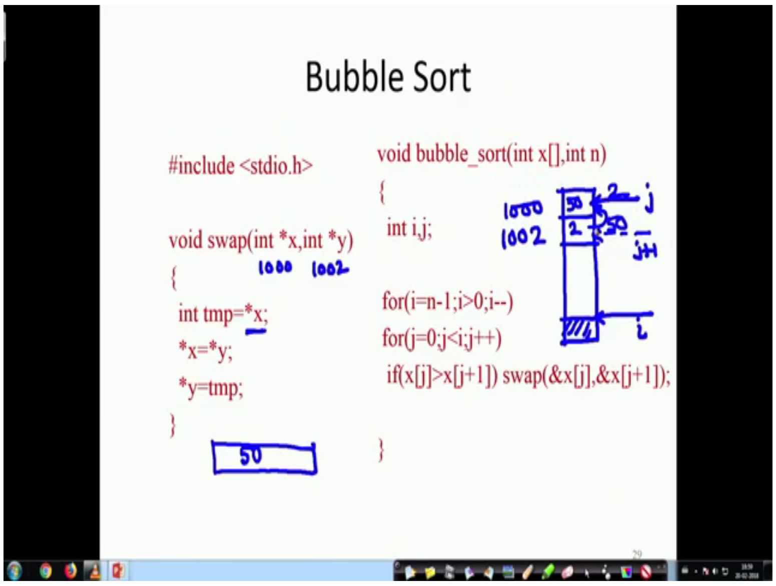
Problem Solving through Programming in C

Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture-47
Use of Pointer in Function: Context Bubble Sort

So, we are discussing about bubble sort and we had looked at some notion of pointers. So, you have now got the preliminary idea about pointers.

(Refer Slide Time: 00:33)



Bubble Sort

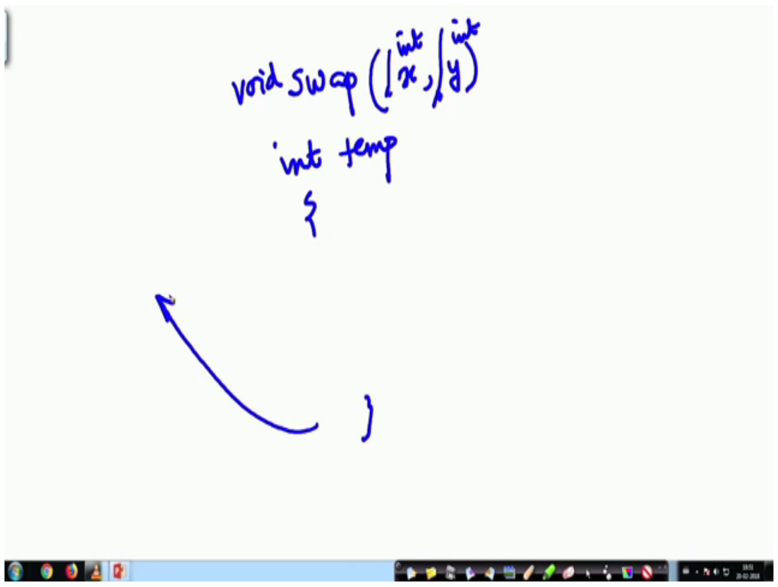
```
#include <stdio.h>

void swap(int *x,int *y)
{
 int tmp=*x;
 *x=*y;
 *y=tmp;
}
```

```
void bubble_sort(int x[],int n)
{
 int i,j;

 for(i=n-1;i>0;i--)
 for(j=0;j<i;j++)
  if(x[j]>x[j+1]) swap(&x[j],&x[j+1]);

}
```

Now let us on this segment we have seen this part of the code, all right this part of the code we have seen this part we have seen right this side this is the bubble sort function on this side, we have got a function which is swapping 2 variables all right I want to swap x and y.

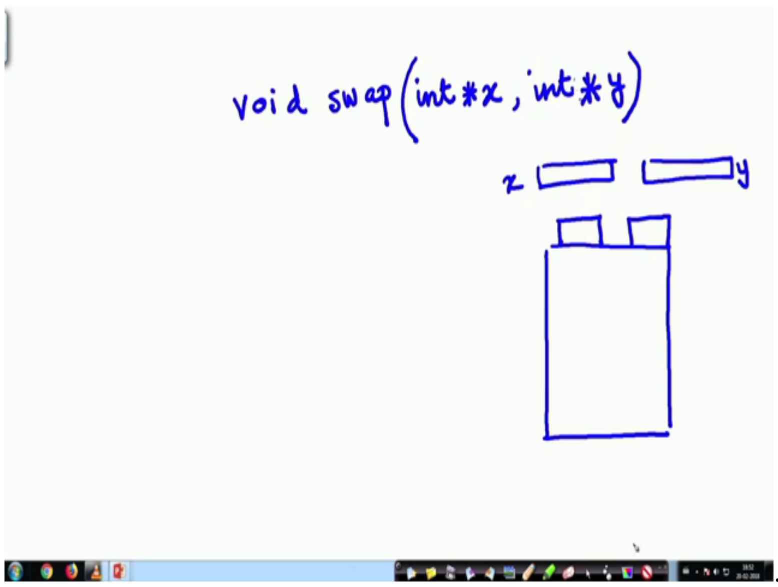
So, this swap function is taking x and y. So, we have seen earlier that in the normal case if we if we swap in a function suppose I have got a void swap xy.

(Refer Slide Time: 01:26)



Then we are shown and we used another variable int temp this was also int this was also int right. So, we had seen that whatever swapping I am doing here that is not being reflected in the main function, that we saw now what we are doing here let us see here I am writing void swap instead of writing x and y I am writing int star x comma int star sorry this we should write space here although not necessary,

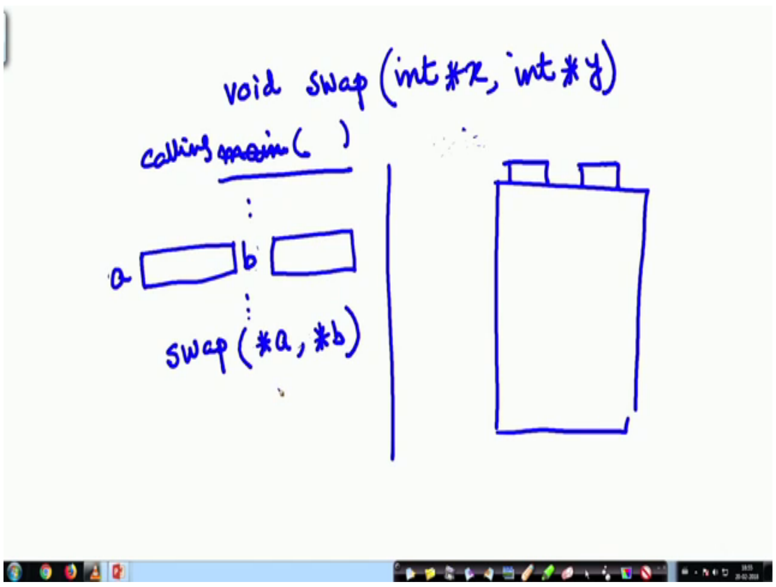
(Refer Slide Time: 01:59)


.

But int star y; that means, what I am passing here is my swap function, here is my swap function and it has got 2 input parameters a function has got always one output, there can be a number of input parameters earlier it was x and y.

So, some variables x and y were having the actual parameters being copied into them, but now I have not used x and y instead I am using pointer to x and pointer to y. So, what is being sent here now is let me do it afresh this faster void swap int star x int star y.

(Refer Slide Time: 03:30)



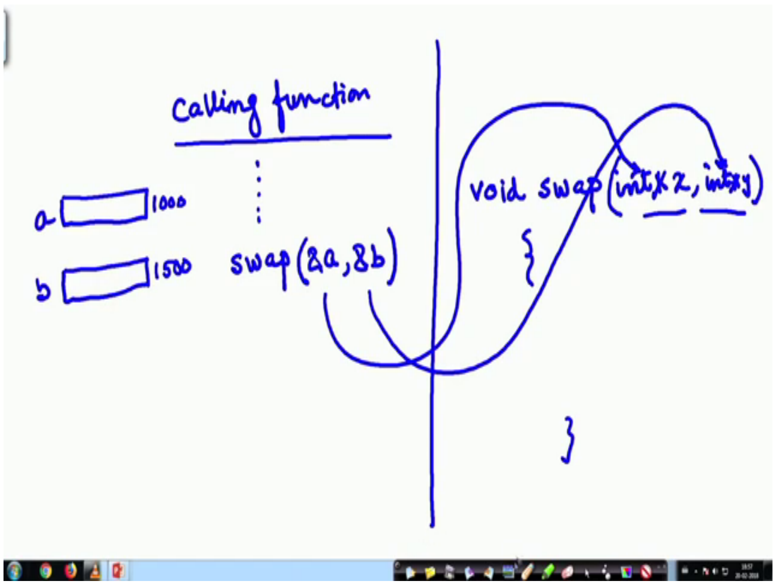
And that means, now I am in my main function here is my main function or the calling function not necessarily the main function in this case it was called from bubble sort. So, the calling function I should not say main it should the calling function the calling function has got 2 variables x and y.

Now when I am calling this function void swap it is here void swap it is already written here. So, I am actually having the function here the swap function I am not passing the values of x and y, let me to be to be more natural let us make this actual parameters are a and b a and b.

So, somehow I am calling here swap in my calling function I am calling swap a comma b. If I had done this then this value of a and the value of b would have been copied there, but instead what I am doing now I will not write this instead I will write int star a and int star b. So, I can I can say here I am passing star a comma star b; that means, what I am passing the pointers let us I will I will do it again let us look at this say here I am passing the address. So, again let me do it in a nicer way.

So, here is the calling function

(Refer Slide Time: 06:09)



And the calling function has got the variables a and b and inside the calling function I am saying swap what should I pass on I instead of saying a and b, I am saying and a comma and b and you know what is and a and b and a means the address of a and b means address of b right. So, I am actually passing the addresses of a and b. So, the address of a was say thousand and this was maybe 1000 500 2 addresses.

Now here is my called function void swap int, now a and b have been declared int star x comma int star y. So, what is happening whatever I am doing here what is being passed here and a is being passed and here and b is being passed; that means, this address and this address are being passed 1000 and 150 are being passed.

And also I know here that what I am getting are the pointers to the variables that had to be swapped it was not int x or int y let us go back to this once again. So, here you see I am calling xj and xj plus 1 in my bubble sort algorithm in my bubble sort algorithm I had the array and j was here and j plus one was here this was already filled something and I am comparing. So, here is my i. So, I will work in this zone and I am looking at two elements j and j plus one and I want to swap them.

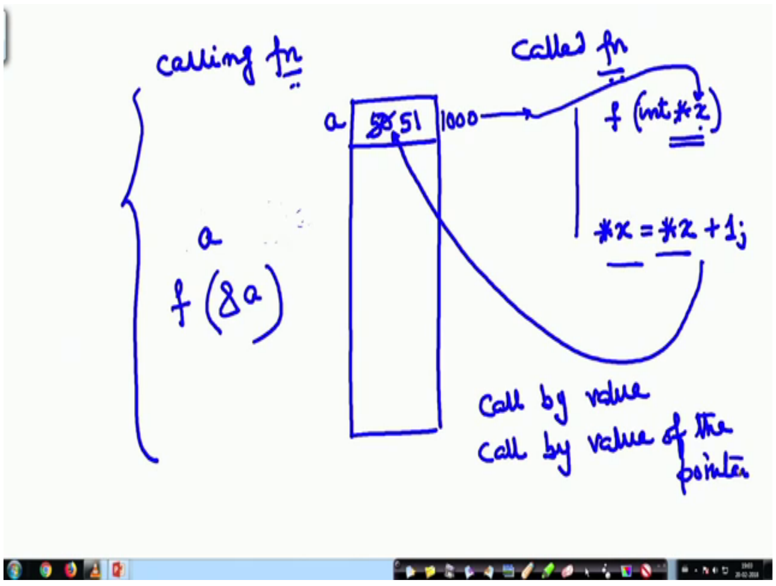
So, what I am doing is I am passing on the address of this here. So, suppose this address was thousand and this address was 1002. So, this is being passed here. So, this is 1000 and this is 1002. Now what is happening here temp is a local variable see temp is not a pointer temp is a local variable suppose here the point was this was 50 and this was 2.

So, temp is getting the content of x right. So, 1000s content thousands content was 50, content of the pointer, content of x all right. So, the 50 is being copied here and then what is happening content of y is coming to content of x. So, content of y is coming to content of x.

So, this is becoming 2 this is being swapped and is becoming 2 and temp is coming as the content of y. So, this is becoming 50. So, what is happening all the changes were reflected here in the main body of the data all right. So, that is how we could really reflect the change on to the calling function.

So, stated in another way it would be like this let us say I have got a data here and here is my calling function and here is my called function.

(Refer Slide Time: 10:32)



Now, there is some variable that is being used by the calling function say a; that means, this a is somewhere here this a has been referred here. So, let me it is not proper to show it here again. So, the called func now the calling function is using the variable a somewhere here alright. So, when this variable is being used as a parameter to this called function instead of sending the variable say some other function I call and in that I pass on the address of that variable a.

So, that address whatever this address is this is 1000 that is being passed on to this called function. So, the called function on the other hand knows that it is input parameter is say called function is f is int what I am getting is int star x; that means, what I am accepting as called function is accepting is nothing, but a pointer to an integer variable.

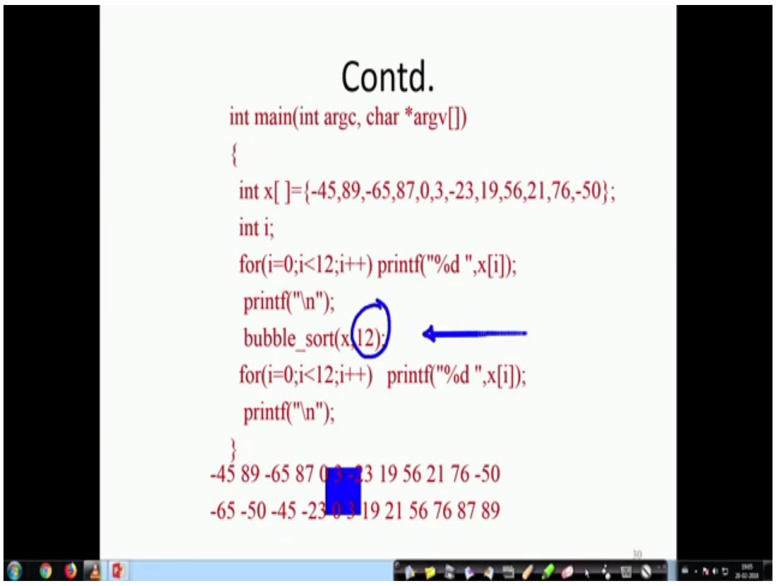
Now so, the pointer to the integer variables the 1000 comes up and here suppose it does something with the content. Now here when I take this say star x; that means, x is a pointer x is a pointer to an integer and when I take star x asterix x; that means, is a content of this and suppose I do it star x plus 1; that means, this the content of this address same address x was 1000 and the content of that 1000 is being added. So, here it was 50. So, that is being added where it is being added here. So, this is becoming changing to 51.

So, this if you recall we had talked about two things when the functions are called we said that usually c always calls by value except for the case of arrays, but here is an example where I can also say this has been called by value of the pointer. So, in this case it is an example call by value of the pointer; that means, what I have copied it is only the pointer. Therefore, the change that is being reflected is being reflected in the main program.

So, in that way it is also a call by reference. So, in that way we can reflect the change of the function. So, if you ever faced the crisis that well I am writing a function, but I know a function can only return 1 value, but I have got multiple values to return this is a nice way in which you can do it ok.

So, this is this we have a new concept we have learnt the use of pointers the notion of pointers and how it can be used in the context of just learning another language there is a bubble sort ok. So, the bubble sort once again we revise let us go to the next one and will come back.

(Refer Slide Time: 14:42)



Contd.

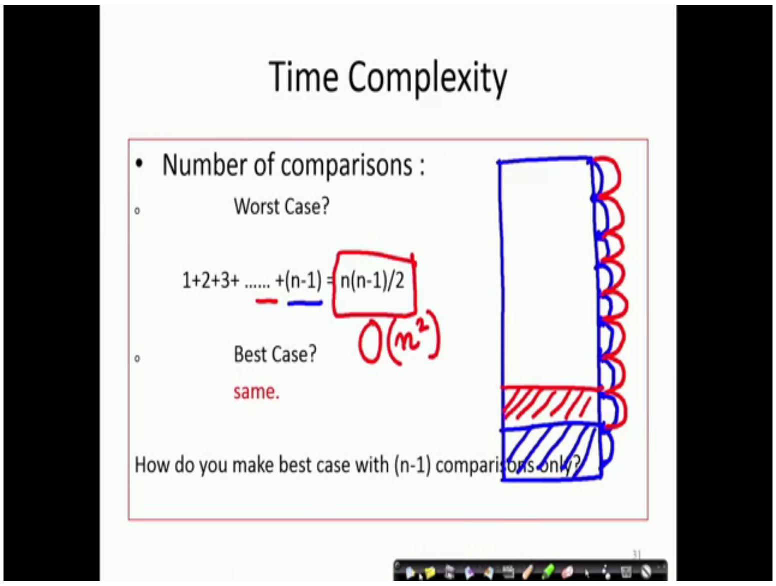
```
int main(int argc, char *argv[])
{
 int x[ ]={-45,89,-65,87,0,3,-23,19,56,21,76,-50};
 int i;
 for(i=0;i<12;i++) printf("%d ",x[i]);
  printf("\n");
  bubble_sort(x,12);
 for(i=0;i<12;i++)  printf("%d ",x[i]);
  printf("\n");
}
```

```
-45 89 -65 87 0 3 -23 19 56 21 76 -50
-65 -50 -45 -23 0 3 19 21 56 76 87 89
```

So, the main program is taking a main function is reading an array all right this array is being read as i is an integer i is an integer. Now for i 0 to less than 12 there were 12 elements here print the array then it is calling bubble sort sorry, when is calling bubble sort it is calling bubble sort with the array being passed and the size of the area being passed here. Now we go to bubble sort; that means, I go to the earlier slide. So, what I had seen earlier.

So, there we go and call the bubble sort and the bubble sort in turn calls, the swap ultimately that swapping is done inside the bubble sort and then we print the ultimate data all right.

(Refer Slide Time: 15:50)



Time Complexity

- Number of comparisons :
  - Worst Case?

    $1+2+3+ \ldots\ldots +(n-1) = \boxed{n(n-1)/2}$

    $O(n^2)$
  - Best Case?

    same.

How do you make best case with (n-1) comparisons only?

So, what will be the time complexity we have seen the time complexity in terms of the big o notation at times. So, what is the worst case if you think of let us once again recapitulate think of the algorithm that for something like this first I had this size n.

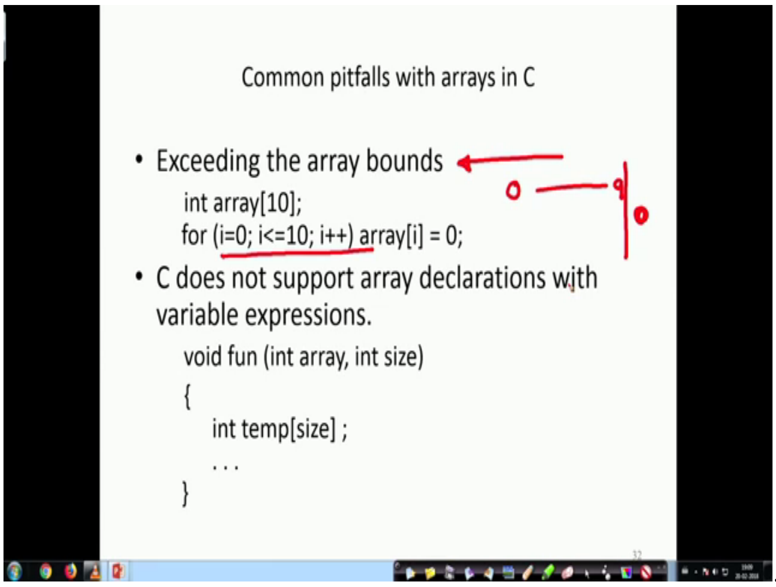
So, initially the number of comparisons were 1, then worst case 2; I first compare between these two, then I compare between these two, then I compare between these two. Now, depending on the comparison I may or may not exchange all right may not show up, but I have to do n minus one comparisons. So, first I do n minus one comparisons and if the along with that I have done the swapping. So, the first position is now filled up with the largest element.

Then again I have to start from here compare again twice thrice in that way I go on comparing the second time and by now this number of comparisons is n minus 2. So, the second one will also be filled up with the heaviest element and in that way I will go on ultimately last time is 1.

So, the overall if I add them the number of comparisons is this and that is of the order of n square, because it breaks down to n square minus n etcetera right. And what is the best case what will be the best case best case is if you do not have to do any swap it is already sorted, but still you do not know that it is sorted you will have to compare them anyway.

And therefore, the best case will be the same all right best case will be the same can you make the best case with n minus 1 comparisons only that we can think of later, but now with this next let us move to so, all these searching And sortings we were doing was using arrays.

(Refer Slide Time: 18:22)



Common pitfalls with arrays in C

- Exceeding the array bounds

```
int array[10];
for (i=0; i<=10; i++) array[i] = 0;
```

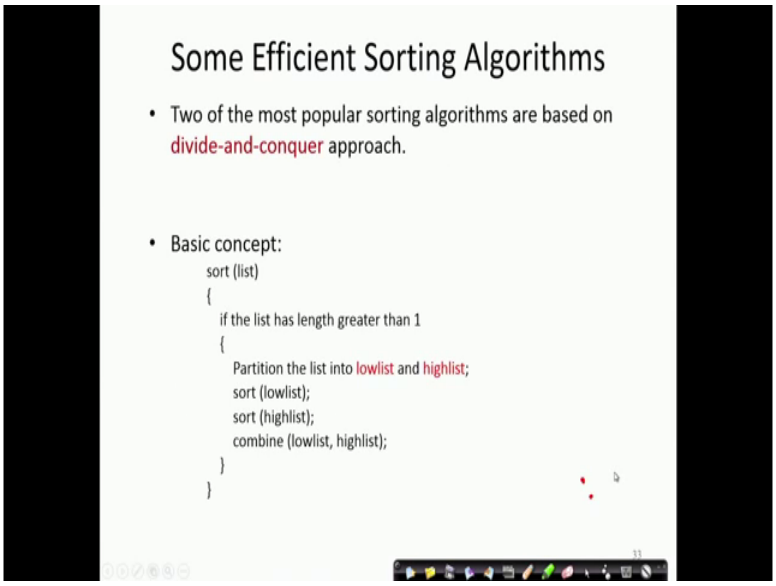
- C does not support array declarations with variable expressions.

```
void fun (int array, int size)
{
    int temp[size] ;
    . . .
}
```

Now, just a word of caution about some common pitfalls with arrays in C is that often we can exceed the array bound without really noting of that. For example, array size has been declared to be 10; that means, my indices are from 0 to 9 right. So, if I write a for loop like this then it will and I try to initialize it.

So, it will initialize to 1 0, then 1, then 2, then 3, till it is less than equal to 10. So, the tenth element is here that will also be put to0, but that may be destroying some other memory location therefore, that check is not there and c does not support array declarations with very variable expressions.

(Refer Slide Time: 19:20)



Some Efficient Sorting Algorithms

- Two of the most popular sorting algorithms are based on divide-and-conquer approach.
- Basic concept:

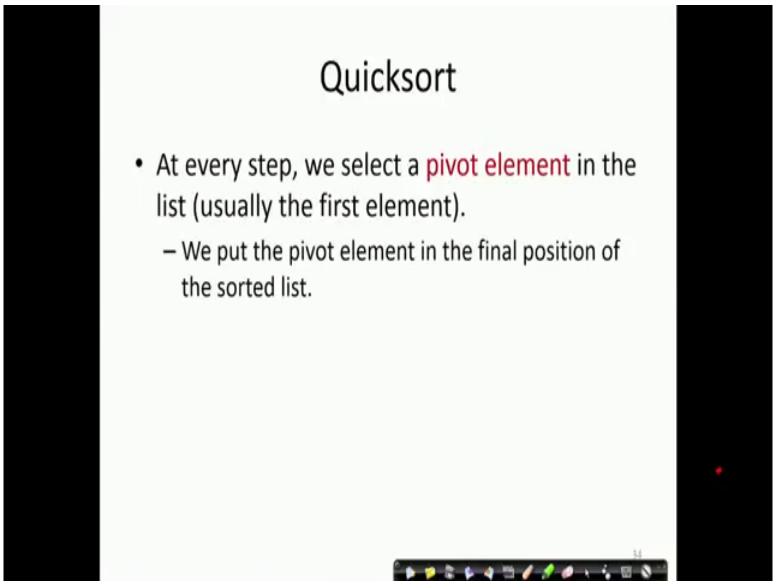
```
sort (list)
{
  if the list has length greater than 1
  {
    Partition the list into lowlist and highlist;
    sort (lowlist);
    sort (highlist);
    combine (lowlist, highlist);
  }
}
```

So, quickly if we summarize what we learned here is let us we are not talking about this in this course, but what we have learned here is searching we have seen and we have seen two very efficient methods of searching one is the linear search is simple and the better method is saw a binary search, but binary search is effective when the array is already sorted.

We have learnt about how to sort an array, we have particularly looked at two sorting algorithms one is selection sort, another is the bubble sort. And while discussing about bubble sort we have also looked at a very interesting way of communicating between a function and a call caller function and the called function, when I have to communicate or pass on return more than 1 variable to the calling function. There we are using the pointers.

So, we introduced the concept of pointers are nothing, but the addresses of the variables. So, instead of passing on the value I am passing on the address and whatever I am doing I am doing on the address ok. So, that is another important thing that we have covered in this lecture next, I think we will go to some other concepts of arrays itself.

(Refer Slide Time: 21:12)

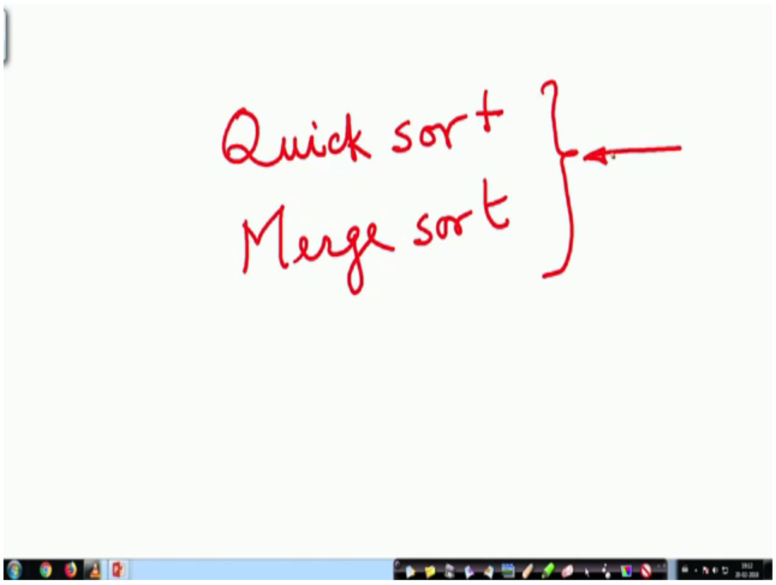


Quicksort

- At every step, we select a pivot element in the list (usually the first element).
  - We put the pivot element in the final position of the sorted list.

Now, just to mention that there are two different two more there are I mean here we have seen that the complexity of these algorithms are of order of n square both for selection sort and for bubble sort, but can we have it better we have got two very nice sorting algorithms; one is known as the quicksort another is known as the merge sort ok.

(Refer Slide Time: 21:41)



Quick sort

Merge sort

Quick sort and Merge sort, which reduces the complexity of sorting and are very popular algorithms, but we are not discussing it in this course for those of you interested can learn it later.

Next we will be moving in the next lecture to discuss about how arrays can handle I mean strings can be handled as arrays that will do in a separate lecture. So, up to this for now and will continue in the next lecture ok.