

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

Lecture - 45
Sorting Methods

So, we were discussing about sorting. And in particular we were discussing about, one sorting technique which is a selection sort. So, in selection sort what do we do we have got an array.

(Refer Slide Time: 00:28)

Selection Sort

- General situation :
x: 0 k size-1
smallest elements, sorted remainder, unsorted
- **Step :**
 - Find smallest element, **mval**, in $x[k..size-1]$
 - Swap smallest element with $x[k]$, then increase k .

0 **k** **mval** **size-1**
[array]
swap

And of size, so, that means, the index is 0 to size minus 1. So, what we do? We some parts we assume that some parts are sorted. So, first thing that we do is find the smallest element. So, if from 0 to k is sorted already in the order. Then my job is to sort from k to size minus 1. So, what we do the step is, that we find the smallest element minimum value in the list x from k to size minus 1, all right. I find the minimum value, and then swap the that minimum value with the k th position; that means, this position, with this position I carry out here in this position I bring the minimum value. And these goes on and then increase k, k will be increased to the next position. And in this way, I will be going on.

(Refer Slide Time: 01:41)

Subproblem

mval?

```
int min_loc (int x[ ], int k, int size)
{
  int j, pos; /* x[pos] is the smallest element found so far */
  pos = k;
  for (j=k+1; j<size; j++)
    if (x[j] < x[pos])
      pos = j;
  return pos;
}
```

find the minimum in between k+1 to size-1

So, the sub problem is the sub problem is to find the minimum element. So, if I go here as we have seen in the, I mean in this array, my task is to find the minimum position ok. So, where is say this part, this part is already sorted, I am sitting here, and this is my k th position, right. This my k th position, and I want to find the minimum where is the minimum value ok. Where is the m value, minimum value where? So, for that, this is the function to find the minimum location in the array x, and starting from this location k, up to the loc up to the location size minus 1.

So, I need to needs know size, I need to need need to know k. That is how we are giving the parameters, as we had explained in functions. So, at every point, you look at how the functions are written that will help you revising this ok. Now the body of the function, x pos will be the smallest element found so far. So, initially I am making pos to be k.

So, x pos is this point, now from k plus 1 to up to size I go on checking, whether this x j this position is less than this position. So, I find out in this way, here I find the minimum right, just see if you can understand the code. And find the minimum in between k plus 1 to size minus 1. In between this am trying to find the minimum. So, whenever I get the minimum am I will be returning that particular position. So, this is the position that I will be returning this position ok. So, what would be my full algorithm therefore? So, I have got a function, I have got this function, min loc, which will return with the minimum location.

(Refer Slide Time: 04:38)

Subproblem

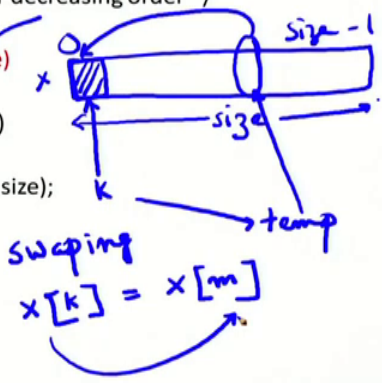
```
/* Yield location of smallest element in x[k .. size-1]; */  
  
int min_loc (int x[], int k, int size)  
{  
    int j, pos; /* x[pos] is the smallest element found so far */  
    pos = k;  
    for (j=k+1; j<size; j++)  
        if (x[j] < x[pos])  
            pos = j;  
    return pos;  
}
```

Now so, it will yield with the location of the smallest element.

(Refer Slide Time: 04:42)

The main sorting function

```
/* Sort x[0..size-1] in non-decreasing order */  
  
int selsort (int x[], int size)  
{ int k, m;  
  for (k=0; k<size-1; k++)  
  {  
      m = min_loc(x, k, size);  
      temp = x[k];  
      x[k] = x[m];  
      x[m] = temp;  
  }  
}
```



So now you see a selection sort, main sorting function will be this. It takes the array and the size ok.

So, this is my array x, and this is the size. So, I will have to work between 0 to size minus 1. I have got 2 intermediate variables k and m. For k equal to 0 to k size minus 1, what am I doing? First am starting with this, this element; that is k equal to 0. And what am I doing? I am finding out from k in the array x up to size I am finding the minimum

location. So, suppose here is the minimum. Then I am what am I doing here? Temp is going to ak. So, this k is going to temp. And this ak is being returned by am this is the minimum is coming here, and then this temp is going here. So, here what am I doing is, I am swapping x my array was internally I called the here there is a small mistake I can find. It should be xk, all this should be x, all right because I am dealing with x.

So, xk will get x minimum m because and vice versa and this y. So, what is happening here let us see.

(Refer Slide Time: 06:55)

The main sorting function

```

/* Sort x[0..size-1] in non-decreasing order */

int selsort (int x[], int size)
{ int k, m;
  for (k=0; k<size-1; k++)
  {
    m = min_loc(x, k, size);
    temp = a[k];
    a[k] = a[m];
    a[m] = temp;
  }
}

```

The diagram illustrates the selection sort process. It shows an array [50, 20, 70, 10, 100] at the top. An arrow points from the value 10 to the index k=1. Below, the array is shown as [10, 20, 70, 50, 100] with a swap between 70 and 50. At the bottom, the array is [10, 20, 50, 70, 100].

Suppose my array was something like this 50, 20, 70, 10, 100. So, I am first this is my k equal to 1. I want to bring the minimum at this position. So, what I do? I from here I start searching for the minimum. I find tenth with the minimum, then I swap here is this piece of code, I swap I bring 10 here, and 20, 70 remains and 50 I take here. And then what do I do? K is incremented so now k is here. Now I find min loc if what is the minimum here compared less than this or not there is none so, fine next.

So, this remains next I increment this. So, this is also in position, this is in position, because it need not be exchanged next my k is increased here at this step, k plus plus it has been increased here. I find the minimum in this area; I find this to be the minimum. So, what is happening? 10 was in place, 20 was in place, I swap 50 and 70 here. So, 50 comes here I find the minimum, 70 remains here, 100 remains here. Now I change my k

and see if there is any minimum left here at the min location is not returning me anything. So, therefore, I get this to be my sorted array ok.

So, this is what so, you can see that so in this way we can, sorry, here is an example you see here.

(Refer Slide Time: 09:07)

The slide titled "Example" illustrates the selection sort algorithm on the array [3, 12, -5, 6, 142, 21, -17, 45]. It shows 10 iterations of finding the minimum element and swapping it with the first element of the current subarray. The minimum element is highlighted in red, and the element being swapped is highlighted in cyan.

Iteration	Array
1	[3, 12, -5, 6, 142, 21, -17, 45]
2	[-17, 12, -5, 6, 142, 21, 3, 45]
3	[-17, -5, 12, 6, 142, 21, 3, 45]
4	[-17, -5, 3, 6, 142, 21, 12, 45]
5	[-17, -5, 3, 6, 142, 21, 12, 45]
6	[-17, -5, 3, 6, 12, 21, 142, 45]
7	[-17, -5, 3, 6, 12, 21, 142, 45]
8	[-17, -5, 3, 6, 12, 21, 142, 45]
9	[-17, -5, 3, 6, 12, 21, 142, 45]
10	[-17, -5, 3, 6, 12, 21, 142, 45]

The example you see here, I start with this. So, 3 is the minimum, I can see here 12 minus 5 6 etcetera I can see, minus 17 is the minimum. So, I find out minus 17 here I swap that. So, minus 17 comes here, and 3 goes to the place of minus 17. So, these 2 now 12, 12 and I find the minimum from 12 onwards, I find minus 5 to be the minimum. So, that one will come here. So, minus 5 comes here, and 12 is swapped there. Now within this part I find the minimum, minimum is 3. So, 3 will be swapped with 12, and 12 goes there in the place of 3 you see. Now so, I start with 6, I start finding the minimum. 6 is the minimum fine. So, it remains I increase k. So now, 142, in this way, I go on from here, who is the minimum for starting from 142? 12 is the minimum.

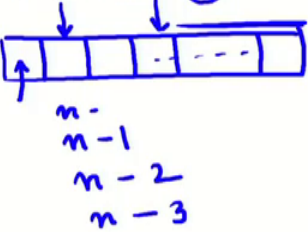
So, 12 and 142 will swapped 12, and 142 has been swapped. Next 21, 21 within this 21, 142, 45, 21 is the minimum. Therefore, I keep it in place. Next is I take 142 and 45, I find that 45 is minimum I swap that, and 142 goes there. Next is 142 is the, that is the sorted array that we get. This is what is known as selection sort. So, the philosophy of selection sort is, that am selecting the minimum element among the unsorted elements

and placing it in the position proper position where that minimum element should be; that is why we are selecting and putting it so, that is why it is known as selection sort.

(Refer Slide Time: 11:27)

Analysis

- How many steps are needed to sort n things ?
 - Total number of steps **proportional** to n^2



The diagram shows a horizontal array of six cells. The first cell has an upward-pointing arrow. The second and third cells have downward-pointing arrows. The fourth cell contains a dashed line. Below the array, the numbers $n-1$, $n-2$, and $n-3$ are handwritten in blue ink, corresponding to the second, third, and fourth cells respectively.

23

So, how many steps are needed to sort n things here? If I do this, how many steps am I taking. You see, how many steps am I taking? All through here if you look at this, how many steps am I taking? If the list size of the list was n then the total number of steps that are required is proportional to n square, because for every element why it is why it is so? Let us try to understand this.

The reason is that for every position, how many positions are there? 1 2 3 like that, there are n positions. Now for every position, for this position, how many comparisons I need to find out the minimum? N minus 1 comparisons n comparisons for all these, next I increment it here. So, how many do I need? I need for each of these for n , for n such elements for each of them I need n comparisons then n plus for this I need n minus 1 comparisons, then I need n minus 2 comparisons, then I need n minus 3 comparisons in this way for example, for this n 1 2 3. So, n minus 4 comparisons I need in this part, I have to compare between these things. So, in that way I go on. So, for every so, for each of them I need to compare this, right for n such things, I have to do this, n n plus n minus 1 n minus times n . So, that will be the order of n square if I find the product, right.

(Refer Slide Time: 13:49)

Analysis

- How many steps are needed to sort n things ?
 - Total number of steps **proportional** to n^2
 - No. of comparisons?
 $(n-1)+(n-2)+\dots+1 = n(n-1)/2$

Of the order of n^2
- Worst Case? Best Case? Average Case?

23

And number of comparison total number of steps and number of comparisons is again, n into n minus 1 by 2 like as I said n minus 1 comparisons plus n minus 2 comparisons, last one is one comparison. So, if you add this one to n minus 1 is n into n minus 1 by 2, you know that, and that is of the order of n square, if I break it up it will be n square minus n by 2.

So, that is again of the order of n square of the order of we write it as of the order of n square all right.

(Refer Slide Time: 14:24)

Insertion Sort

- General situation :

	0		i	size-1
x:	smallest elements, sorted		remainder, unsorted	

Compare and Shift till $x[j]$ is larger.

0 **j** **i** **size-1**

24

Now next we come to another sorting algorithm, which is known as insertion sort. What is the general situation? Again, just like the selection sort, we have got the smallest element sorted; it from 0 to this point is sorted. The remaining is unsorted, all right. Now what we do is we compare and shift till x_i is larger ok, I go on shifting.

So, what I do is I start with this element, and I put this in the proper position, I insert this in the proper position. So, I move this in this side or in this side, and I will come to the proper position of this element, this element is being put since it is smaller the smallest elements is sorted. So, I am taking this, and I am putting it in the place where it should belong because I have not seen up to this.

(Refer Slide Time: 15:42)

Insertion Sorting

```
void InsertSort (int list[], int size)
{
  for (i=1; i<size; i++)
  {
    item = list[i];
    for (j=i-1; (j>=0)&& (list[j] > i); j--)
      list[j+1] = list[j];
    list[j+1] = item;
  }
}
```

The diagram illustrates the insertion of the element 5 into the list [12, 7, 5, 17, 9]. The element 5 is highlighted with a blue box. The list is shifted to the right, resulting in [5, 12, 7, 17, 9].

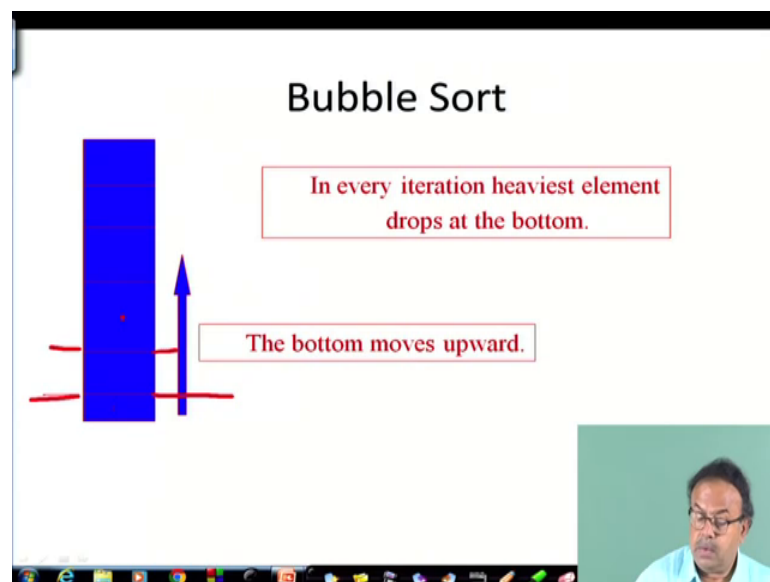
So, so, insert sort is something like this let us see if I can show you a example first.

So, what am doing is, I am comparing and shifting, till x_i is larger. I am continuously shifting the elements, in this way I go on shifting this. Ultimately x_i will be larger than all these elements. So, therefore, my sorted position will be up to this, this is larger right. So, I am sorry, I just went the other side other way. So, let us study this algorithm. This algorithm is void, why it is void? Because it is not returning anything it is taking the array and is sorting this is taking this array and this sorting this. So, it is type is void it is taking a list and it is size. Now for i equal to 1 to size, I am taking item to the list. So, I have got elements like say 12, 7, 5, 17, 9. So, first I am taking i is one so, am taking this element. This is the item. For j s now I am looking at you see j equals i minus 1, and j is

greater than 0 and list. So, it is not am I am not going into this side. I will have to go in these direction, what is this mean? I start with j minus 1, and go till j is greater than equal to 0, and the list is greater than l .

So, all right so, my l is here; that means, I am not I am going till the end of this array on the other side, I am going on the other side. And I shift what was my j here, j plus 1 and the list are being swapped. So now, it will become 7, 12, 5, 17, 9. Now and then in that way I go on shifting, all right. So now, if I come here, I increment l after one item so, I will come 5 and 5. I will go on checking, what is the proper place of 5? So, it will be 5 will be swapped with this 12, 7, 17, 9. In that way I will try to find out every each of these is for proper position. So, I think we can go for a more popular sort.

(Refer Slide Time: 19:07)



Let us go to the more popular sort, I think that will be better selection sort you have learnt insertion sort is another form. But let us learn the more popular one that is the bubble sort. Bubble sort as the name implies. It starts from the bottom, and the minimum element is pushed up to the, how is that done? Just as a bubble floats up from the bottom and goes to the top the minimum element should be pushed up the minimum element should be pushed up to the top.

So, let us see how it happens. So, I first compare between these 2 elements, then these 2 elements, and in this way, I go on comparing between each of them and push it up ok. in every iteration, the heaviest element drops at the bottom. So, what is happening is so, if I

go here from the top, I try to find out if this element, if this particular element, I am sorry this particular element is heavier than this particular element. Then this particular element should go at the top and this should come down here, all right? Next, I compare sorry, next I compare what is happening? Next, I compare between this so, this was shows the heavy element. And I find that this is not an heavy element. It is this heavier than this. So, this may be more heavier if much heavier than this. So, this is not pushed out ok, remains. Now next I compare between this, and every time I am making the same mistake may be this is the heavier, lighter element. So, this lighter element will go up, and this heavier element will come down.

So, it will be something like this, all right. So, in this way in one direction ultimately what I will do? I will have the heaviest element at the bottom, and all the elements here are lighter than this, this element lighter than this element, but these elements are not sorted. So, what I will do? I will restrict my next iteration within this zone, and again start between this elements, and see whichever is heavier that will be pushed down. Am I clear?

You see, every time I am comparing between these and am pushing that down. I think it will be clearer in a moment. Every iteration the heaviest element drops at the bottom and the bottom this part is done and the bottom moves upward. So, next time onwards my search will be restricted between this zone, this zone to the top and the heaviest will come to the next time my search will be restricted to this zone like that. So, the heaviest here, next heaviest here, next heaviest here, heaviest here, next heaviest here, next heaviest here, like that it will go on. So, I want to see if I can give an example first.

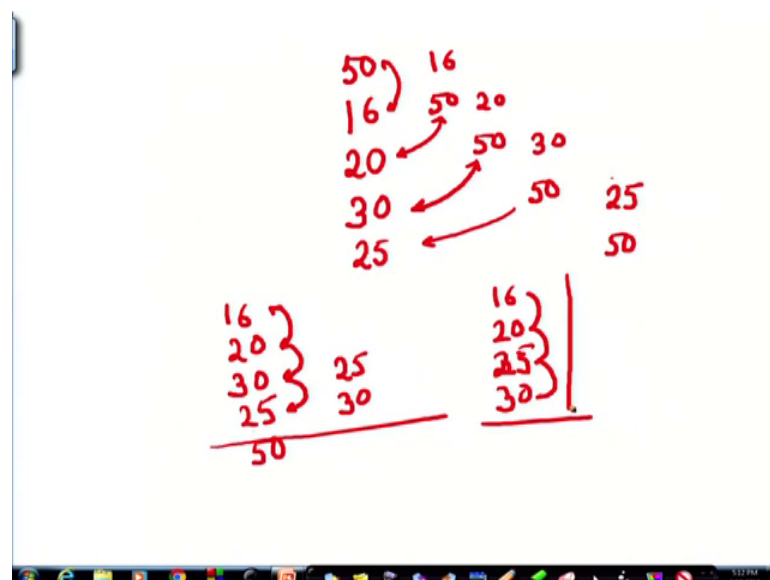
(Refer Slide Time: 23:15)

```
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
```

Yeah, so, what is happening here is, say this was an unsorted array. The heaviest one has been first pushed down this 89 was the heaviest one.

So, by piece by piece comparison I have pushed it down. Next time I was restricted to this part of the array 89 has been pushed down. So and let me try to show you.

(Refer Slide Time: 24:10)

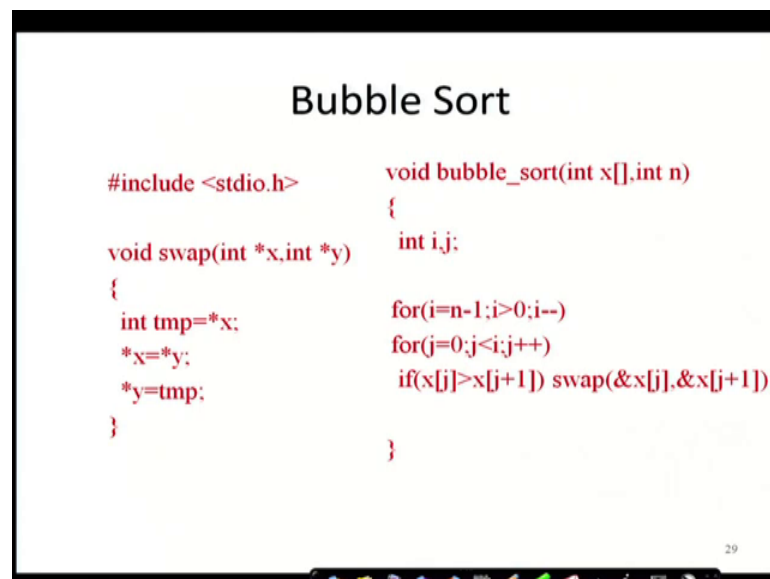


That say, suppose I have got an array like this 50, 16, 20, 30, 40. So, first I compare between these 2, 16 50 is heavier. So, I come here next, I compare between these 2 50 and 20, 50 is heavier. So, 20 comes here 50 comes here. Now 50 and 30, 50 is heavier.

So, 30, 50 and then 50 and 40 so, I compare this. So, it becomes 450. So, in one turn what do I get now I get 16, 20, 30, 40, 50 now it was search the many things are sorted only one was in out of order so, that is been done. But suppose this was let me let me just change it, it becomes became too easy. So, suppose it is 25, then after this swap, this became 25 and 50. So, I have this array. Now I again check between this 2. Heavier is in place, I check between this 2, no issue heavier in place, I check between this 2.

So now what will happen? 25 will come here, and 30 being heavier will come here. And 30 and I need not compare any further. I can because this is this I know is the heaviest. So, what do I have now? 16, 20, 30 not 30, 30 has been swapped 25, 30. I again compare between this 2. So, there is no change, as soon as I find that, there is no interchange; that means, all this things are in order. This is the idea of the selection sorry the bubble sort. Every time the minimum is shifting going to the bottom. Or I could have done it in the other way I could have justified the name of the bubble sort, if I had taken the lowest element, and pushed it up ok. So, quickly if we look at the algorithm a little bit.

(Refer Slide Time: 27:14)



```

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]);
}

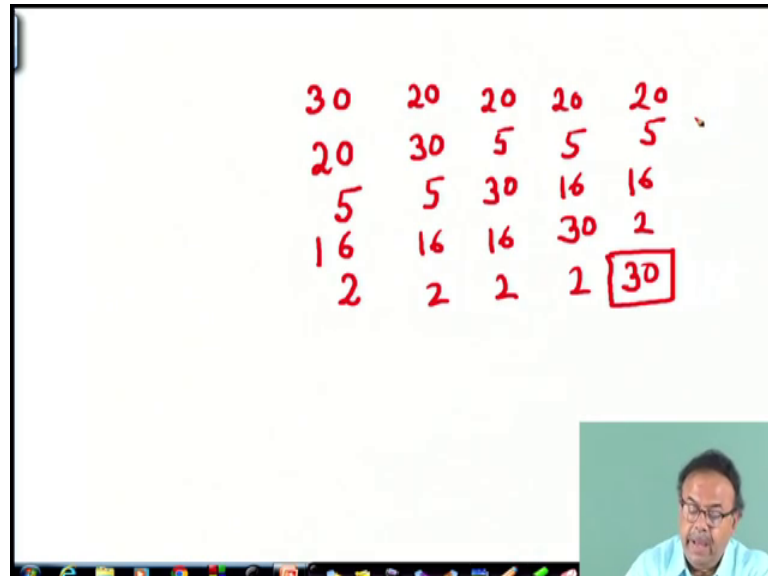
```

29

So, we have got this is a bubble sort algorithm. Let us see, now in order to explain this, I will need sometime, because here we are using some new ideas, star x and star y. I think I will take it up in separate lecture, and discuss this separately, all right. But just as I had explained now, the way the bubble sort works, I will just give you one more example to

show how a bubble sort works. And then in the next lecture, I will show you the algorithm because that will require some more explanation.

(Refer Slide Time: 28:02)



Let us say, I have got this 30, 20, 5, 16, 2. So, what will be hap happening in the first iteration? 30 will be compared with 20, 30 is heavier. So, it would 20, 30, 5, 16, 2. Now 30 will be compared with 5, what will happen and this will be swapped. 5, 30, 16, 2. 30 will be compared with 16. So, what will happen? 16, 30, 2 and 30 is again greater than 2. So, 16, 2, 30. So, 16, 2, 30 by that I get this 30 at the bottom, I now I start with again; 16, 2 right.

(Refer Slide Time: 29:05)

20 5 5 5
5 20 16 16
16 16 20 2
2 2 2 20

30

20

So now, I have got 30 is at the bottom 25, 16, 2. And 30 is already in proper place, here I will swap. So, what will happen? 5, 20, 16, 2 then 20 will and 16 so, it will be 5, 16, 20, 2, then 5, 16, 2, 20, 20 is in the proper place. So, 30 and 20 are in proper place. And 5, 16, 2 I will have to handle.

(Refer Slide Time: 29:44)

5 5
16 2
2 16

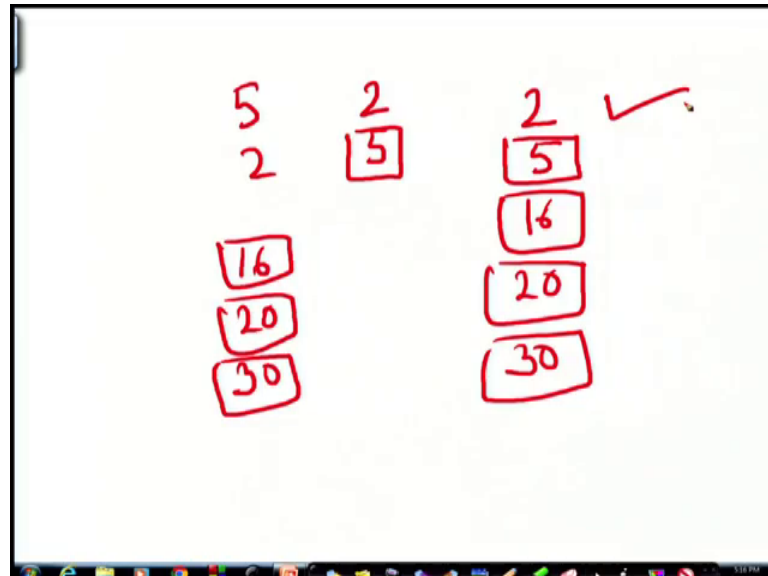
20

30

So, what will be there? 5, 16, 2. So, 5, 16, 2, and here I have got 20, and 30 already in the proper place sort it. So, here 5 and 16 no change, 16 and 2 there will be some change. So,

5, 2, 16 so, 16 is in the proper place. So now, between 5 and 2 so, my thing will be 16, 20, 30 are in place.

(Refer Slide Time: 30:10)



16, 20, 30 are in place, and 5 and 2 I have to deal with, these are fine. So, my pointers are changing, right, between these 2 now 5 is heavier. So, it will be 2, 5, 5 is in the proper place. So, 2 5 in the proper place, 16 proper place, 20 proper place, 30 proper place. So, only 2 there is no question of swapping. So, I get the complete sorted file ok. So, this is the principle of bubble sort, I will be explaining the algorithm the code in the next class. In the meanwhile, you can think of you can read book and you can try to see look at this code and the algorithm yourself. I will discuss it in the next week lecture.

Thank you.