

Problem Solving through Programming In C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

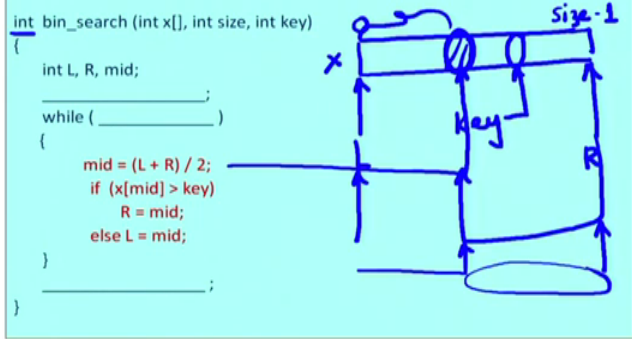
Lecture - 44
Binary Search (Contd.)

We were discussing about the binary search function or binary search procedure.

(Refer Slide Time: 00:25)

The basic search iteration

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not
found, return -1 */
int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    while ( _____ )
    {
        mid = (L + R) / 2;
        if (x[mid] > key)
            R = mid;
        else L = mid;
    }
    _____ ;
}
```



So, here we can see that binary search is of type integer is a function of type integer that will be returning an integer. And it has got the parameters the list or the array which is being designated as x here, the size of the array and the key. Now if the key appears say, the array is here, the array is this, or the list is this, this is called x. And it starts with 0, and the last one is therefore, whatever is in size minus 1.


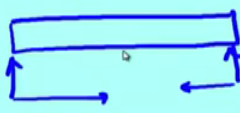
So, if the key there is a key, and if the key appears anywhere here, it will return the position of the point where the key is, if this be the position that be returned. And if not found it will return minus 1, so either it will find the position send the position. So, that is also integer, or minus 1 hence the height is integer, alright. Now we have got 2 pointers, one is L the left pointer left index and the right index of this. And from there we are finding out the middle. So, we are finding out midpoint by this, like, now if the key is smaller than the middle element.

If the key is smaller than this middle element, then obviously, our search will be on this side. Therefore, this R will be moved to this point. L will remain here and my search will be restricted in this zone. Otherwise L will be moved here, R will not be disturbed and the search will be restricted to this zone. This much has been achieved by this piece of code, what else do we need? What else do we need to do? Now how long will this go on?

(Refer Slide Time: 02:43)

Return result

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not found, return -1 */  
  
int bin_search (int x[], int size, int key)  
{  
    int L, R, mid;  
    _____;  
    while ( L+1 != R )  
    {  
        mid = (L + R) / 2;  
        if (x[mid] <= key)  
            L = mid;  
        else R = mid;  
    }  
    if (L >= 0 && x[L] == key) return L;  
    else return -1;  
}
```


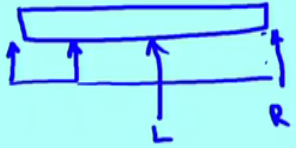


So, in this way, what will happen is, I will move either L or R, either this or this, and when will this be completed, how long will this loop go on? If you think a little bit we will find that it will go on as long as L and R are not crossing over. So, if I shift and search this part R is here.

(Refer Slide Time: 03:15)

Initialization

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not found, return -1 */  
  
int bin_search (int x[], int size, int key)  
{  
    int L, R, mid;  
    L = -1; R = size;  
    while ( L+1 != R )  
    {  
        mid = (L + R) / 2;  
        if (x[mid] <= key)  
            L = mid;  
        else R = mid;  
    }  
    if (L >= 0 && x[L] == key) return L;  
    else return -1;  
}
```



Suppose I so, I initially start with L equal to be minus 1 and R to be the size. So, I am shifting from so, here is my array and L value of L is minus 1. So, it is here and R is equal to size; that means, pointing here size plus 1; that means, sorry this point is size minus 1. So, that is actually here, and while as I share, I shift L and I shift between this part etcetera.

Then my shift L, R is here; as soon as L and R crosses over, if R will R moves this side then also a L and R can cross over. If they cross over then that is the; that is my point where I should stop.

(Refer Slide Time: 04:18)

Initialization

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not found, return -1 */  
  
int bin_search (int x[], int size, int key)  
{  
    int L, R, mid;  
    L = -1; R = size;  
    while ( L+1 != R )  
    {  
        mid = (L + R) / 2;  
        if (x[mid] <= key)  
            L = mid;  
        else R = mid;  
    }  
    if (L >= 0 && x[L] == key) return L;  
    else return -1;  
}
```

So, I initialize this; so, and I go on shifting this, depending on either L or R. That is exactly what we are doing when we took the say 1, 2, 5, 7, 9, 13 like that 15. So, I start it with the middle number, and when the key was suppose my key is 16, all right? Or say.

Let me erase this, my key is say 8; 12. Suppose, my key is 12, what will happen? My L is here, there is L, this is R, I start with mid is 7 so, key is greater than this. So, I will have to restrict my search in this zone I take the middle of this. So, R becomes this, this becomes R, and I am sorry, am sorry R remains the same. I have to search between this zone; so, what I do? I shift L I have to search between this zone. So, I shift L to this. So, this is my 12, and this is R I have to search in between this. I will find this to be the mid, fine? And I find that the key is less than the mid; so, I will restrict my search within this zone, this zone, within this zone right.

Since it is less so, then what should I do? If since am moving on this zone, I will shift R keeping L fixed I will shift R here. So, this will be R now; now between this n L l and R I find mid, this is my mid ok. I find 9 to be less than 12; that means, it should be on this side. So, I move L, as I move L; L is crossing R; that means, now I have come to a position where I have exhausted everything, and I could not find the key, all right?

So, if L is greater than 0, and x L is equal to key, L is greater than equal to 0. And x L is the is a key then I return to the point, otherwise return minus 1. So, that is the binary search algorithm ok. This is how am carrying out the algorithm through a C code.

(Refer Slide Time: 07:27)

Binary Search Examples

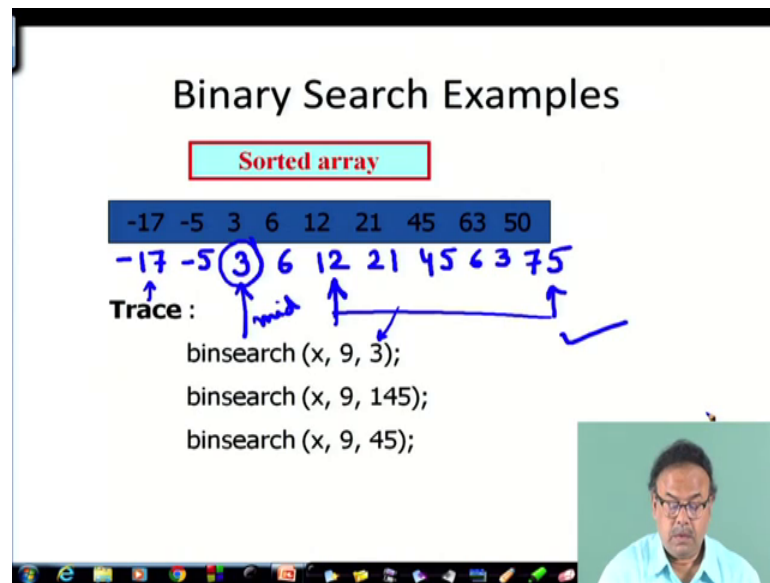
Sorted array

-17	-5	3	6	12	21	45	63	50
-----	----	---	---	----	----	----	----	----

-17 -5 3 6 12 21 45 63 75

Trace :

- binsearch (x, 9, 3);*
- binsearch (x, 9, 145);*
- binsearch (x, 9, 45);*



So, here is an example am writing down an array. Suppose, sorry am writing down a sorted array, suppose it is minus 17, minus 5, 3, 6, 12, 21, 45, 63 and 50. 63 it cannot be 50, say this one is say this up to 63. That is my array.

Now if I carry out so, the size is 1, 2, 3, 4, 5, 6, 7, 8. I must have one another one. So, let me have 75, and if 3 be my key this is the key. So, what will happen? I will start L here, R here, and I will find the midpoint here. So, the midpoint will be here, 12 and 12 is greater than the key. So, my search will be within this zone, this R will be shifted here. So, this is my now R, and this is L, I will find out midpoint 3 here now. As soon as I come to this mid, and I find that this mid is matching the key my search is found, all right? I get I get my key in a particular position.

(Refer Slide Time: 09:24)

Binary Search Examples

Sorted array

-17 -5 3 6 12 21 45 63 50

-17 -5 3 6 12 21 45 63 75

Trace :

binsearch(x, 9, 3); → L=-1; R=9; x[4]=12;
L=-1; R=4; x[1]=-5;
binsearch(x, 9, 145); L=1; R=4; x[2]=3;
L=2; R=4; x[3]=6;
binsearch(x, 9, 45); L=2; R=3; return L;

We may modify the algorithm by checking equality with x[mid].

13

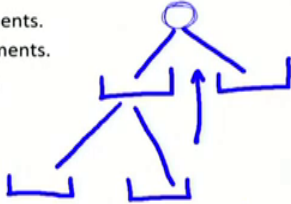
So, you can write it in different ways. So, so, in this way, if I do that, then I will get different results. So, minus 17 I will just missed out minus 5, 3, 6, 12, 21, 45, 63 and 75. So, if for this, my L was at minus 1 R was at 9, and I my mid was this x 4 12 ok. So, then x 1 is minus 5. I carry on doing this. L is still 1, R is 4, R has been shifted to here. And x 2 is 3, I found it.

So, that is my result, in that way I will carry out and you can find out that L and R and then I returned. L I go on shifting. So, that is what it has happened. I have got the mid value, and I have got the value, but then still am going on shifting. You can you can also, what I have suggesting is, it is also possible to check the equality with the x mid as soon as I find that the element has matched with the mid then it is found.

(Refer Slide Time: 10:46)

Is it worth the trouble ?

- Suppose there are 1000 elements.
- Ordinary search
 - If **key** is a member of **x**, it would require 500 comparisons on the average.
- Binary search
 - after 1st compare, left with 500 elements.
 - after 2nd compare, left with 250 elements.
 - After at most 10 steps, you are done.



14

I think you have understood it you can try writing out the algorithm yourself these are binary search we will see later that it can be also done through recursion.

Now why are you doing so much? Earlier, linear search was very much simple, right? Here suppose we had thousand elements, if we had thousand elements, then the ordinary search if a key may key is the member of x , it would on an average require on an average it would have required 500 comparisons on an average. But what will happen in the case of binary system search? After the first compare 1000 elements.

So, we are left with only 500 elements, because I start it from I start it with an array, and then based on that I have either gone on this half of the array or this half of the array depending on whether the key is less than or greater than the midpoint this is the mid. So, on an average first compare I am left with 500 elements. Next compare this part is again divided if it is on this side am left with 250 elements, right. After at most 10 steps, we are done in that way I go on dividing it and after at most 10 steps I am done.

So, I have thousand elements ok. So, I can so, in that way I get 250 elements, and 125, 125; then half of that 60, 65 or something like that.

(Refer Slide Time: 12:42)

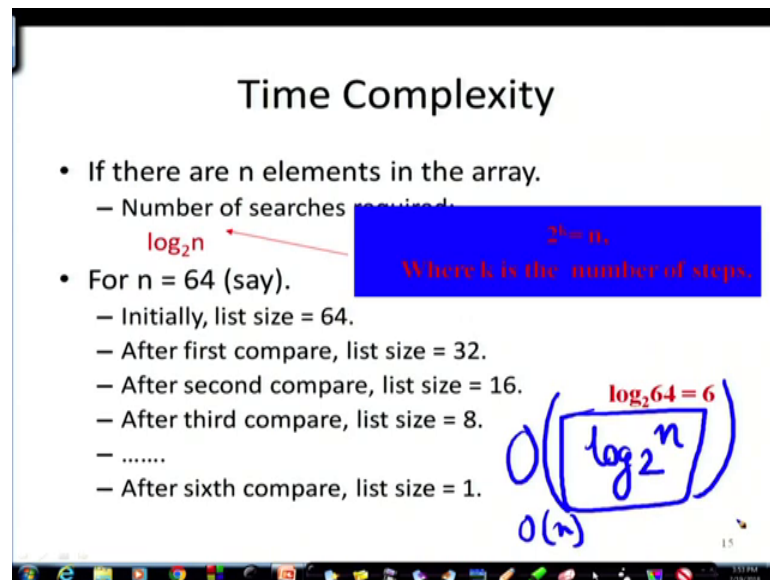
Time Complexity

- If there are n elements in the array.
 - Number of searches required is $\log_2 n$
- For $n = 64$ (say).
 - Initially, list size = 64.
 - After first compare, list size = 32.
 - After second compare, list size = 16.
 - After third compare, list size = 8.
 -
 - After sixth compare, list size = 1.

$2^k = n$
Where k is the number of steps.

$O(\log_2 n)$
 $O(n)$

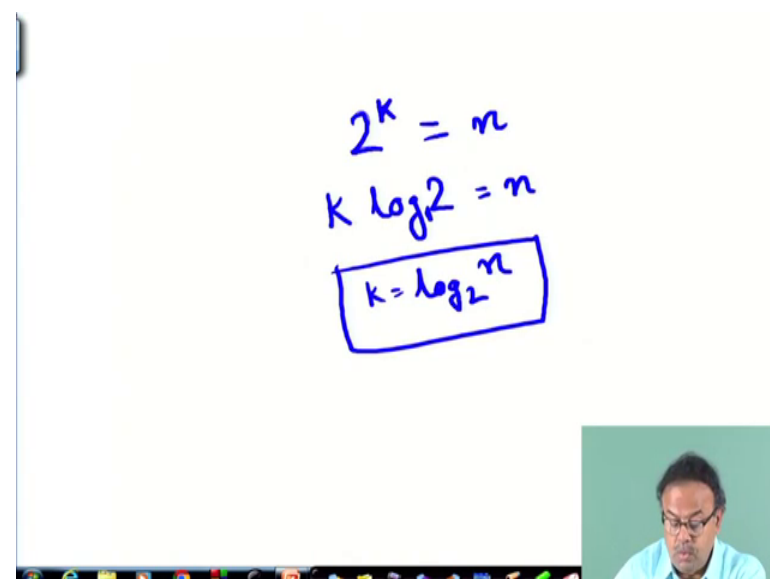
$\log_2 64 = 6$



In that way, number of steps that I will be following will be around 10. In general, if there are n elements in the array, number of searches required is 2 to the power k should be n , where k is number of steps. For n if there be 64. Initially, the list is 60 the size is 64, after first comparison list is 32. After second it is 16, then 8, then 4, then 2 and then 1 so, 6 steps. So, basically at every step am breaking it down into half. Therefore, if I need the k steps.

(Refer Slide Time: 13:30)

$2^k = n$
 $k \log_2 = n$
 $k = \log_2 n$



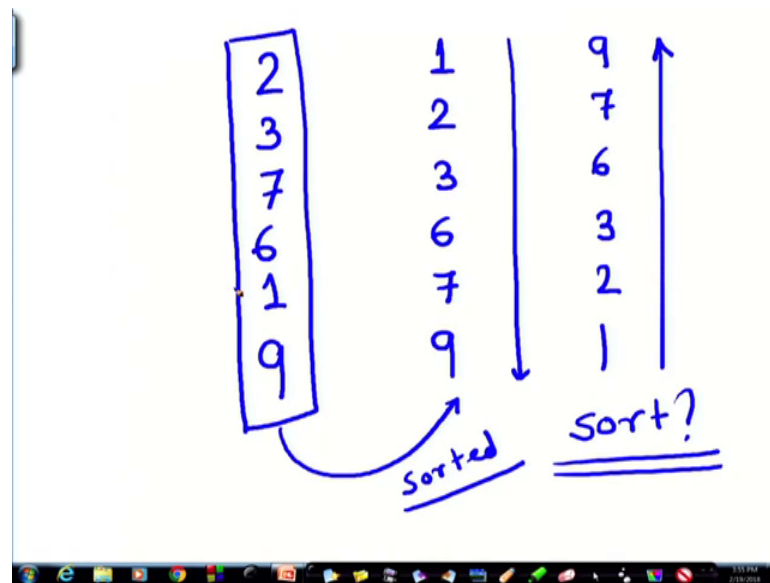
Then, if I need k steps, then 2 to the power k should be equal to n ; that means, how many steps are there? How many, what is k if I take \log and $k \log_2 k$ is equal to n . Therefore, k will be $\log_2 n$ to the power base of \log_2 . So, $\log_2 64$ to the base 2 is 6 . We will be completing in 6 steps. Whereas, in the worst case I would have needed 64 comparisons in the case of linear search. Now given this idea of binary search.

So, what have you seen? We have seen binary search is a search algorithm, that enables us to search for an element much faster, than linear search the approach being that we divide the array to be searched in half every time. So, we divide it into half and restrict our search in half, and in the next step I further divide it into 2 half and restrict it to even a smaller steps.

So, every time am halving whatever array am checking. So, 2 days to the power k time by half date where k times have half date 2 days to the power k should cover the entire n . And so, from there we get the complexity to be of $\log_2 \log_2 n$ to the base 2 $\log_2 n$ to the base 2 in general it will be $\log_2 n$ if 64 is replaced with n (Refer Time: 15:43) then it should be $\log_2 n$ to the base 2 , that is my time complexity compared to. So, I can say this order of $\log_2 n$ to the base 2 , compared to order of n , that was the case in the case of a in the case of linear search.

Now, we will move to another very fundamental problem in programming. We often need that say for example, here we have done one thing that is, I had a array like say $2, 3, 7, 6, 1, 9$.

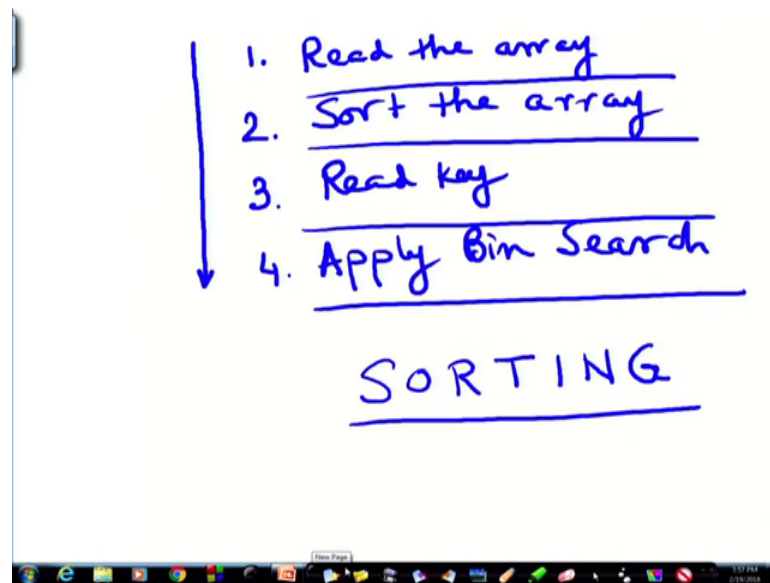
(Refer Slide Time: 16:23)



Now this was an array, on which I cannot run binary search. Why I cannot run binary search? Because binary search can be done only on sorted array. So, if I sort it, either in increasing or decreasing order. Say, for example, it becomes 1, 2, 3, 6, 7, 9. This is a sorted array, or the other thing could be 9, 7, 6, 3, 2, 1. These also a sorted array. In the other direction, it is descending order this is the ascending order ok.

So now another problem is, to arrange the things arrange the objects, suppose they are number of numbers like this. We have to arrange them, sorted or arranged in a particular way. The question is, how to do sorting, how to sort? So, suppose I have given this array to my program, this is the input. Then if I want to apply binary search, then I cannot directly apply it apply binary search on this. So, what I can do? I can first take the array say I read the array let me write.

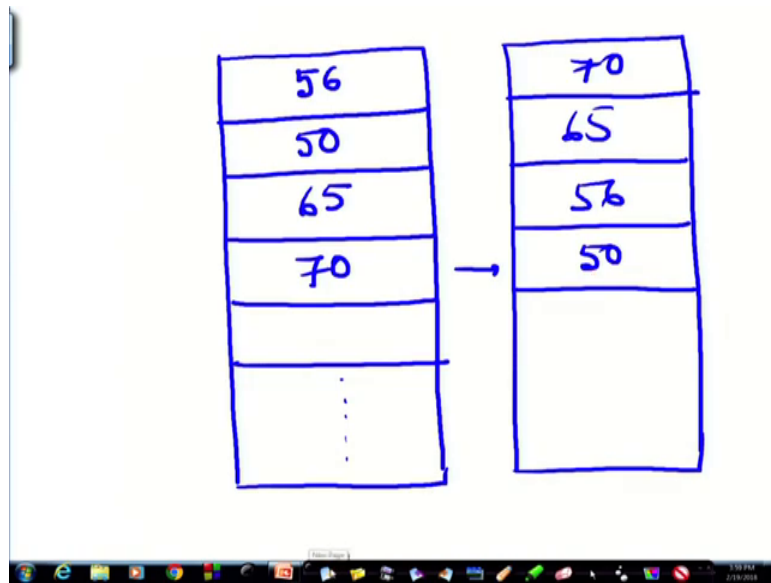
(Refer Slide Time: 18:00)



Then sort the array, and then read key apply binary search. Now binary search takes for n number of elements it will be faster. But how do we sort the array? So, it is faster than linear search, but is it advisable to approach in this way if the array is not sort, is it advisable to sort the array and then apply binary search? In order to understand this, we have to first see how an array can be sorted, how an array can be arranged in either increasing or decreasing order.

That is also a very important problem that is called the sorting problem. And this has got very important I mean application, in many of many of objects whenever I so, for example, for example, you have got the marks of the students, all right? The you have got a marks array hm.

(Refer Slide Time: 19:50)



Say, you have got 2-dimensional matrix may be where, all right, let us take one dimensional one dimensional array; where, the roll numbers are here, every row is corresponding to a roll number. And here I think, I will I will just do it again. I have got the marks here; 56, 50, 65, 70 etcetera, etcetera.

Now, I want to find what is the highest marks. One thing is that I can find that, out or I want to attribute ranks, first second third like that. In that case I can sort this array, and from here I want to have an array like this; where the first element will be 7, the second element will be 65, then 56, then 50. This also sorting the marks. In a decreasing order, and along with that I can also sort the roll numbers, and can publish which roll number got marks right. So, what we want to see next is sorting.

(Refer Slide Time: 21:50)

Sorting: the basic problem

- Given an array
 $x[0], x[1], \dots, x[\text{size}-1]$
reorder entries so that
 $x[0] \leq x[1] \leq \dots \leq x[\text{size}-1]$

2
5
5
6
6
7
9

16

Let us define the basic problem first. Sorting means given an array like 0 to some size minus 1 reorder the entries so that they are in this way; that x_1 is greater than equal to x_0 , x_2 greater than equal to x_1 . So, if it be this sorry, why is this greater than equal to coming up? If for example, 2, 5, 5, 6, 6, 7, 9; this also sorted array. So, both these elements could have taken any order, but since they are same they can be assumed to be sorted. That is why this greater than equal to thing that is coming up.

(Refer Slide Time: 22:45)

Sorting: the basic problem

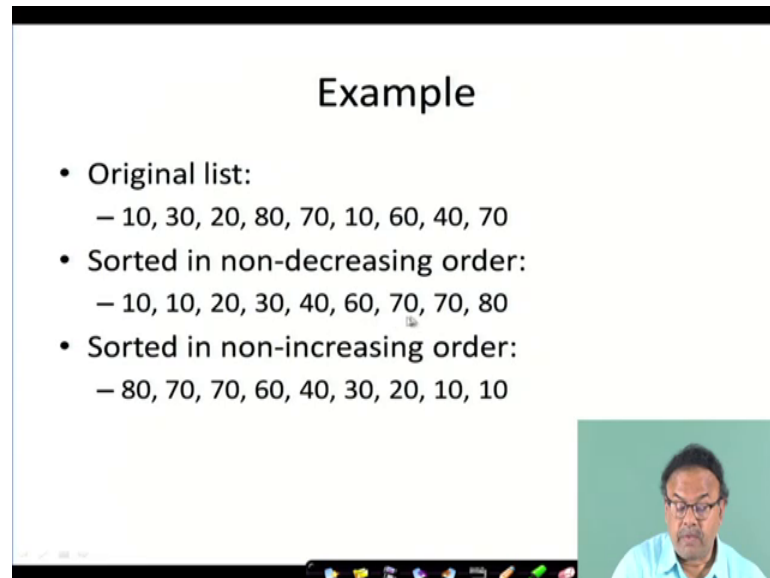
- Given an array
 $x[0], x[1], \dots, x[\text{size}-1]$
reorder entries so that
 $x[0] \leq x[1] \leq \dots \leq x[\text{size}-1]$
 - List is in non-decreasing order.
- We can also sort a list of elements in non-increasing order.

2
5
5
6
6
7
9

16

The list here is in non-decreasing order, non-decreasing. I cannot say increasing order because it is not increasing 2, 2, 5, 5, 6, 6, 7. So, here of course, there was an increase, but here there is no increase. So, this is rather instead of calling it increasing order, we can call it non-decreasing order. A list of elements in non-increasing order it is not decreasing order. So, I can also sort it in the other way as it shown earlier.

(Refer Slide Time: 23:30)



The slide is titled "Example" and contains the following text:

- Original list:
– 10, 30, 20, 80, 70, 10, 60, 40, 70
- Sorted in non-decreasing order:
– 10, 10, 20, 30, 40, 60, 70, 70, 80
- Sorted in non-increasing order:
– 80, 70, 70, 60, 40, 30, 20, 10, 10

The slide also features a small video inset of a man with glasses and a light blue shirt in the bottom right corner, and a Windows taskbar at the bottom.

So, here is an example, this is sorted in non-decreasing order, you can see this array. 10, 30 now this original list. Now if I sort them in non-decreasing order it will 10 minus 10, 10, 20, 30, 40, 60, 70, 72, 70's, 80. And if I do it in a non-increasing order, then minus 80, 70, 70 in this way. This is not increasing in this way, here it is not increasing, all right? Here it is not decreasing if I go in this direction. So, these are the 2 possibilities.

(Refer Slide Time: 24:08)

Sorting Problem

- What we want : Data sorted in order

0 **size-1**

x: Unsorted list

↓

Sorted list

18

So, the sorting problem is, if we have an array from 0 to size minus 1, I want to sort it and get a sorted list, all right?

(Refer Slide Time: 24:25)

Selection Sort

- General situation :

0 **k** **size-1**

x: smallest elements, sorted remainder, unsorted

• **Step :**

-10 10 20 30 | 50 | 32 | 72 | 31

0 **k** **mval** **size-1** **k**

↓ swap ↓

19

That is what I want to have. So, the first algorithm that will be talking about is selection sort. Selection sort is something like this. That suppose we have got a list here and this part of the array is already sorted, this part of the array is sorted in a non-decreasing order. This already the smallest element is here, the smallest element minus 10, then 10, then 20. This part is somehow as sorted. Now am less left with sorting this part only this

remainder. What should I do? I am going to select the candidate for this position, right. Am going to select the candidate for this position. For that what shall I do? I will start since part is sort, all the elements of this side or either equal to this or less than this.

So, I will have to look at this part, and find out minimum element that is here. This element that is minimum, since this part is sorted, this element this element must be greater than this elements. So, among these, only for these people this one is the smallest. So, I select that I select that, and put it in this position I swap how do I do that? Suppose here it was, it was minus 10, 10, 20, 30. And here it was 50, this point, this part is sorted. 50, 32, 72, 31. Now I remember this position, and search from here and find out the minimum. So, this 31 comes I swap it here.

So, what happens to my array? This part, let us see what it will happen. It will now become from here it was 30 it will be 31, 32, 72, 50. So, up to this part it is sorted now. Now form here I will try to find out the minimum element. I find this is the minimum element. And I sort out the minimum element. I swap this minimum element with itself. So, up to this is sorted. Out of these now I have to find out the candidate for this position.

So, I find out the minimum I come here and swap this. So, it becomes 50, 72. So, here in this selection sort the approach is that we have to we have to we are going position by position. And I am selecting the element that belongs to that particular position. So, whenever am starting with the search then at the beginning I am having am starting with the if this be my array.

(Refer Slide Time: 27:57)

Selection Sort

- General situation :

0	k	size-1

smallest elements, sorted	remainder, unsorted
- Step :

0	k	mval	size-1

swap

Then am first searching for this position. And this position will have will be the minimum of all these. I search all through here, and find the minimum suppose the minimum must 10 was here.

So, I take and bring this minus 10 here. So, it becomes minus 10 I know this is the minimum. Now am sorting am now looking for this position. I select the element, minimum element in this zone. I find here something 10. So, this part is done. So now, 10 is swapped here. So, 10, 10 and then this part is unsorted. This part I will sort. In this we way we go on. This is one of the sorting approaches. We will deal with it further in the next lecture.