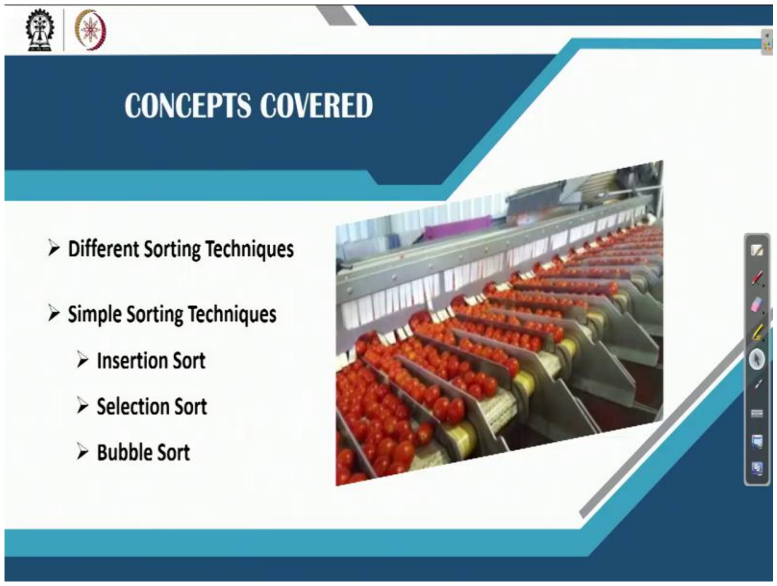
Data Structures and Algorithms using JAVA
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 50
Sorting Algorithms (Part-I)

Hello, so, in many applications, one task is highly frequent and the tasks are searching and sorting. We have already covered searching algorithms and how to write programs dealing with searching mechanism. Now, we will discuss another important task that is called sorting. Now sorting as a very fundamental algorithms, but it is being exercised like anything and many sorting techniques are now known, which they, they have their own advantages as well as disadvantages, but it is better that if we have a little bit review of different sorting techniques are those are available.

(Refer Slide Time: 1:15)


CONCEPTS COVERED

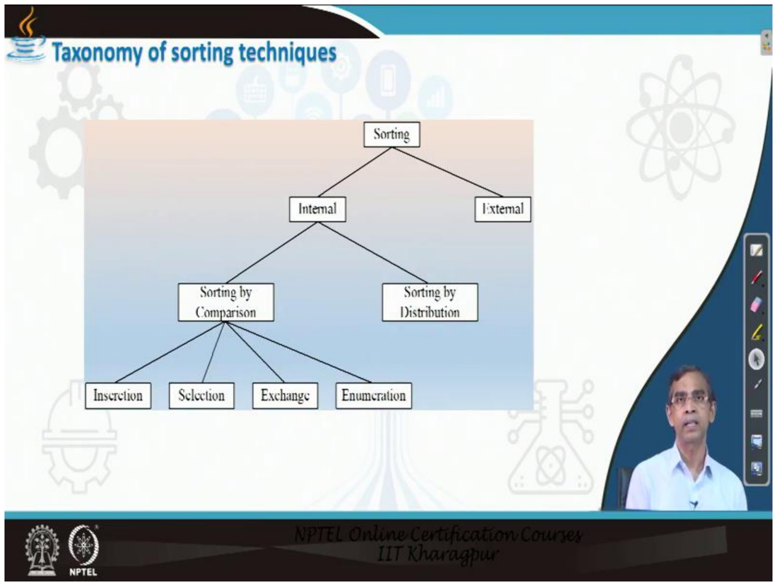
- Different Sorting Techniques
- Simple Sorting Techniques
  - Insertion Sort
  - Selection Sort
  - Bubble Sort

So, in these lectures, this is first part of the, of this module and there will be few more parts; three more parts there, we can cover other parts covering other techniques. So, in this module in this lecture, we will try to first cover what are the different sorting techniques are now known in the literature. And then we will try to learn about very few simple sorting techniques. And in this category, there are three techniques are known as simple sorting techniques, which is called insertion sort, selection sort and bubble sort. So, this is our plan for today.

(Refer Slide Time: 1:52)

Different Sorting Techniques

NPTEL Online Certification Courses
IIT Kharagpur



Taxonomy of sorting techniques

NPTEL Online Certification Courses
IIT Kharagpur

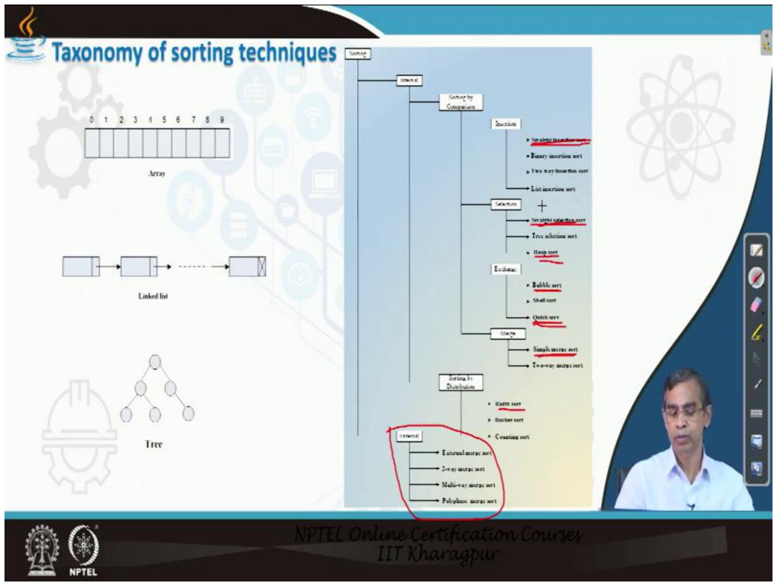
Now, let us start about the different sorting techniques which are known in the field. Now, all the sorting techniques can be divided into two broad categories, they are called internal and external. Internal sorting technique is basically technique when we apply to data which are stored in a primary memory that means in computer memory. On the other hand, external sorting techniques is applicable to data when they are stored externally, externally means, it is stored in some tape or device or hard disk like.

So, this technique is good, I mean external technique is good for or applicable or useful for when huge data to be sorted and this is huge data such that it hold the data you cannot bring to your memory, computer memory or primary memory. So, that is an external, so, this technique is something different, which is not possible to discuss in this course, rather we will limit our discussion to internal sorting technique ideally.

Now, all internal sorting techniques again can be categorized into two classes, it is called the sorting by comparisons and another is called sorting by distributions. Sorting by distribution one sort of (com) technique it is called without any key comparison it is called. So, it is basically starting with key comparison and sorting without key comparison we can say.

Now, when we (dis), when we say about sorting by key comparison, there are again four different principles are followed; one called principle of insertion, principle of selection, principle exchange and principle of enumeration. Now, so, all sorting techniques, they are, these are basically broad classification or taxonomy of different sorting techniques. And based on these different principle there are many sorting techniques are evolved.

(Refer Slide Time: 3:57)



Taxonomy of sorting techniques

Now, again the different sorting techniques demands the data to should be stored in a particular structure. Now, altogether any three, any three, anyone of the, these three data structure like areas, linked list and tree can be considered. So, that is what I want to say that if your data stored in the form of an array, then a particular type of techniques can be applicable not all. Similarly, if your data stored in the form of a tree then you can apply certain techniques not that all techniques can be applied to there.

So, depending on the data structure, the different sorting techniques are to be followed. While we are discussing about different data structure like linked list or tree, we have given some hints about that okay sorting is possible and how it can be there. But now we will discuss about a detailed procedure about different sorting techniques when the data is stored in a different structure.

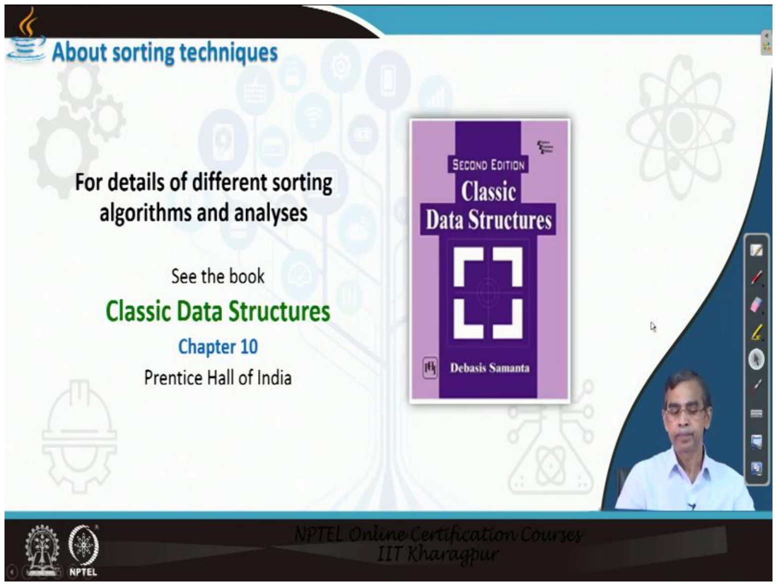
Now so this is another consideration. Now, let us see all internal sorting techniques, how they can be further subdivided into different types. So, any internal sorting technique as shown in this slide can be based on sorting by comparison and sorting by distribution. Now, let us come to the sorting by distribution, they are radix sort, bucket sort and counting sort are very popular and people like it.

And so, for the sorting by key comparison there are many different techniques are known and based on the principle of insertion, there are technical state insertion sort, binary insertion sort, two-way insertion sort and least insertion sort. Now, based on selection, state selection sort and tree selection sort and heap sort.

Based on exchange, the sorting technique is called bubble sort and quick sort and there is a best of principle call enumeration, it is called the merge sort, simple merge. And sorting distribution I told you it is a radix sort then these are the bucket sort and counting sort. These external sorting technique, there are different sorting techniques are for if you want to store the data, which are stored not in computer memory, but in a secondary storage.

Now, as you see, there are many sorting techniques, it is not possible to cover all techniques, however, we will (discuss), we will limit our discussion to certain techniques, which are the fundamental sorting techniques and it has a very, I mean, it is basically it is interesting to study. So, I have mentioned that this simple insertion, then selection, heap sort, bubble sort, quick sort and merge sort, all those things, all the sorting techniques will be covered in this course.

(Refer Slide Time: 6:59)



About sorting techniques

For details of different sorting algorithms and analyses

See the book

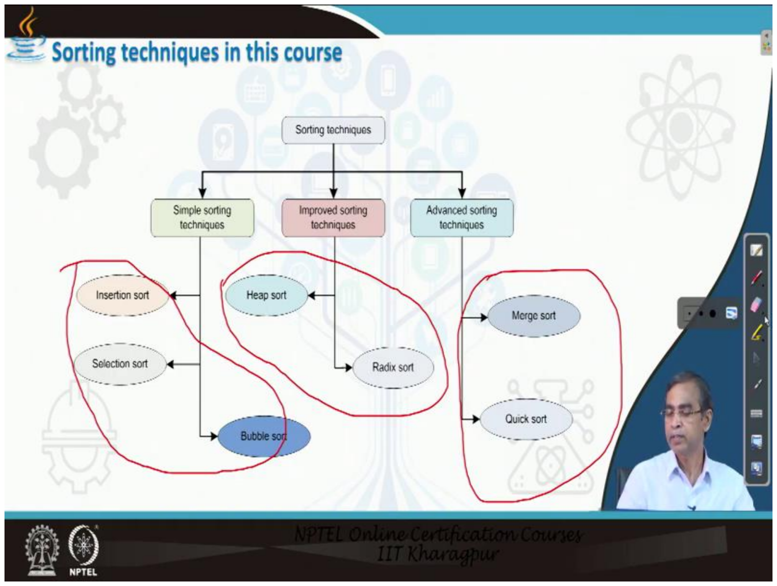
Classic Data Structures

Chapter 10

Prentice Hall of India

So, for many more detailed discussion about many other techniques, so I refer to you this book in chapter 10 there is a detailed discussion of different sorting techniques are available. So, you can follow this book and then learn much more.

(Refer Slide Time: 7:19)



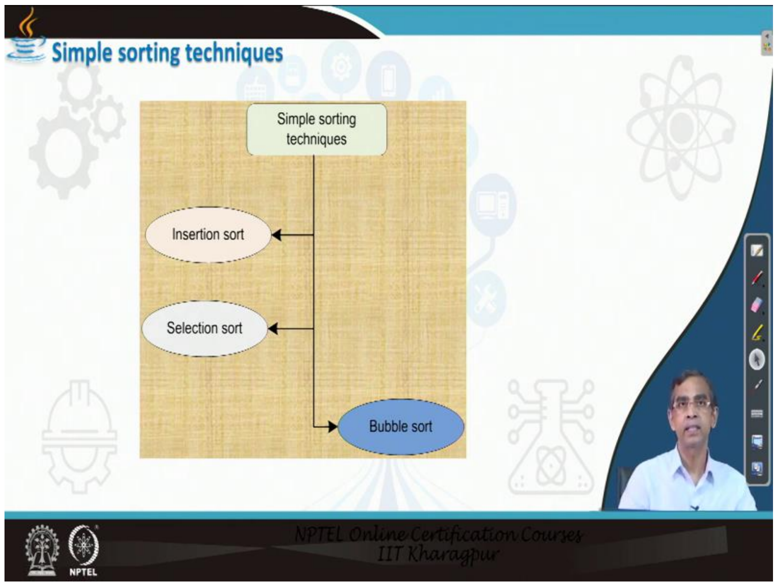
Sorting techniques in this course

Now, the techniques that we will cover in this in this lecture or rather in this course we can say, these are the techniques will be covered, I have already given idea about that this is basically based on the sorting, simple sorting technique, these are the technique is basically based on different principle called improved sorting technique.

And these are the basically technique which is later on devised is called advanced sorting techniques. So, so, all the sorting techniques is categorized into three different criteria as in simple, improved, advanced. So, today, let us first discuss about simple sorting techniques, it is easy to understand actually, because that is why that they are name is simple.

(Refer Slide Time: 8:09)

Simple Sorting Techniques



Simple sorting techniques

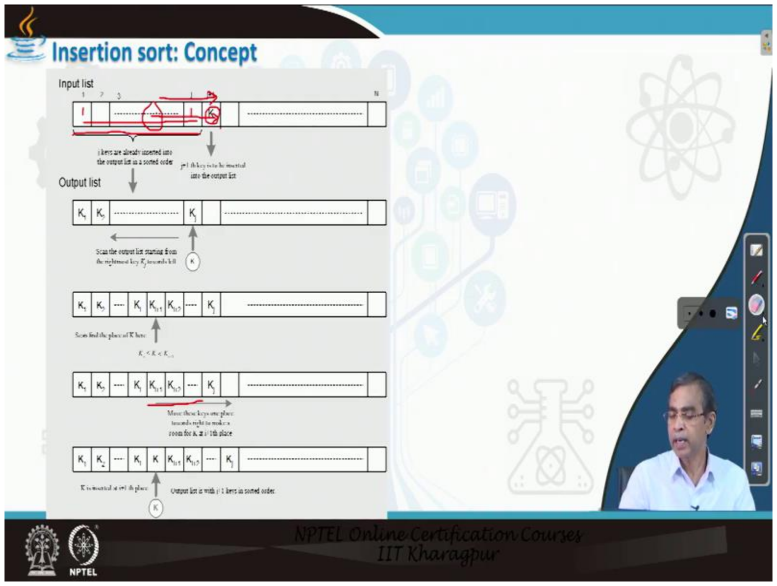
So, in this category, in simple sorting technique, there are three techniques we have chosen one is called insertion sort, selection sort and bubble sort.

(Refer Slide Time: 8:19)

Insertion Sort

So, first we will discuss about insert some sort. Now, here the sorting of elements, which are stored in an array, they also can be equally applied if the data stored in a, but I limit the discussion with reference to the array data structure, linked list as in the same way you can extend, if you understand the array then the linked list also can be understood easily.

(Refer Slide Time: 8:44)



Insertion sort: Concept

Input list
1
2
3
j
j+1
N
i keys are already inserted into the output list in a sorted order
j+1 th key is to be inserted into the output list
Output list
K_1 K_2 K_j
Scan the output list starting from the rightmost key K_j towards left
K
K_1 K_2 K_i K_{i+1} K_{i+2} K_j
Scan find the place of K here
$K_i < K < K_{i+1}$
Move these keys one place towards right to make a room for K at i+1th place
K_1 K_2 K_i K K_{i+1} K_{i+2} K_j
K is inserted at i+1 th place
Output list is with j+1 keys in sorted order
NPTEL Online Certification Courses
IIT Kharagpur
NPTEL

Now, let us discuss about the insertion sorting concept. The idea is that the elements are stored in an array and you have to have several loops, it is called the process or iterations whatever you can tell. So, in each iteration, so it is if it is an ith loop or in ith iteration, so the element Ki needs to be placed in its proper position. Now, how the K ith element can be placed in a proper position I can tell you the idea.

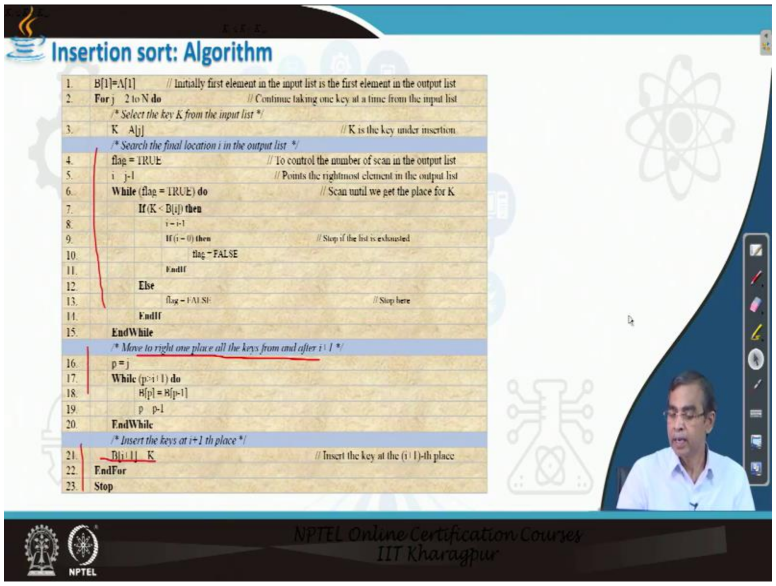
Say suppose, at any moment, this is the already partially sorted part of the list and here in the i plus 1th element that is the element that you want to place it. Now, this element can be any this place or it can be anyone. Now, if this element is the value of this element is any within this, within these values, that means this is the lowest one, this is the highest one at the moment if these values is within this one then this can be placed here, if this value is greater than any value in this list then it will remain in the same place that is idea.

Now, what exactly the idea that you should follow is that first of all we have to this Ky, then we compare this one, whether this value is greater than or you can say that okay that whether this value is greater than or equals to this value. If it is there, then we go to the next position and it will come on whenever we say that one value which is less than this value, then we can place this value there, where we find this test.

Now, in order to place it, what is the idea is that you need to swap this position to this position or you can move all elements from this position to one this one. So, movement of elements is equal because in order to place this element here, we have to make a place shifting all the elements once towards this direction up to this one.

So, there actually the comparison as well as the shifting of elements is required. Now, this is the procedure it is explained in details here. So, if we find that this is the position we can move this and we can place the element here. So, this is a concept that is there we can understand this concept better if we illustrate with some example.

(Refer Slide Time: 11:34)

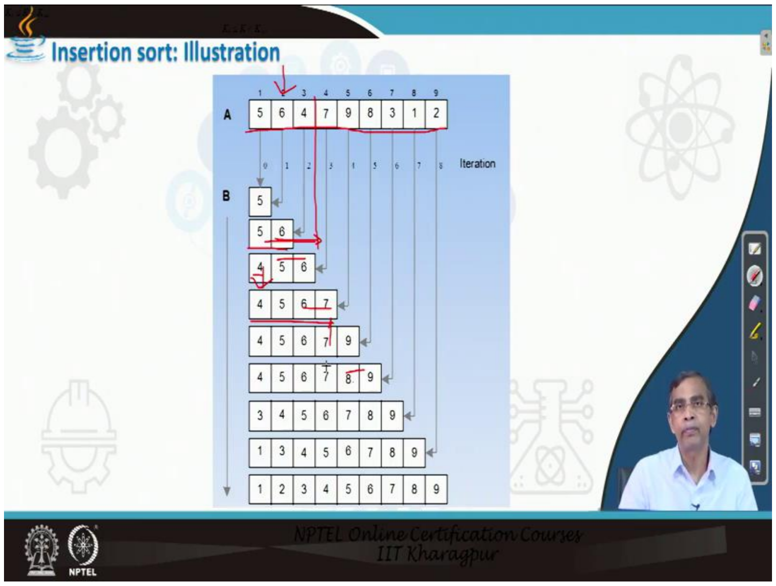


Insertion sort: Algorithm

```
1.  B[1]=A[1]          // Initially first element in the input list is the first element in the output list
2.  For j  2 to N do                       // Continue taking one key at a time from the input list
        /* Select the key K from the input list */
3.      K   A[j]                                            // K is the key under insertion
        /* Search the final location i in the output list */
4.      flag = TRUE                          // To control the number of scan in the output list
5.      i   j-1                               // Points the rightmost element in the output list
6.      While (flag = TRUE) do                     // Scan until we get the place for K
7.          If (K < B[i]) then
8.              i = i-1
9.              If (i = 0) then                           // Stop if the list is exhausted
10.                 flag = FALSE
11.             EndIf
12.         Else
13.             flag = FALSE                                            // Stop here
14.         EndIf
15.     EndWhile
        /* Move to right one place all the keys from and after i+1 */
16.     p = j
17.     While (p>i+1) do
18.         B[p] = B[p-1]
19.         p   p-1
20.     EndWhile
        /* Insert the keys at i+1 th place */
21.     B[i+1]   K                               // Insert the key at the (i+1)-th place
22. EndFor
23. Stop
```

And here is the algorithm that you can think about, this algorithm consists of two parts; first of all you have to find the element in which position it should be placed, so this part is discussed here. Then move to the right one place all the key because you have to make a room for the new the further element to be placed there and then you just simply place the element there. So three steps are there, which is basically written in the form of an algorithm, you can follow this algorithm so, that you can understand better.

(Refer Slide Time: 12:09)



Insertion sort: Illustration

Now, let us illustrate this in a simple example so, that he can understand this concept far better. Starting from the initial loop like and here is a suppose this is the input list that is given to you, which is not in sorted order. Now, first part that is the default only, so this will remain in the first location because we do not have to compare So, we can start our traversal from this point only.

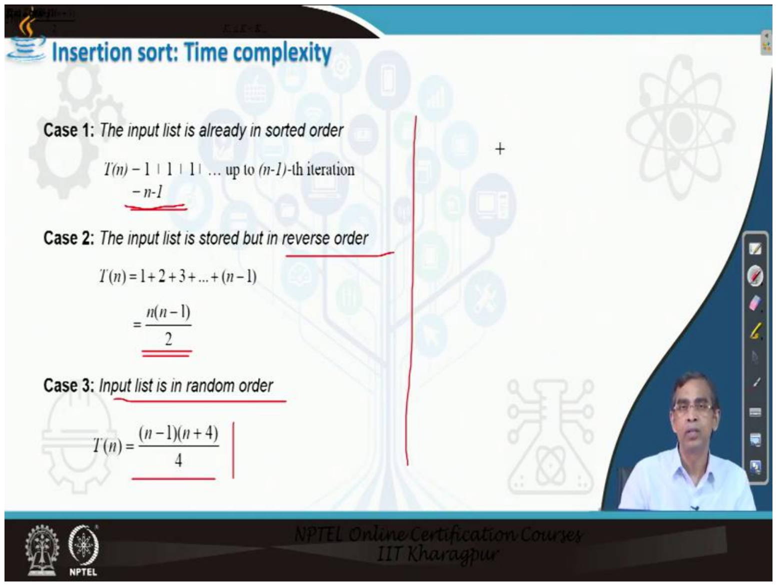
So, this is basically first loop or iteration 1 we can say. Now, here what you can do is that this 6 will be compared with 5 and we see that 6 is greater than that means, we do not have to move so, we can keep it here. So, after the first iteration we can get it then we can take the 4 the next second iteration. Now, we see that 4 is compared to 5, so, we can say that 4 should be placed here.

Then what we can do is that 6 onwards and 5 6 onwards you can move this till this 4th position, we can move into, so that 5 6 can be moved here and 4 can be placed here. Next come to 7, we can see the 7 should be placed here because all elements starting from here it come here. So, it placed here, 9 also same thing.

Now, when we will come to the 5th iteration, the element 8 is coming, we start here and then place the location here. So, moving up 9 here and then place the location here and this way it will continue. Now, here again I am, there is an alternative idea also that you can follow, sometimes it works also, but it is not guaranteed that this work better than this for whatever it is there. The idea is that, alternative idea is that here what I am doing in a in any iteration, we start from the starting location and compare in this direction.

Alternatively, we can start from the, this location and go this direction to find this one, but under any way we have to find that within the partially sorted list the new element that we are considering where it should be placed. So, if you can start from here or if you can start from here, that is not an issue but you have to find it and movement will be always there. Because whenever you see that, this element should be placed here in this sorted part, then we can move, we can move accordingly fewer elements towards the right. So that procedure you can consider. So, this about that technique.

(Refer Slide Time: 14:50)



Insertion sort: Time complexity

Case 1: *The input list is already in sorted order*

$T(n) = 1+1+1+\ldots$ up to $(n-1)$-th iteration

$= n-1$

Case 2: *The input list is stored but in reverse order*

$$T(n) = 1+2+3+\ldots+(n-1)$$

$$= \frac{n(n-1)}{2}$$

Case 3: *Input list is in random order*

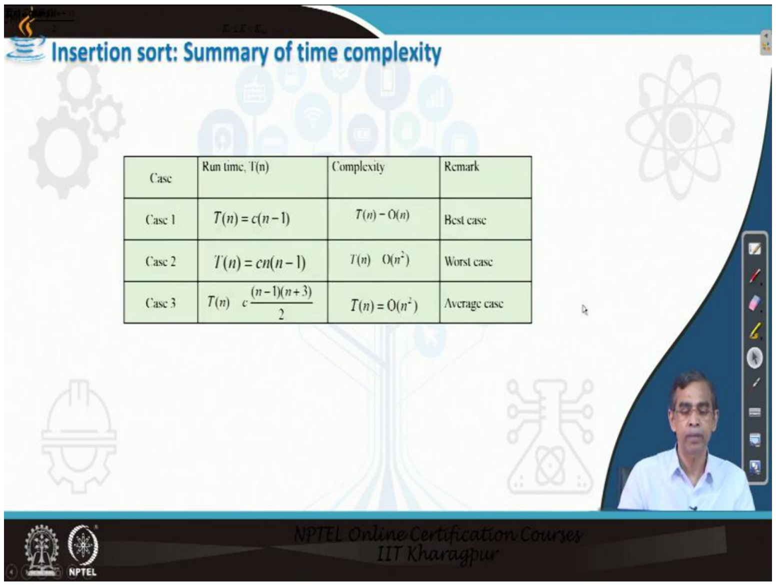
$$T(n) = \frac{(n-1)(n+4)}{4}$$

And then so far the complexity analysis is concerned you can check that there are three cases, one case is basically when the input list is already given in sorted order, then complexity that is recording n minus 1, here n minus 1 is basically I am telling about how many movements that is required. So, a movement is the we see expensive procedure here, comparison is also there. So, competitive as well as movement altogether it basically says that n minus 1 total numbers it is required.

Now, if the input list is sorted, but in reverse order another case, you can see that total number of moments as well as comparison it will take around this order. Now, input list if it is in random order whatever the case that we have discussed it is basically belong to that case, then it can be proved that number of (comp) time, runtime that is required is basically in this order. So, these are number of movements and comparing that is required.

Now, a thorough discussion about this analysis you can follow in the book Classic Data Structure so, that he can understand how this calculation comes, but it is not possible to discuss the detail calculation because it is time consuming and too many discussion is required. Anyway, so, you have understood about the one simple sorting technique called insertion sort.

(Refer Slide Time: 16:08)

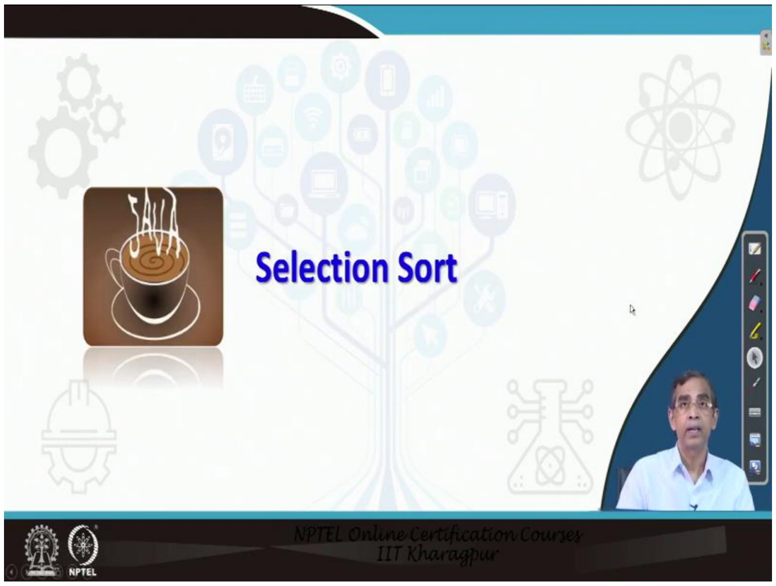


Insertion sort: Summary of time complexity

Case	Run time, T(n)	Complexity	Remark
Case 1	$T(n) = c(n-1)$	$T(n) - O(n)$	Best case
Case 2	$T(n) = cn(n-1)$	$T(n) \quad O(n^2)$	Worst case
Case 3	$T(n) \quad c\frac{(n-1)(n+3)}{2}$	$T(n) = O(n^2)$	Average case

And I can express the complexity in this big O notation you can check it in the best case it is, worst case and average case these are the big O notation.

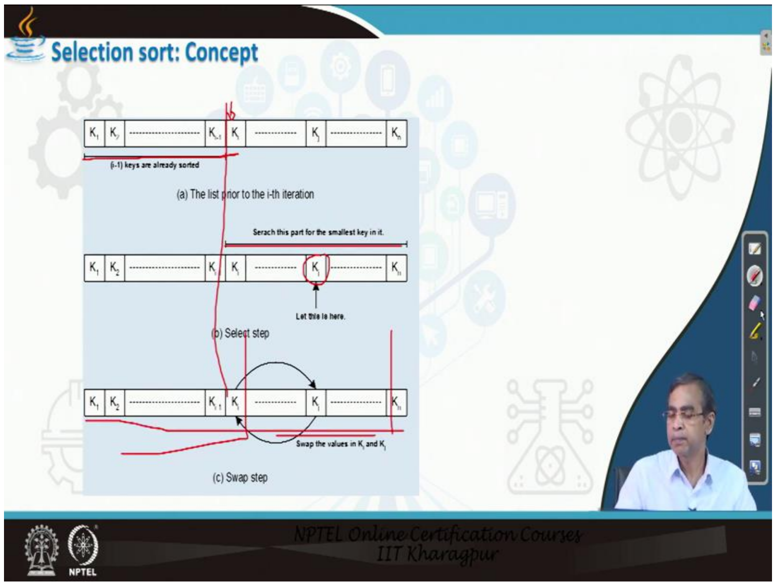
(Refer Slide Time: 16:19)



Selection Sort

Now, let us come to the selection sort another simple sorting technique.

(Refer Slide Time: 16:26)



Selection sort: Concept

K_1 K_2 K_{i-1} K_i K_j K_n
(i-1) keys are already sorted
(a) The list prior to the i-th iteration
Serach this part for the smallest key in it.
Let this is here.
(b) Select step
Swap the values in K_i and K_j
(c) Swap step
NPTEL Online Certification Courses
IIT Kharagpur
NPTEL

This selection sort is the basic principle or the concept that it follows can be expressed like this. So, we can start many iterations here total n minus m on iterations will be needs to be considered. So, now first we have to consider the entire list that is the starting iteration we can say the first iteration. In this first iteration you have to find, which is the smallest element in the list?

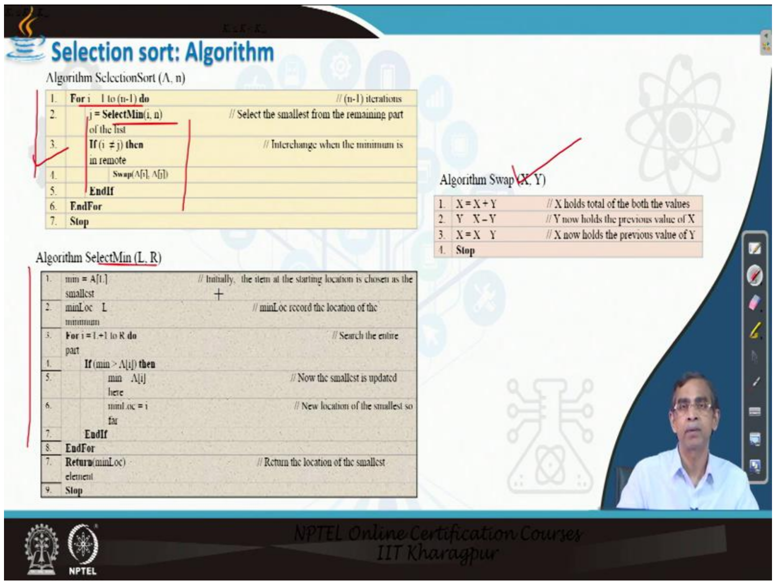
Then you can search for the entire list to find the minimum values in the list and then whatever the minimum values are there, we will be able to, we should place into the first location in first iteration, but whenever you place it, then it needs to be swapped. So, the first, so the element which is in the first location and if the minimum element is space available in the ith location, so we can swap that element between ith location and first location. So, this is a first pass we can say.

The second pass, we have to consider starting from the second position of the array to the (())(17:39). Again we can find a minimum in between position 2 to n. And then find a minimum, again say suppose the minimum is found at ith location, then we have to show up the element between the ith and second location. So, this way if we continue till n minus 1 pass the last element will be automatically sorted in that position and we can stop it.

So, this is the basic concept that is followed here and which is explained in this figure. So, here suppose initially this part is already sorted, then we have to find that minimum next minimum in this position. So, what we can do is that we can search for the minimum value within this part and let it is available there.

So, what you can do these Ki and Kj needs to be swapped. So, Kj will come here, Ki will come here. So, initially after this part was sorted, now after this part, this part is sorted. So, this way will increase up to this one then we can complete the elements to be sorted. So, this is a concept that you can follow. Now, here, let us consider the algorithm that you can think about.

(Refer Slide Time: 18:52)



Selection sort: Algorithm

Algorithm SelectionSort (A, n)

```
1. For i   1 to (n-1) do                                  // (n-1) iterations
2.     j = SelectMin(i, n)          // Select the smallest from the remaining part of the list
3.     If (i ≠ j) then                          // Interchange when the minimum is in remote
4.         Swap(A[i], A[j])
5.     EndIf
6. EndFor
7. Stop
```

Algorithm SelectMin (L, R)

```
1. min = A[L]          // Initially, the item at the starting location is chosen as the smallest
2. minLoc   L                                   // minLoc record the location of the minimum
3. For i = L+1 to R do                                              // Search the entire part
4.     If (min > A[i]) then
5.         min   A[i]                                    // Now the smallest is updated here
6.         minLoc = i                                    // New location of the smallest so far
7.     EndIf
8. EndFor
7. Return(minLoc)                                 // Return the location of the smallest element
9. Stop
```

Algorithm Swap (X, Y)

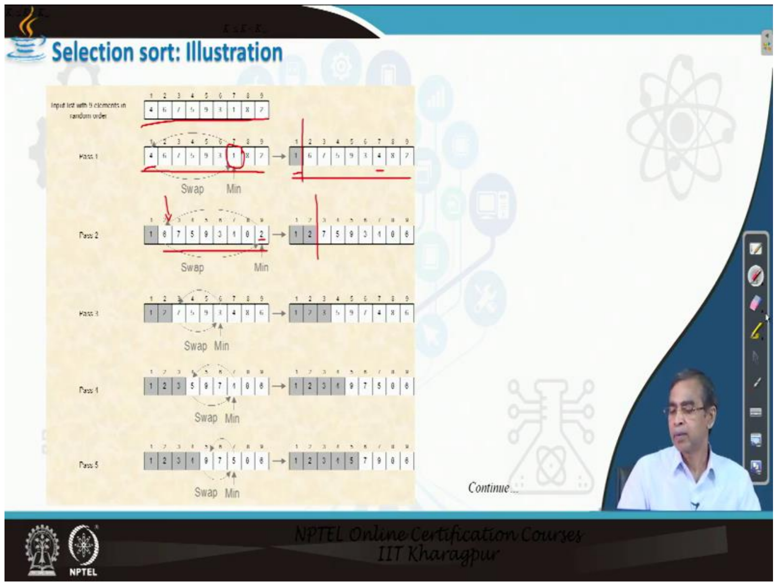
```
1. X = X + Y          // X holds total of the both the values
2. Y   X – Y          // Y now holds the previous value of X
3. X = X   Y          // X now holds the previous value of Y
4. Stop
```

NPTEL Online Certification Courses
IIT Kharagpur

This is the algorithm that is given to you, the algorithm consists of swapping that is the technique that you can consider this is for swap method. There are many other swap method also you can consider and here is basically you have to find a minimum value. So, it basically minimum that is I select minimum and then how the select minimum algorithm is explained here. And then this basically after selecting minimum, swapping and then this iteration, as you see there is loop for i equals 1 to n minus 1 because I told you n minus 1 number of iteration is required.

So, you can implement it. So, this basically the idea So, this algorithm is easily you can write program, not necessary to write generic program, you can write some simple program taking array as an input, and then you can write it and later on you can make into a generic version. So, that any other type of data type can be considered for sorting. So, again that is left as an exercise for you. So, you should study you can practice it.

(Refer Slide Time: 19:52)



Selection sort: Illustration

Now, here is an illustration that you can think about, so, how it works. So, this is the starting array that you can take into account and here the first part that is the first iteration, so 4th element, now in the, in this iteration what you want to do is that, which is the minimum element available in the whole list we have to find it first, we see the minimum element is present here.

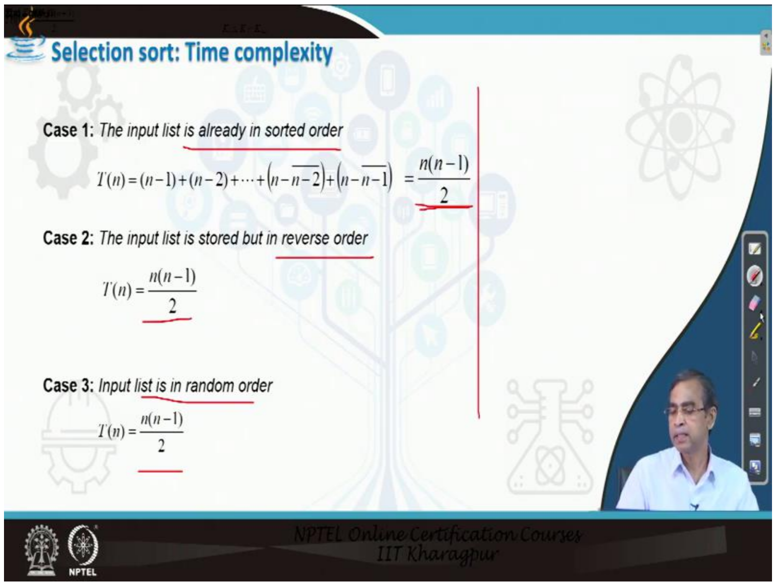
So, what we do you can do is that 4 and 1 should be swapped. So, 1 will be placed here and 4 comes here and this basically the upper. So, up to this part is now sorted. In the second part, so we are here and then we have to find which is the next minimum value that is in this part. Now, we see that 2 is the minimum value, then we can swap and we can bring it here. So, this is up to the part and so on. So, it will continue.

Now, again there may be more than one element see array can contains duplicate elements. So, it is quite possible that that two or more elements are same value are present in the list then how we can do the sorting. So, it is the same we can apply, but whenever the first occurrence that mean first minimum of the first will be considered and then we can replace it.

So, this way this algorithm gives you in place sorting algorithm also called stable sorting algorithm, there is a concept called in place and stable, you can consult the book Classic Data Structure you can find the concept about in place and stable sorting techniques. Anyway, so, this idea whether as a very trivial case or very extreme case, you can consider how the sorting algorithm works or not even in insertion sort also.

Here we have consider the random order you can consider any other order ascending, descending order that is a test case you can check it. And further is that you can consider all elements are equal then whether this will work or not. So, it will also work you can same procedure you have. So, that algorithm is written in that way that it can take all possible cases and then it can work. So, you can write the program and convince yourself that okay it is working for all possible cases it is there.

(Refer Slide Time: 22:07)



Selection sort: Time complexity

Case 1: *The input list is already in sorted order*

$$T(n) = (n-1) + (n-2) + \cdots + (n-\overline{n-2}) + (n-\overline{n-1}) = \frac{n(n-1)}{2}$$

Case 2: *The input list is stored but in reverse order*

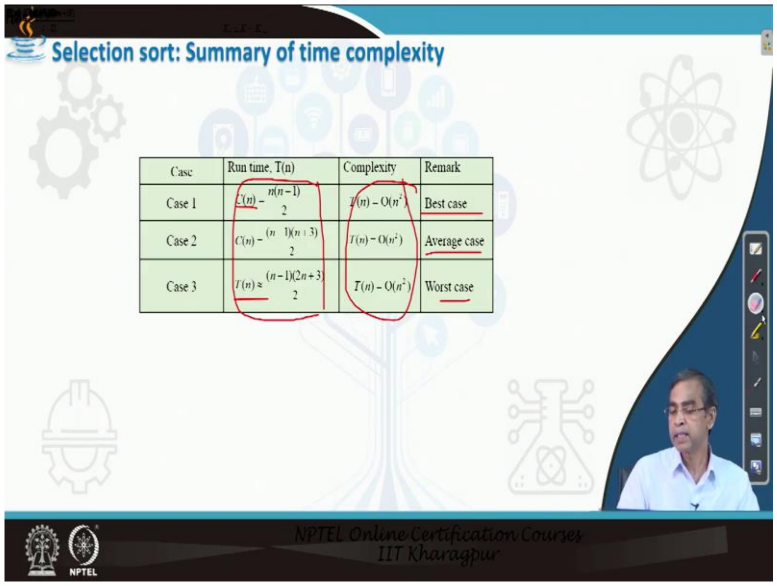
$$T(n) = \frac{n(n-1)}{2}$$

Case 3: *Input list is in random order*

$$T(n) = \frac{n(n-1)}{2}$$

Anyway, so, this is the idea about the selection sort and let us come to the complexity analysis. If the list is already in sorted order, then the total number of comparisons as well as movement, those are required can be calculated as n into n minus 1 by 2, is one case. If it is even in reverse order also, it does take same amount of running time and if it is in the random order it also takes. So, whatever be the cases you see, it basically the same complexity it takes.

(Refer Slide Time: 22:49)



Selection sort: Summary of time complexity

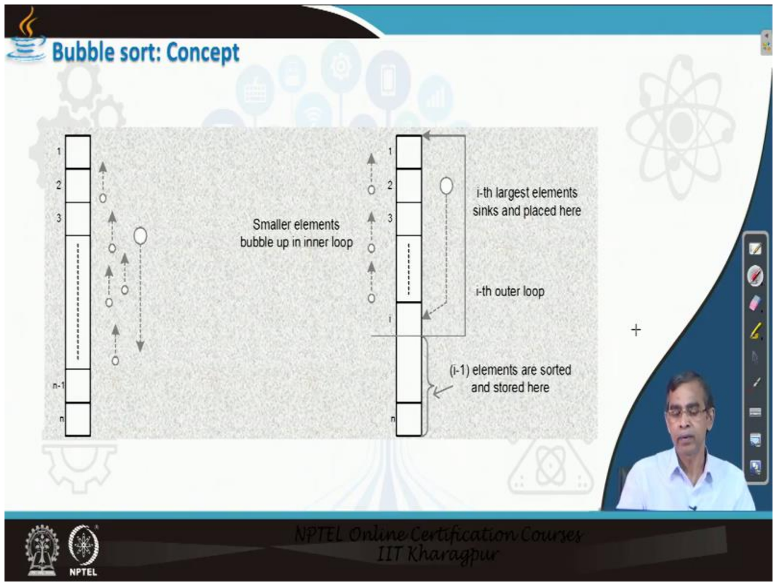
Case	Run time, T(n)	Complexity	Remark
Case 1	$C(n) = \frac{n(n-1)}{2}$	$T(n) = O(n^2)$	Best case
Case 2	$C(n) = \frac{(n-1)(n+3)}{2}$	$T(n) = O(n^2)$	Average case
Case 3	$T(n) \approx \frac{(n-1)(2n+3)}{2}$	$T(n) = O(n^2)$	Worst case

And here is a table that you can consider about, consider about summarizing the complexities of all the cases; best case, average case and worst case. These are the complexity that means SNTT complexity or running time, this is the actual calculation the details of the actual calculation that can be had from the book. So, I advise you to study the book if you are interested how these calculation comes here. So, these are comparison and total complexity time that is required there. So, this is basically the simple and one sorting algorithm selection sort.

(Refer Slide Time: 23:18)

Now, I will discuss about another simple sorting technique, it is a bubble sort.

(Refer Slide Time: 23:23)



Bubble sort: Concept

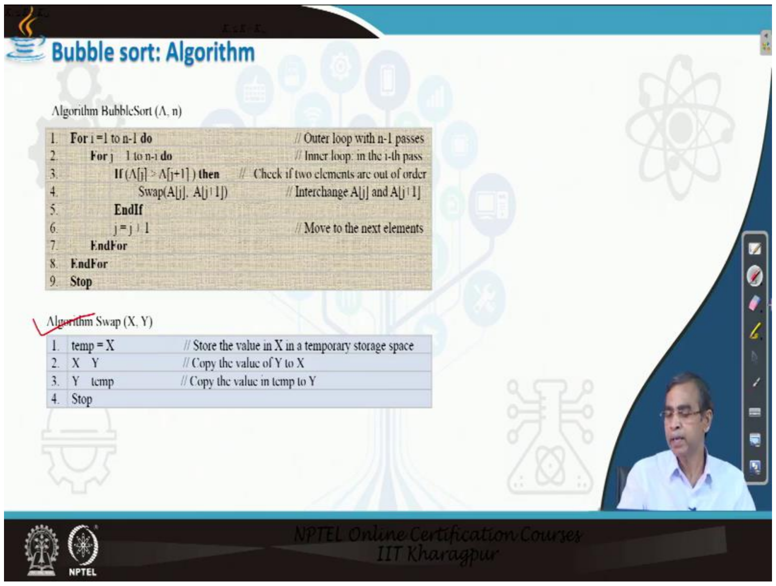
Smaller elements bubble up in inner loop
i-th largest elements sinks and placed here
i-th outer loop
(i-1) elements are sorted and stored here
1
2
3
n-1
n
i
NPTEL Online Certification Courses
IIT Kharagpur

The bubble sort idea came from the bubble as you know the those are the lighter bubbles who go to the top and then heavier bubble should go to the bottom, this is a concept it is there. So, here basically all lighter bubbles up and then heavier bubbles will come there. So, finally at the bottom, it will store the heavier bubble.

Now, here lighter and heavier in the context of numbers if you say the smaller number are the lighter bubbles and then largest number of the heaviest bubble, so that is why. So, this means that in this bubble sorting technique, the idea is that all the heaviest numbers in each iteration go towards the down if we consider that bottom most part of the array or rightmost part of the array store the largest number onwards.

And then the leftmost part of the array or the top, top position of the array store the smallest element. So, this is the idea about this one basic principle that it is follows in this category, in this category. Now, let us see one example so that we can how we can do it. So earlier sorting technique, consider only one loop, but here actually we need two loops.

(Refer Slide Time: 24:50)



Bubble sort: Algorithm

Algorithm BubbleSort (A, n)

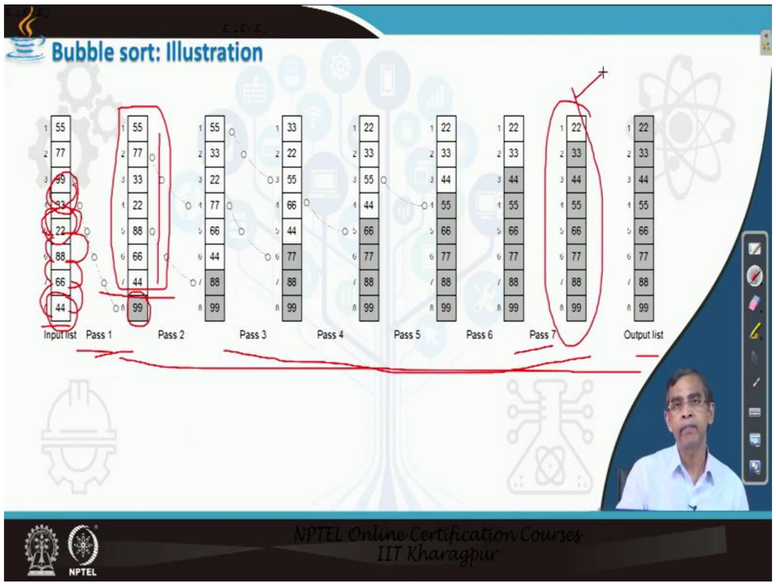
```
1. For i =1 to n-1 do                              // Outer loop with n-1 passes
2.     For j   1 to n-i do                         // Inner loop: in the i-th pass
3.         If (A[j] > A[j+1] ) then    // Check if two elements are out of order
4.             Swap(A[j], A[j+1])                  // Interchange A[j] and A[j+1]
5.         EndIf
6.         j = j + 1                               // Move to the next elements
7.     EndFor
8. EndFor
9. Stop
```

Algorithm Swap (X, Y)

```
1. temp = X       // Store the value in X in a temporary storage space
2. X   Y          // Copy the value of Y to X
3. Y   temp       // Copy the value in temp to Y
4. Stop
```

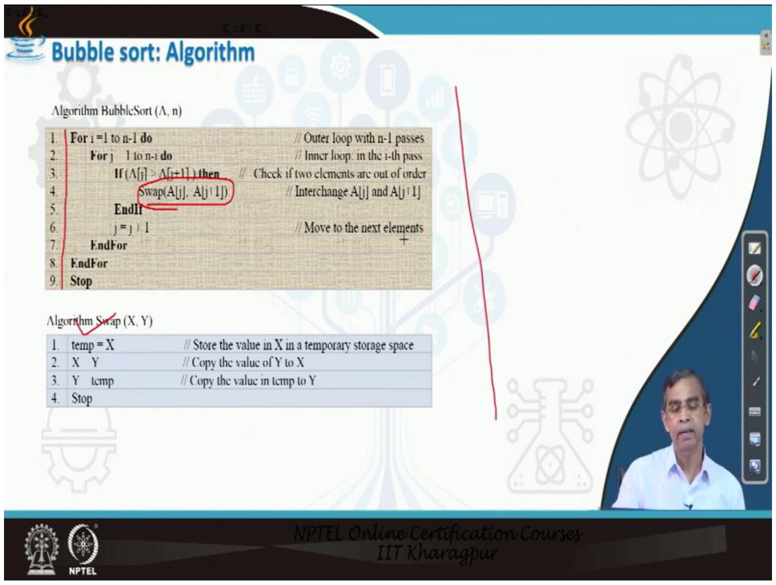
Now here actually, let us see, how this technique can be better discussed. Here is an algorithm, this algorithm consider the swap, this is a swap method you can use because we need the swapping I will come I will come to the discussion how the swapping can be done. And this is algorithm, better I can come back to the algorithm later on, let us first discuss about the technique, so, that we can discuss about the algorithm.

(Refer Slide Time: 25:06)



Bubble sort: Illustration

Input list: 55 77 99 33 22 88 66 44
Pass 1: 55 77 33 22 88 66 44 99
Pass 2: 55 33 22 77 66 44 88 99
Pass 3: 33 22 55 66 44 77 88 99
Pass 4: 22 33 55 44 66 77 88 99
Pass 5: 22 33 44 55 66 77 88 99
Pass 6: 22 33 44 55 66 77 88 99
Pass 7: 22 33 44 55 66 77 88 99
Output list: 22 33 44 55 66 77 88 99



Bubble sort: Algorithm

Algorithm BubbleSort (A, n)

```
1. For i =1 to n-1 do                          // Outer loop with n-1 passes
2.     For j   1 to n-i do                     // Inner loop: in the i-th pass
3.         If (A[j] > A[j+1] ) then            // Check if two elements are out of order
4.             Swap(A[j], A[j+1])              // Interchange A[j] and A[j+1]
5.         EndIf
6.         j = j + 1                           // Move to the next elements
7.     EndFor
8. EndFor
9. Stop
```

Algorithm Swap (X, Y)

```
1. temp = X          // Store the value in X in a temporary storage space
2. X   Y             // Copy the value of Y to X
3. Y   temp          // Copy the value in temp to Y
4. Stop
```

NPTEL Online Certification Courses
IIT Kharagpur

Now, here suppose this is the input list and then these are the different passes that is required finally, we can reach to the output list. Now, in the first pass as you see the idea it basically we compare the two adjacent elements so, 55 and 77 for example, here and if we say that they are not in order, they are not in order means if we say that the previous one is greater than the next one then they are not in order because we want to make in ascending order.

The ascending order says that in order means that the previous one should be lower than the next one even if equal number then we cannot do anything. We should not think about it, just keep it there. So, we can say that okay previous one is (greater) less than or equals to the next one then we can say in order. Anyway, so, here we see 55 and 77 in order so, we do not have to do anything, 77 99 also in order, so do not do anything.

Now, whenever we come here 33 and 99, what we say that they are not in order. So, what we can do that we can interchange them or swap them, as you see this 33 should come here and 99 should come here. So, now 99 comes here, then 99 and 22 we have to again compare then we can see that they are not in order because 22 should come here. So 22 go there and 99 will come here.

Now, you see 99 is coming right this way, then when 99 here 88 is here, then we say that they are not in order and then 88 go there and 99 will come here, so 99 is come here. Now, and then 99 is here 66 you can see that it is not in order, so, we can, 66 can be placed here and 99 will come here. Then 99 is here and 44, we see that 99 is greater than not in order. So 44 and 99 should be swapped it is like this.

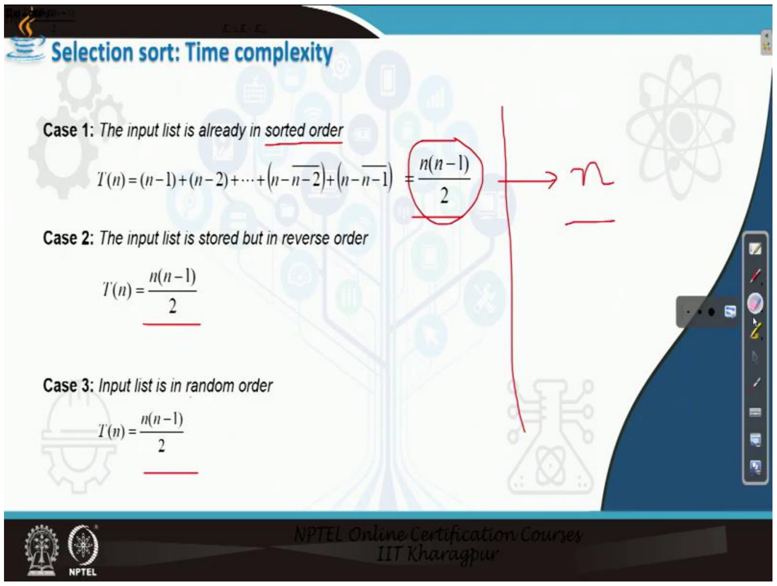
So, ultimately 99 comes in the right, so, this is basically heaviest bubble it comes there and in between whenever swapping the elements is reorganized this way, we can see the those are the smaller bubble actually the lowest element is moved this direction, because here you can 22 moves to where, 33 moves there and so on. So all smaller element move there. So, completes the first pass, so after pass 1, we can this one.

Now, for the next pass what you can do is that as this part is already over, so, we should not consider the next part we will consider we limit our discussion to the, this list only. So, each pass as we see the size of the arrays are decreasing in numbers. So, this, the same procedure we can follow again, starting from here and you can see the 88 is a heaviest bubble, so, it can come here.

And this will continue and if the if the total number of elements is 8 then it takes around 7 passes. So in this after 7 passes, we see that all elements are stored in this order only. So, list is already finished and this will give you the output list that means element is in sorted order. So, this is the idea about sorting technique. And this idea is basically implemented by an algorithm, by an algorithm, this algorithm is basically mentioned here. This is an algorithm in which you can follow to write the code very cryptically is very written in a very nice manner.

So that if you can follow it and then writing the program, if you follow this algorithm and write the program you will be able to understand better and then swap this is the on method you have to define according to this you can define and then you can see that it is working. So, this is left as an exercise, so that you can write the program immediately after learning it so that you can understand, other than, other than you are basically taking it by manually practicing, so better write the code and then practice it you can store the number as an integer and then solve the problem.

(Refer Slide Time: 29:34)



Selection sort: Time complexity

Case 1: *The input list is already in sorted order*

$$T(n) = (n-1) + (n-2) + \cdots + (n-\overline{n-2}) + (n-\overline{n-1}) = \boxed{\frac{n(n-1)}{2}} \rightarrow n$$

Case 2: *The input list is stored but in reverse order*

$$T(n) = \frac{n(n-1)}{2}$$

Case 3: *Input list is in random order*

$$T(n) = \frac{n(n-1)}{2}$$

NPTEL Online Certification Courses
IIT Kharagpur

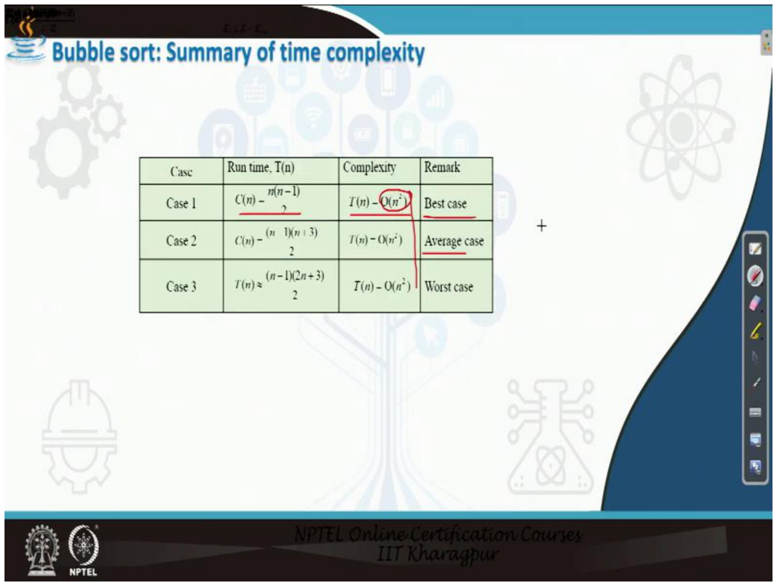
Now, let us come to the complexity analysis. There are again many different cases are there as earlier also we have discussed about whether list given in the sorted order, you will see that the complexity will come into that way. And if it is reverse order we can see this one and input list in random order also this one

Now, this is the idea that is right. But here bubble sort technique if already in sorted order for example, now it is taking unnecessary because in that case, there will be no movement of bubbles, the heaviest movement always in their position, lightest movement always in their position, so no movement. So, we can just have a click there, we can keep a fact, if initially the should be 0 indicating that okay we have to repeat that pass.

Now, whenever we see there is no movement or any swapping, then we can make the pass at the end of the one iteration that pass can be 1. So, whenever the pass is 1, then it will not repeat the same procedure and it will stop. So, in that case, only one pass is required and in one pass only one (compa) n number of comparison is required. So, that way this complexity can be reduced from n square to n only.

So, this is a particular modification of the bubble sort and this modification famously termed as panel sort. In many books, it is written as a panel sort, in the classic data structure also you can have the idea about panel sort. So, a modification of this algorithm you can find in the book I have not discussed here in this lecture.

(Refer Slide Time: 31:19)



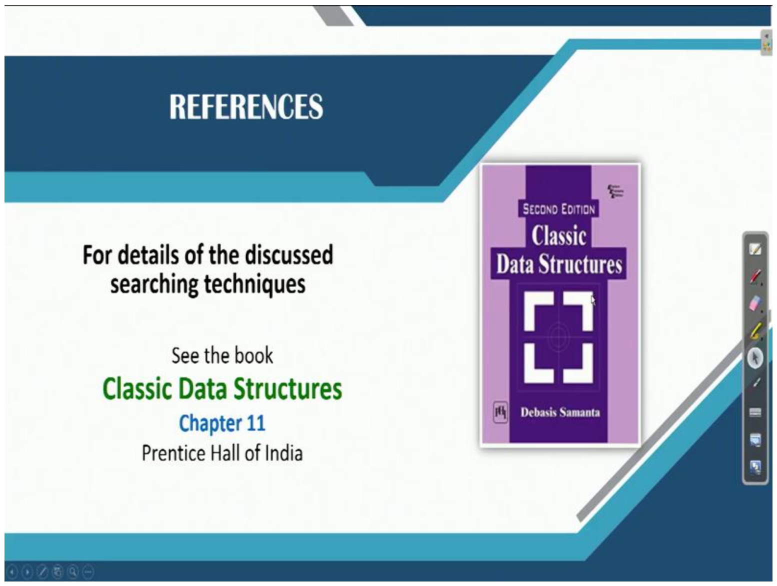
Bubble sort: Summary of time complexity

Case	Run time, T(n)	Complexity	Remark
Case 1	$C(n) = \frac{n(n-1)}{2}$	$T(n) = O(n^2)$	Best case
Case 2	$C(n) = \frac{(n-1)(n+3)}{2}$	$T(n) = O(n^2)$	Average case
Case 3	$T(n) \approx \frac{(n-1)(2n+3)}{2}$	$T(n) = O(n^2)$	Worst case

NPTEL Online Certification Courses
IIT Kharagpur

Now, it is a summary of the different complexity that we have discussed here. So, the best case, the number of comparison and total complexity, that is a running time complexity you can say, average case and in this case is order of n square and if you modify this one, then these can be made into order of n. But the original bubble case is basically order of n square without modification. So, these are the different sorting techniques that we have discussed, these are basically belongs to the simple sorting technique.

(Refer Slide Time: 31:50)



REFERENCES

For details of the discussed searching techniques

See the book
Classic Data Structures
Chapter 11
Prentice Hall of India

And for many more details, discussion and few integrate point regarding the different sorting technique and twist which is basically important for many competitive examination, that you can clear from this book as well as. So, I advise you to follow this book for further study. With these things I would like to stop it here, I will discuss the other sorting technique called improved sorting technique which will be discussed in the next lecture. Thank you very much.