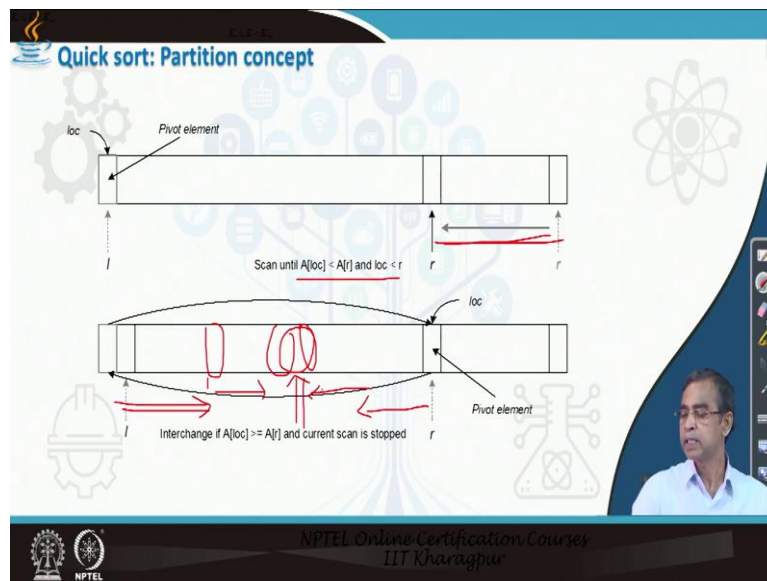
There are many algorithms available in the book here and there, so partition a can be followed following any one book I am telling you one idea about. So, the how the partitioning can be done. So, idea is that this is suppose l and r are the two extreme location and this is the part of the list, whatever it is the list in initial list, where we have to place that pivot element into its final position.

Now, from here, we have to look at there and we can swap this element this element if we see that this element is greater than this element, because always it is, this size should be greater than this pivot element, then once it is there, then pivot element will come here, then the

element will go there, then again from there we will check it and then we will follow this direction, but whenever we move it, we see that whether all elements in this part is less than the pivot element here or not.

So, whenever we are here and looking here, we always see that the greater element and whenever it is here and looking from here, we see that elements are less than they are not. So, this procedure will continue and will stop here whenever we see that both these l and r moves into the same position.

(Refer Slide Time: 21:45)



Now, let us explain this algorithm this one example here. So, suppose at any moment the pivot comes here and this is there and we just move into this way so that it we can right move this pointer, if we find that this element is less than this part actually, then because this should be, if you place p, then it should be less than this element that is (())(22:13) logic. So, if we see that, if this pivot element and we can move this pointer.

If we see that this element is less than that means, in other words, we can say, we can move or swap this element to this element, if we see that any elements is greater than this pivot element that concept is there because swapping will takes place or this moment will stop or halt whenever we see that the element which is pivot element is greater than this element, then we can halt otherwise, if it is less than then we have to move this one.

So, finally, it is there, now pivot come here and this element here, then again we repeat the same procedure, but in this direction pointer will move here, the l will move in this direction, again whenever we see the pivot will check that whether this pivot element is greater than the

element which is understanding. So, if we say that it is greater than then we will adjust, if this pivot element is greater than then we just simply move it and as soon as we see that pivot element is less than the element which here then, we can just halt this movement and stop it and then pivot will come here.

So, this way pivot will just swing from here to there and again and again pivot is there, again we can start from here and this same procedure will continue, let the pivot come here then pivot again then the next will come here and then let pivot is there. Now, this way, so left and right actually moving towards each other, when you see the left and right, they come in the same place, then we can say that pivot is already placed in this position and we stop it here. So, this is a procedure that we can think about.

(Refer Slide Time: 24:09)

Quick sort: Algorithm partition

```
Algorithm Partition(left, right)
1. loc = left // The left most element is chosen as the pivot element
2. While (left < right) do // Repeat until the entire list is scanned
3.   While(A[loc] < A[right]) and (loc < right) do // Scan from right to left
4.     right = right - 1 // No interchange. Move from left to right
5.   EndWhile
6.   If (A[loc] > A[right]) then // Interchange the pivot and the element at right
7.     Swap(A[loc], A[right]) // The pivot is not at the location right
8.     loc = right // Next scan (left to right) begins from this location
9.     left = left + 1
10.  EndIf
11. While(A[loc] > A[left]) and (loc > left) do // Scan from left to right
12.   left = left + 1 // No interchange. Move from right to left
13. EndWhile
14. If (A[loc] < A[left]) then // Interchange the pivot and the element at left
15.   Swap(A[loc], A[left]) // The pivot is not at the location left
16.   loc = left // Next scan (right to left) begins from this location
17.   right = right - 1
18. EndIf
19. EndWhile
20. Return (loc)
21. Stop
```

Continue

NPTEL Online Certification Courses
IIT Kharagpur

And, that procedure can be explained with few more steps and that algorithm that logic that I have told you, so this algorithm is implemented here. So, this algorithm you can take your own time to go through and then understand about it. And so, this is the procedure called a partition and which is the most important step it is there.

(Refer Slide Time: 24:27)

As, an illustration, I can take some time from you and then you can discuss about it. So, this is a pivot element and this is the right most element. And, as you see that pivot element is greater than 33. So, what we should do? That we should swap it, so pivot is coming here and 33 goes there. So, this swapping and then we have to again repeat it, but from there because 33 is considered we should not consider next element we have to then again we have to check that if it is less than then we just simply move it right, if 55 is right, because these sides the all elements should be less than this 55.

Now, 88 is greater, so there again swapping is required because 55, that means that element should be the less than then only we will be able to move it as if it is greater, then we have to stop, I mean, we have to halt the moment. So, again either the swapping is required. So, 55 will come here and then 80, so here the 88 will come there. Now, this procedure may be in the next slide, we can explain it. So, that we can understand about.

(Refer Slide Time: 25:43)

Quick sort: Illustration partition

scan from left to right

33	88	22	99	44	11	66	77	55
----	----	----	----	----	----	----	----	----

loc

(b) After the scanning from right to left while the pivot is at left. Next, it begins the scanning from left to right

scan from right to left

33	55	22	99	44	11	66	77	88
----	----	----	----	----	----	----	----	----

loc

(c) After the scanning from left to right while the pivot is at right. Next, it begins the scanning from right to left

Continue

NPTEL Online Certification Courses
IIT Kharagpur

Here is a next slide, here you can see. So, 88 comes here and 55 place here, now 55 here again, we can start from here. Now, again, we see that 77 is greater. So again swapping is required. So, 55 will come here and 77 is come there, we just took move it whenever we see that it is greater than we can move it. So, now 55, I can just clean this part better. Now, let us start it again.

(Refer Slide Time: 26:17)

Quick sort: Illustration partition

scan from left to right

33	88	22	99	44	11	66	77	55
----	----	----	----	----	----	----	----	----

loc

(b) After the scanning from right to left while the pivot is at left. Next, it begins the scanning from left to right

scan from right to left

33	55	22	99	44	11	66	77	88
----	----	----	----	----	----	----	----	----

loc

(c) After the scanning from left to right while the pivot is at right. Next, it begins the scanning from right to left

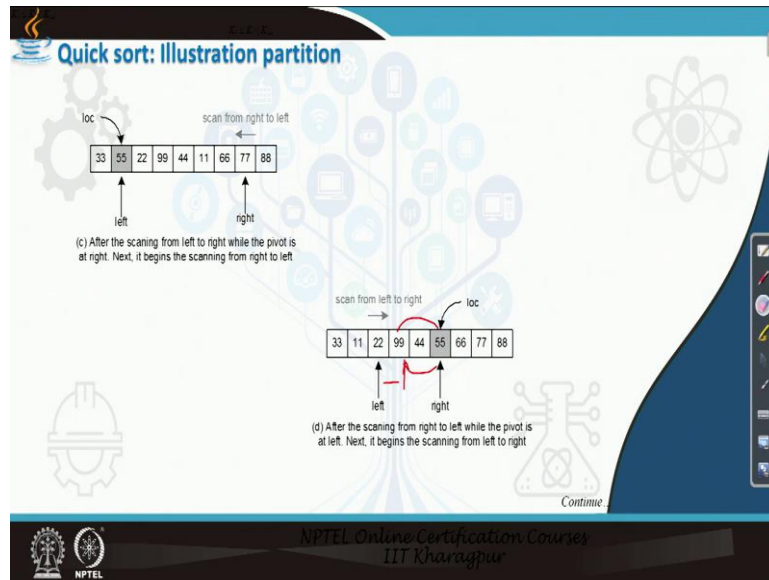
Continue

NPTEL Online Certification Courses
IIT Kharagpur

So, pivot is here we have to start from here. Now, we see that 55 is less than 77. So, we can move here. Now, this 67, 66 also less than, so we move here. So, finally we come here, when we see that 55 is greater than 11. So, this require shifting, so what shifting is that 11 will come here and 55 will go there. So, this moment will come there then actually here 55 will

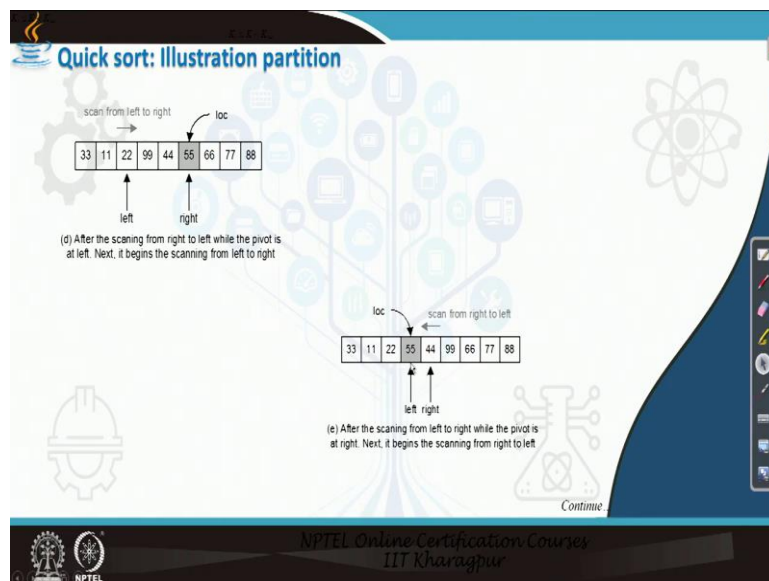
come here and 11 will come here. Now, this possibly, let us see in the next slide that we can think about.

(Refer Slide Time: 26:59)



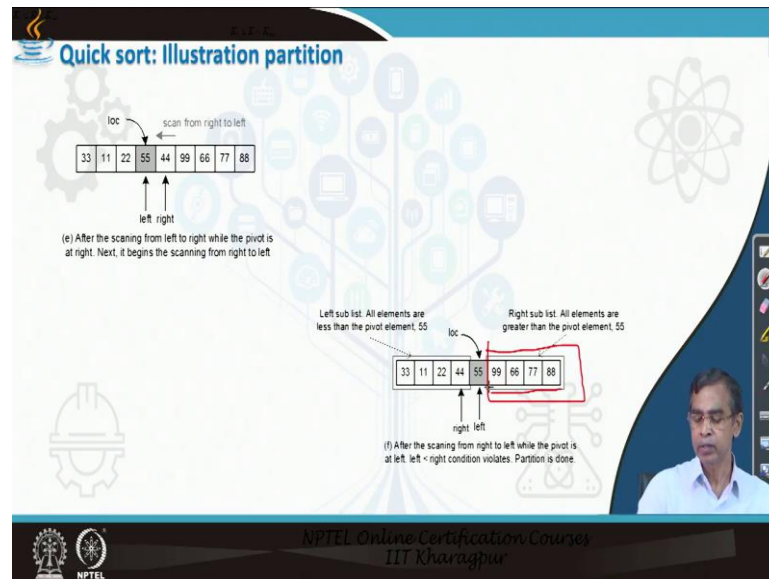
In the next slide we can see, in the next slide as you see 11 goes there and 55 come here, now we can, start, we can start checking from here, and it will continue if we see that this element is less than this element, so, it is called this one because 22 is less than, so pointer will come here and whenever we come here 99, as you see that it is greater than. So, there needs to be swapped.

(Refer Slide Time: 27:32)



So, this part again explain into the next slide, as the next slide as you see here 55 and 22. So, they can be removed here. Now here, 55 and 44 again we can check it, this is, this one and this and then we can say that this is less than, so again swapping is required.

(Refer Slide Time: 27:45)



So, this swapping is come here and then left and right you see with the swapping they can come to the same place. Now, at this position whenever this left and right come to the same place we see that this element always, that this list contains all elements, which is greater than this element, and this list contain all element which is less than this pivot element, this means that that this pivot element 55 each placed in its original, this is its final position and these are the right sub list and these are left sub list, we can write and then we can call the same quick sort method on the sub list again, the same procedure, but it is the number of list is reduced to half actually. So, this is the procedure that we can follow. And so, this is the partition, partition is the main task here.

(Refer Slide Time: 28:45)

Quick sort: Divide-and-Conquer concept

- **Divide**
Partition the list $A[1, 2, \dots, n]$ into two (possibly empty) sub lists $A[1, \dots, p-1]$ (left sub list) and $A[p+1, \dots, n]$ (right sub list) such that each elements in the sub list $A[1, \dots, p-1]$ is less than or equal to $A[p]$ and each element in the sub list $A[p+1, \dots, n]$ is greater than $A[p]$. In other words, satisfying the above mentioned constraint a partition method should compute an index p , which in fact divides the list into two sub lists.
- **Conquer**
Sort the two sub lists $A[1, \dots, p-1]$ and $A[p+1, \dots, n]$ **recursively**, as if each sub list is a list to be sorted and following the same divide-and-conquer strategy until the sub lists contain either zero or one element. This is the desirable minimum size of a sub list to stop further recursion.
- **Combine**
Since the sub lists are sorted in place, no work is needed to combine them. That is, the task combine in the quick sort is **implicit**.

NPTEL Online Certification Courses
IIT Kharagpur

And that is a divide procedure we can say and then the conquer is sorting the sub list and combining it implicit, we do not have to do anything here.

(Refer Slide Time: 28:56)

Quick sort: Algorithm

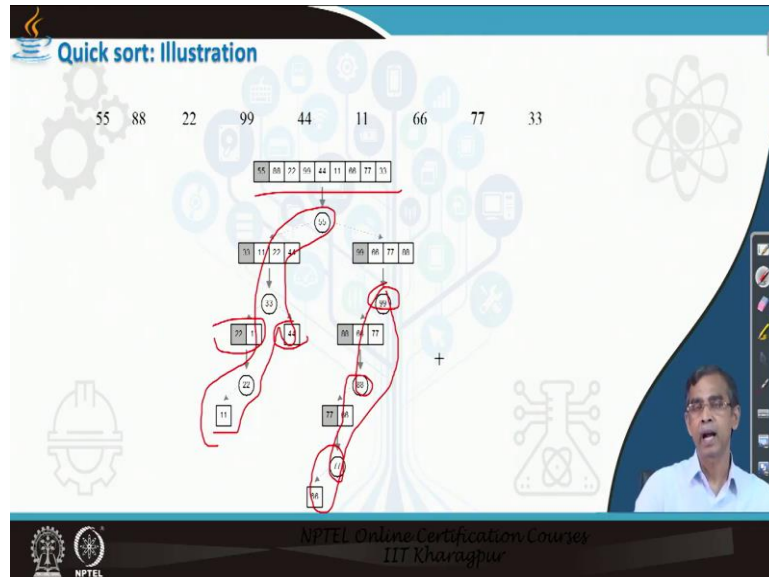
```
1. Loc = Partition(left, right) // left and right are two pointers to locate partitions
   // at left and right, respectively
2. If (left < right) then // Check for the termination condition
3.   QuickSort(A, left, loc-1) // Perform quick sort over the left sub list
4.   QuickSort(A, loc+1, right) // Perform quick sort over the right sub-list
5. EndIf
6. Stop
```

NPTEL Online Certification Courses
IIT Kharagpur

And this is a procedure, algorithm that we have given the total quick sort algorithm, 1st we have to partition given the left and right this is the two boundaries of the list then again, we have to call the quick sort, this is a termination condition recursively we can call this is for the left part, left list and this is right list and that is the quick sort algorithm. So now, so this is recursive. So, this is the quicksort itself we call the same procedure again and again. And here is a partition algorithm that algorithm I have already discussed above that is only heavy algorithm.

Otherwise, everything is very simple. And while we discuss the programming that time we will be able to understand that how nicely it can be programmed only to write the code for right partition and then the squeaks out in a recursive manner.

(Refer Slide Time: 29:43)



So, this is one example that we have used for your illustration, these are total list, so pivot is 55 and then go on recursively. So, this is a recursive tree actually, how it go as is 55, 55 partition the list. So, this is the input list 55 is a pivot it placed, then again sort it, 33 is the pivot, 33 placed left and right. Now, 22 is a pivot, this is empty, so stop this is only 1 stop stop.

So, this already sorted, so 11, 22, 33, 44 and this way 55, you see all these elements are already placed in their order this part and this part also if you see starting from here, so 99 is placed, then 88 placed, then 60, 77 is placed, then 66, so this is also placed. So, this order, so whenever you do it, they are automatically the final position gives place them, so we do not have to do any other procedure for the combine it goes this way. So, this is a procedure that you can think about the quick sort algorithm.

(Refer Slide Time: 31:02)

Quick sort: Time complexity

$$T(n) = (n-1) + 2 \sum_{i=1}^{n-1} T(i) \quad \text{for } n \geq 1$$

with $T(1) = 0$

$$T(n) = 2(n+1)(\log_2 n + 0.577) - 4n$$
$$T(n) = (n-1) + 2T\left(\frac{n}{2}\right)$$
$$= O(n \log n)$$

NPTEL Online Certification Courses
IIT Kharagpur

And, so far, the time complexity is concerned is very similar to the merge sort. So, only the partition time that is required for this actually the time complexity that it is required, if you say the, for the n number of elements partition. So, it is a for the partition is required around n time, it is n minus 1, we can say n minus 1 comparison is required for the partitioning, I mean to place the pivot into its final position plus next is 2 parts. So, it is this, is the partition algorithm and this 2, if you say the 2 list of equally sized, set half then this is the complexity of the algorithm we can say.

So, it is written in this way it is like this one and the details discussion of this complexity analysis here we can say it is around this one. So, it, this complexity analysis it can come to you same as order of $n \log_2 n$, now detail discussion and then the procedure about this complexity analysis, you can find in any book or including classic datasets that you can consider. So, this is the complexity. Now, this algorithm is more or less I mean same complexity than the merge sort.

(Refer Slide Time: 32:29)

Quick sort: Time complexity (Worst case)

$$T(n) = (n-1) + T(n-1) \quad \text{for } n \geq 1$$

with $T(1) = T(0) = 0$

$$T(n) = \frac{n(n-1)}{2}$$
$$= (n-1) + (n-2) + \dots + 2 + 1$$
$$= \frac{n(n-1)}{2} = O(n^2)$$

NPTEL Online Certification Courses
IIT Kharagpur

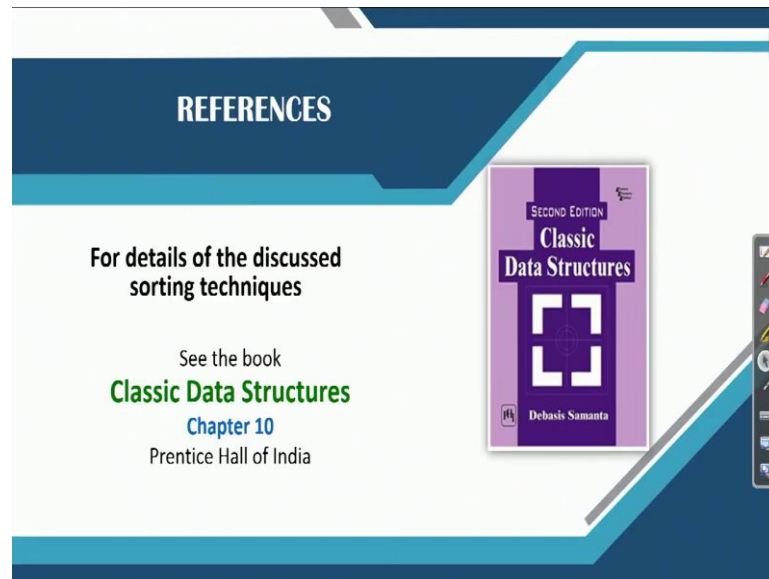
But there are certain comparisons that we have to think about it. Now, this comparison is important when we, there are best case, worst case, average case also. Let us first discuss about the best case, worst case, average and this is the worst case, the worst case occurs when we have to sort a list which is already sorted order, already in sorted order. So, in that case the n complexity that can be expressed in this formula, this is because this is to place that pivot element n minus 1 comparison then it leave only 1 list, the size of the list is again in minus 1.

Now, this part if you go then it is n minus 1, this is there, and if, it is then it is single n minus 2 plus these T n minus 2. Now, if we factorize it, so all these things will come, so n minus 3 plus T n minus 4. Now, this way if you continue, so finally, this will give you n minus 1 plus n minus 2 plus continue 3 plus 2 plus 1 because this is the final termination condition. Now, this the total time that is required you can check it n into n minus 1 divided by 2. So, this complexity is in the order of n square.

Now, so, in case of worst case, time that is equal for the quick sort it takes order of n square time this is comparable to merge sort actually in case of merge sort it is order of $n \log n$. So, that is why the quick sort is termed as, it is nothing but quick when the list is already in sorted order. So, definitely quick sort works worst when the input list is in sorted manner, but it is actually, so that depends on.

So, in most of the cases usually we have to sort the list with not in order or in any order, so, that case is rare rather. So, this way it will not be a big issue for that, but quicksort is really that is why it is called the quick compared to the other sorting algorithm.

(Refer Slide Time: 35:11)



And in this book, you can find the comparison of different sorting algorithms and a graph we have drawn, then you can check that if the number of elements increases, then how their performance also changes. So, the train, you can understand, then you can decide that which algorithm works better for you. Now, so, we have discussed many sorting techniques, merge sort, quick sort, these are the advance and then there are some good sorting algorithm heap sort and the radix sort and finally, we have discussed many simple sorting algorithm like bubble sort, insertion sort and selection sort.

Only few sorting algorithms have been discussed in this course, but there are many more sorting algorithms are there which are also they have their own, good point pross as well as some limitations in each them. So, those things you can learn if you want to learn much more about many sorting techniques, that books Classic Data Structure you can consult, you can find many ideas from their details algorithm, illustration, the comparison, analysis, everything in details you can find from this book.

So, with these things I you would like to stop it here today. Next portion that we will discuss about programming aspects. So, what are the different sorting techniques that we have discussed, how to write program, I will give hint about, how to write the program. But my

advice is that you should learn the logic, algorithm if required you can follow and then write the code of your own that matters to improve your programming skill.

So, this is the one good point that sorting and searching algorithm usually most programmer take up their cases to on their programming skills. So, you will get this opportunity to improve your programming capability if you want to implement all the sorting algorithms of your own. So, thank you very much.