**Week - 02**
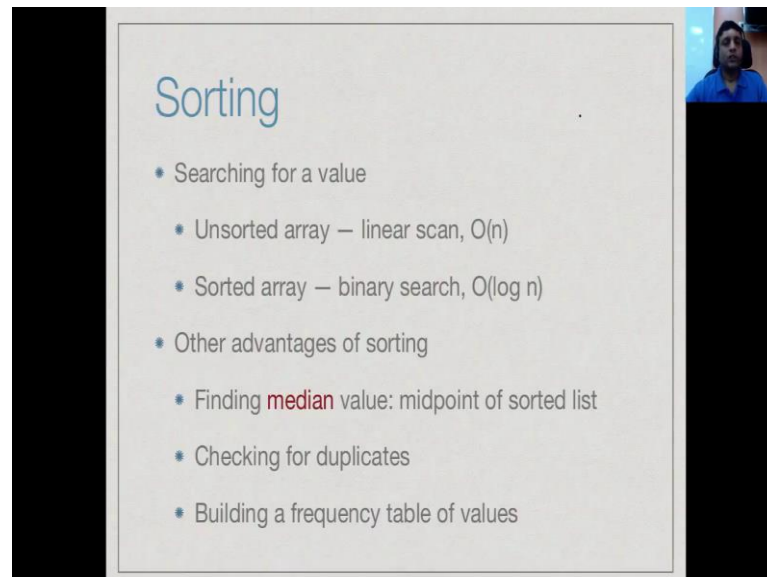**Module - 03**
**Lecture – 11**
**Selection Sort**

So, having seen how to search for an element in array, now let us turn to sorting.
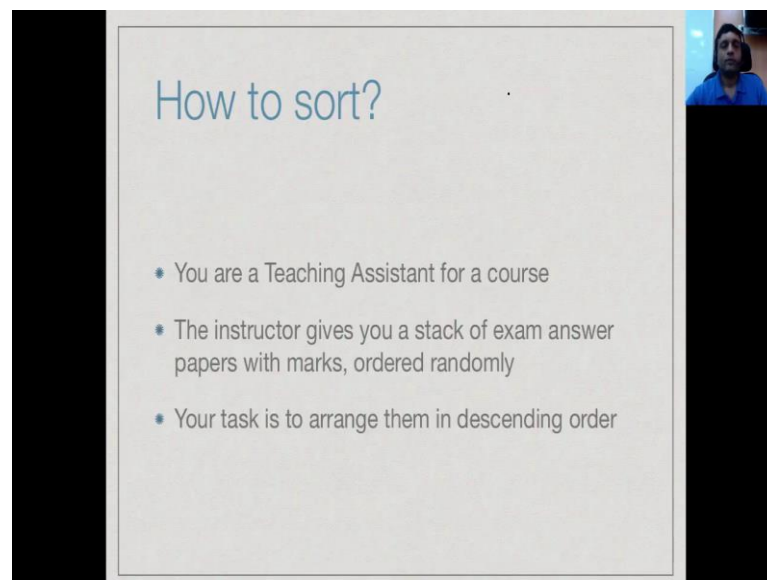
(Refer Slide Time: 00:07)



So, the most basic motivation for sorting comes from searching. As we have seen, if you have an unsorted array, you have to search for it by scanning the entire array from beginning to end. So, you spend linear time searching for the element, whereas if you had sorted array, you can probe it at the midpoint and use binary search and achieve the same result in logarithmic time, and logarithmic time is considerably faster than linear time.

Now, there are other advantages to having the elements sorted. For instance, if you want to find the median, the value for which half the elements are bigger, half the elements are smaller. Well, the median of a sorted array is clearly the midpoint of the array. If you want to do some other kind of statistical things, such as building a frequency table of

values, when you sort these values they all come together, right. So, all the equal values will be in a contiguous block. So, it is much easier to find how many copies of each value are there.

And in particular, if you want only one copy of every value, if you want to remove all duplicates of the array, then you scan the sorted array from beginning to end and for each block of values keep just one copy. So, there are very many different reasons why one may want to sort an array to make further computation on the array much easier.
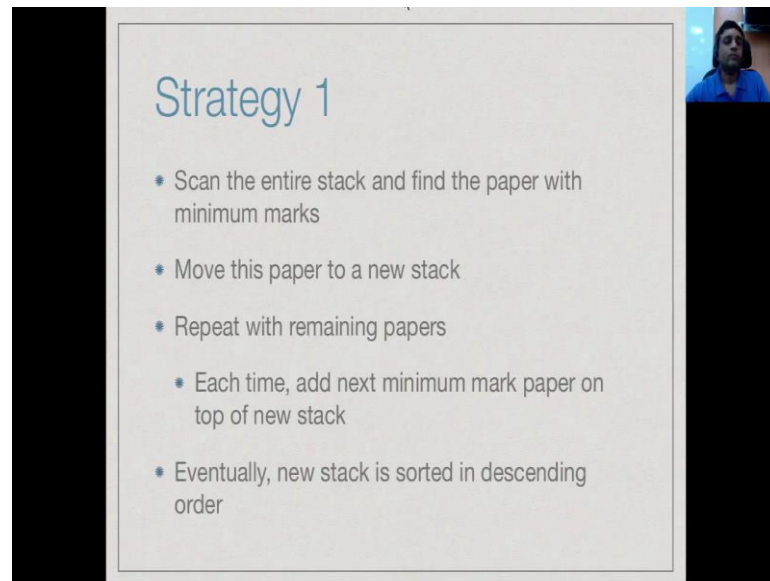
(Refer Slide Time: 01:16)



So, let us imagine how you would like to sort an array. So, forget about arrays and think about just sorting something, which is given to you as a physical collection of object. So, say, you are teaching assistant for a course and the instructor, the teacher who is teaching the course has corrected the exams, now wants you to sort the exam papers in order of marks. So, say, your task is to arrange them in descending order, how would you go about this task?

(Refer Slide Time: 01:47)



So, one nice strategy is the following. You would go through the entire stack and keep track of the smallest mark that you have seen. At the end of your pass you have the paper in your hand, which has smallest mark among the marks allotted to all the students. So, you move this to a new pile, then you repeat the process. You go through the remaining papers after having discarded the smallest one into the new pile and look for the next smallest one, move that on top of the new pile and so on. So, after the second pass you have second smallest mark on the pile. Up to the third pass you have the third smallest mark on the pile and as you can imagine, after n passes you have moved all n papers from the old pile to the new pile in descending order.
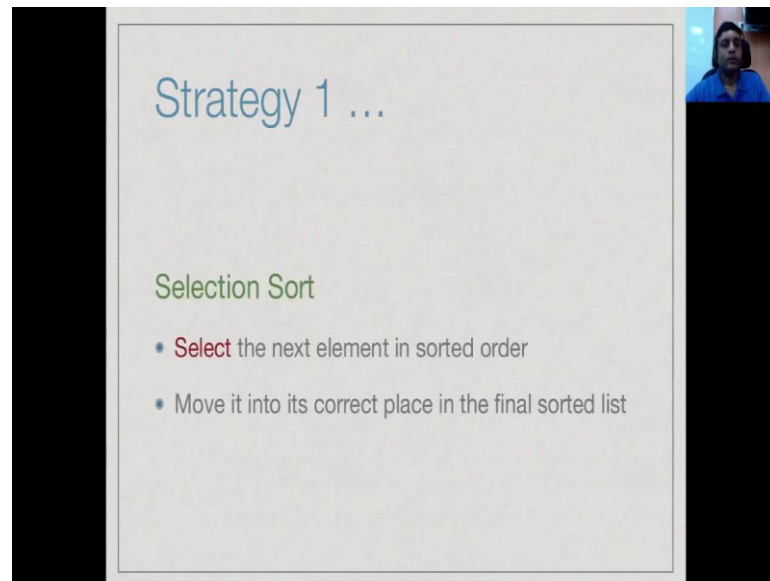
(Refer Slide Time: 02:30)
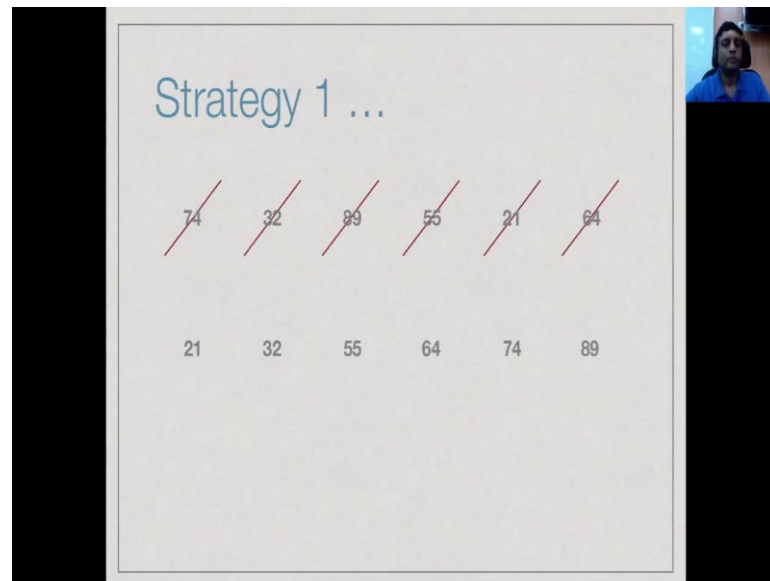


So, this particular strategy can be illustrated as follows. So, supposing we have this list or array of six elements. So, in the first pass we look for the smallest value. So, the smallest value in this case is 21. So, we move 21 to a new list and remove it from the original list. Now, we repeat the scan among the remaining values. The value 32 is the smallest one, so we remove that and move it to new list. We keep doing this. So, at the next step, we move 55 and then we move 64 and then we move 74 and then we move 89, right.

So, in this process if this had been a vertical stack, 21 would be the bottom, 89 would be the top, and so we would have the list sorted from top to bottom in descending order. In this case, it is from right to left in descending order or from left to right in ascending order.

(Refer Slide Time: 03:22)



So, this strategy is called selection sort. So, we select in each round the next element, which is the smallest, and therefore the next we put out in the sorted order and move it to its correct position, that is to the end of the final sorted list.

(Refer Slide Time: 03:40)



Now, this version of selection sort that we just described, builds the second list. In other

words, in order to sort one pile we have to create a second pile of the papers. Now, we can easily eliminate the second pile of papers if we do the following. When we find the minimum element, we know it must go to the beginning of the list. So, instead of creating a new list we move it to the beginning of the current list. Of course, in the beginning of the current list there is another value. So, we have to do something with that value. You just exchange the positions.

So, we find the minimum position and swap the value at that minimum position with the value at the first position. Now, the smallest value in the entire array has moved to the beginning. So, now, we start from the second element onwards and look for the minimum. Again, this will be the second smallest. So, we, now having found that we will move that in the second position in the array value. So, we keep doing this and without using a second list we are able to do the same sorting that we did before.
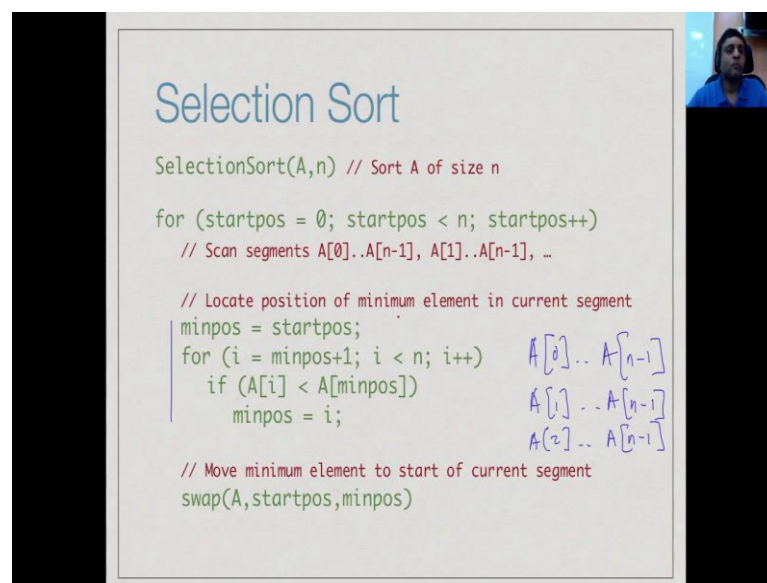
(Refer Slide Time: 04:33)
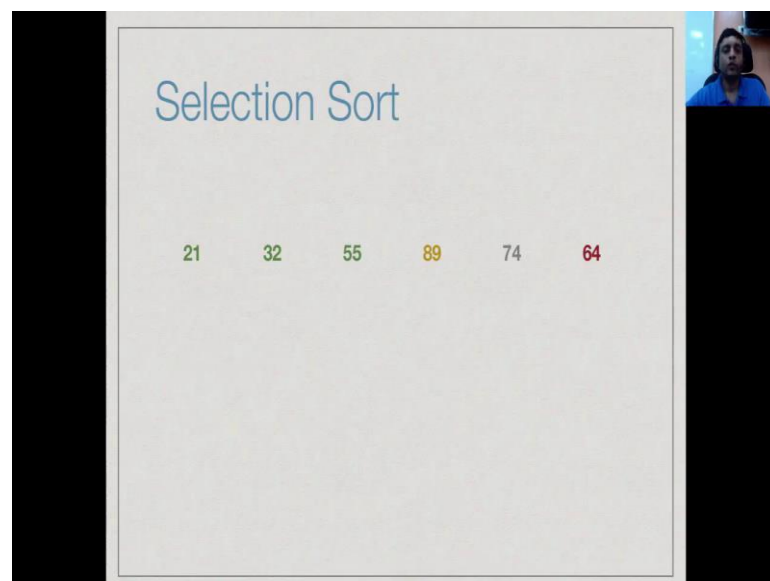


So, to see how this works we take the same array that we had before. As we said, in the first round we identified 21 as the minimum element, which is marked in red and the 1st position is 74. So, we would like 21 to be at the beginning of this list. So, since 74 is there, we exchange 21 and 74 and we get this new list. In this new list, now 21 is marked in green to indicate that it is in its final position. It is the smallest value, it is moved to

where it should be in the sorted list. So, now we scan the remaining elements and we identify that 32 is smallest element. Now, 32 already happens to be in the 2nd position. So, we could think of exchanging it with itself or doing nothing depending on how you want to interpret it. So, 32 now becomes marked in green, so that it stays in the correct position.

Now, in the remaining list 55 is the smallest element mark in red, but it has to be at position 3, which is occupied by 89 marked in yellow. So, we exchange these two and now we have 55 in 3rd position. So, we keep doing this. So, at the next step, we will identify 64 and swap it into the 4th position and then we find, at 74 is already in the correct position. So, we leave it where it is and finally, of course, 89 is the only element. So, when you have a list of one element there is no sorting do be done.

(Refer Slide Time: 05:57)
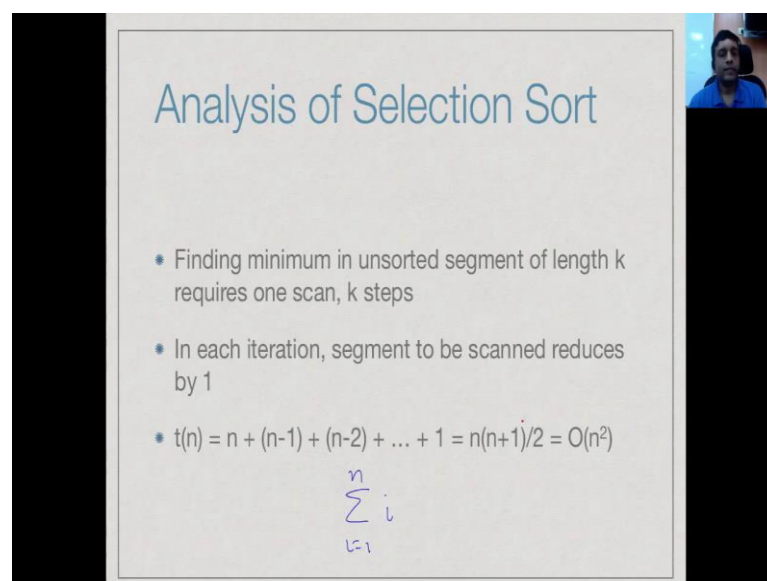


So, this procedure can be described by a very simple iterative algorithm. So, what we do is, that we have, if you remember, we start by scanning the entire array, that is, from 0 to n minus 1, we look for the minimum element in that segment, right. So, we have an iteration, which basically starts at the starting position and then from the starting position onwards it assumes, that the beginning is the, is, is the minimum value and whenever you find a smaller value, you mark that as the minimum position. Now, having done all

this right we have now found the minimum from A 0 to A n minus 1, then you move this value to the beginning. So, you swap the starting position and the minimum position. Now, we have the smallest value at A 0.

So, now, you go back to this loop and now you move starting position from 0 to 1. So, you do the same scan for A 1 to a n minus 1, right and now starting position is 1. So, at the end you swap the second smallest element to A 1, then you do it for A 2 to A n minus 1 and so on, right. So, this is the simple iterative version of selection sort where we just start with the entire array, move the smallest element to the first element, first position, then we take the rest of the array from A 1 onwards move the smallest element to A 1, then start with a 2, find the smallest element, move to A 2 and so on.

(Refer Slide Time: 07:23)



So, how much time does this algorithm take? So, clearly in order to find the minimum element in an unsorted segment of length k we have to scan the entire segment and this takes K steps and in each iteration the segment to be scanned reduces by 1. So, we start by scanning all n elements, then we scan n minus 1 elements, then we scan n minus 2 elements and so on. So, we have over all the number of steps is n plus n minus 1 plus n minus 2 down to 1, which is just the usual summation i is equal to 1 to n of i. And this we know is n into n plus 1 by 2 which is order n square, right. So, this naive algorithm

selection sort in this iterative implementation is n square.
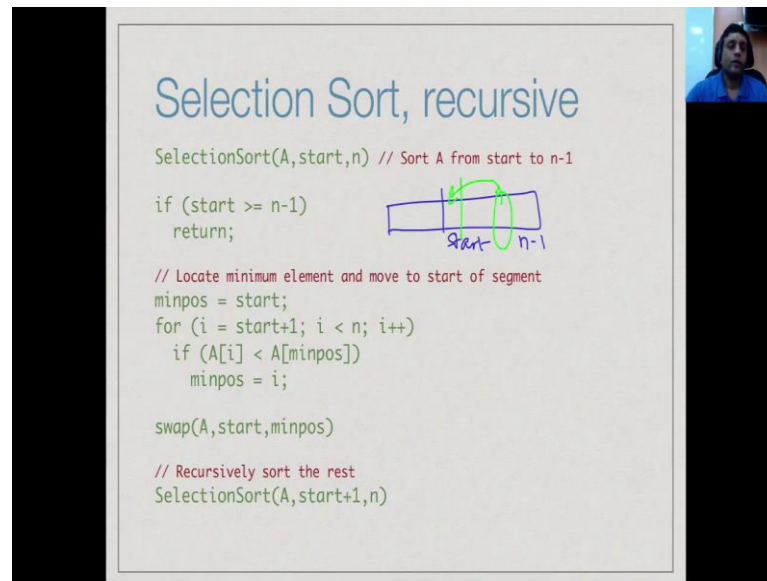
(Refer Slide Time: 08:10)



Now, there is another way of looking at selection sort. So, remember we said, that we start by finding the minimum element moving into the front and then doing the same thing. Whenever you say do the same thing it is useful to think of this as a recursive algorithm just to convince yourself, that what you are doing is correct. So, in order to sort in general A from some position i to n minus 1, what we do is we find the minimum value in the segment, that is, from A to i to A i minus 1 and move it to the beginning, which is A i, right.

So, we have an array A and then we are at a position i and the last position n minus 1. We are trying to sort this. So, what we are saying is, we will find the minimum somewhere and then exchange the value here and now what is now achieved by this is, that this position is now correct, right. So, the value at A i is now in the correct position and therefore, we now, one we need to sort from i plus 1 to n minus 1 and how do we do this recursively? We just apply selection sort. So, we apply selection sort to A from i plus 1 to n minus 1 and then this procedure stops when we have only one element.

(Refer Slide Time: 09:30)
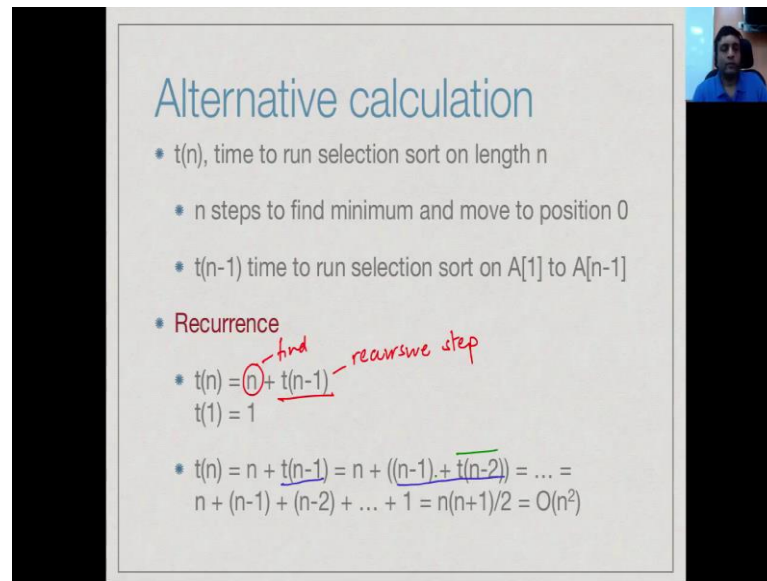


So, if you are going to start from A i is equal to n minus 1 to n minus 1, then we do nothing, right. So, this is the reason to think of it recursively. One reason is, that the code becomes a little easier to read. So, you say, that you want to start sort A from start to n where start is the index of the unsorted segment where it begins and n is the total size. So, the last position is likely n minus 1. So, if start is n minus 1 or bigger, then you do nothing, you just return, otherwise you find the smallest value in this segment from start to n minus 1, right. So, this is the same loop we did earlier.

We start with the minimum position being the beginning of the segment and then we keep walking forwards and whenever we find a value which is smaller than the minimum position, we update it and at the end of this loop we exchange the position that we have found the value at that position with a starting position. So, now, the correct value, so some value from say here has moved to the correct position and now we recursively sort A starting at position start plus 1 up to n. So, is this any different in complexity?

So, let us, right, so remember when we did recursive calculations we said, that typically, you would write a recurrence, right. You will express the complexity of the algorithms itself in a recursive way. So, suppose t n is the number of steps you need to run selection sort in length n, ok. So, the first thing is, it requires n steps in order to find the minimum and move it to the beginning and then having moved it to the beginning the rest of that is array that is left to sort of the size of n minus 1. So, you need time recursively t n minus 1, right. So, therefore, you have t n. So, this is the time to find the minimum. t n is, t n is n plus t n minus 1 which is the recursive step.

And we said, that if we are sorting an array of size 1, we do not have to do anything, we just return. So, t of 1 is 1. So, t of n is n plus t n minus 1. But now, if I take t n minus 1 and expand it using the same expression, I get n minus 1 plus t n minus 2. Now, if I expand t n minus 2, I will get A n minus 2 t n minus 3 and so on. And so this will expand out to exactly what we got for the recursive, for iterative algorithm, namely n plus n minus 1 plus n minus 2 down to n, right. So, what we have seen in this unit is that we have seen, that a very natural algorithm, which we would apply if we did things by hand called selection sort can be formalized both iteratively and recursively and it gives us the complexity, which is order n square.