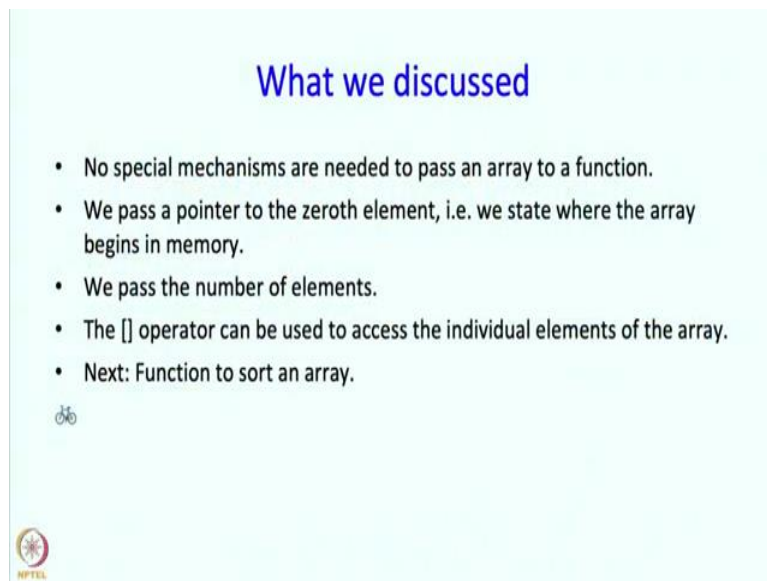


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 16 Part- 4
Array Part-2
A function to sort an array


Welcome back in the previous segment, we saw how an array could be passed to a function.


(Refer Slide Time: 0:23)



What we discussed

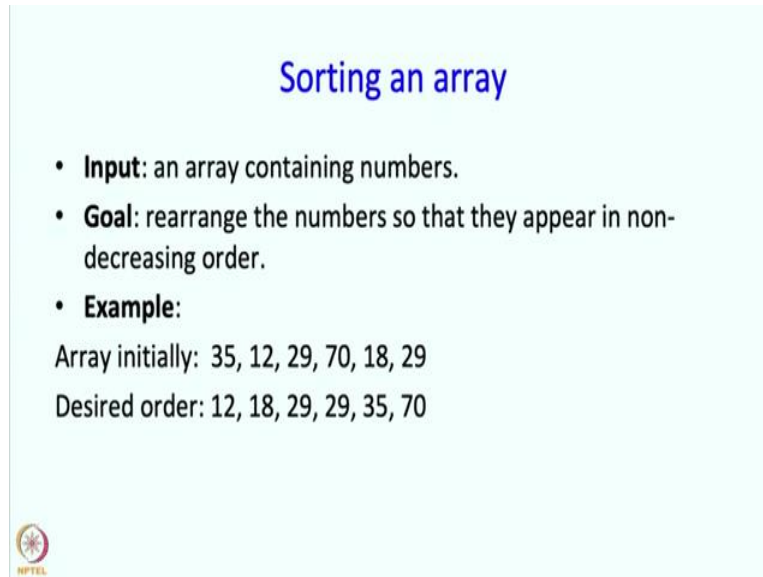
- No special mechanisms are needed to pass an array to a function.
- We pass a pointer to the zeroth element, i.e. we state where the array begins in memory.
- We pass the number of elements.
- The [] operator can be used to access the individual elements of the array.
- Next: Function to sort an array.






Now we are going to use those ideas to build a somewhat more elaborate function. So this function is going to sort an array, okay.

(Refer Slide Time: 0:33)



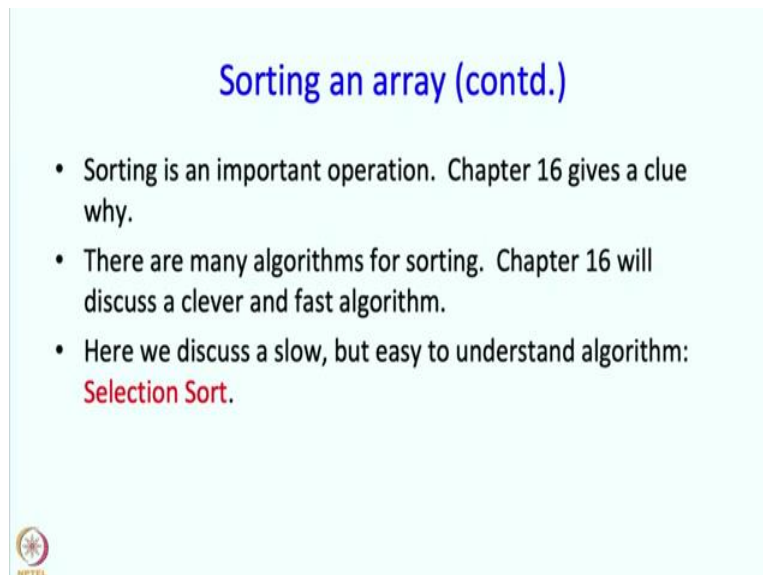
Sorting an array

- **Input:** an array containing numbers.
- **Goal:** rearrange the numbers so that they appear in non-decreasing order.
- **Example:**
Array initially: 35, 12, 29, 70, 18, 29
Desired order: 12, 18, 29, 29, 35, 70




So the input is an array containing numbers. The output, well, the output is going to be present in the same array and you can think of the goal as being to rearrange the numbers so that they appear in non-decreasing order. So as an example, suppose the array initially is 35, 12, 29, 70, 18, 29. Then at the end we want this desired order, so 12, 18, 29, 29, 35, 70.

(Refer Slide Time: 1:02)



Sorting an array (contd.)

- Sorting is an important operation. Chapter 16 gives a clue why.
- There are many algorithms for sorting. Chapter 16 will discuss a clever and fast algorithm.
- Here we discuss a slow, but easy to understand algorithm:
Selection Sort.




Okay. So how do we do this? Well before we say how do we do this, I should point out that is an important operation. And Chapter 16 will give a clue as to why this is the case. So will, will, we are going to do that, so we will just wait until then. There are many algorithms for sorting and Chapter 16 we will discuss a clever and fast algorithm. Here we discuss a slow but an easy to understand algorithm called Selection Sort.

(Refer Slide Time: 1:33)

Selection Sort

Basic idea:

- Find the largest number.
- Exchange it with the element in the last position.
- We have made progress:
 - the last position now contains the largest, as we would like it to.
- Now we can apply the same idea to the first N-1 elements of the array, where N = length of the array.
- Then to first N-2 elements, and so on.




The basic idea of Selection Sort is, find the largest number in the array. Then we exchange it with the element in the last position. Now we have made progress, why? Because the last position now contains the largest number which is really what we wanted the last element of the array to indeed contain the largest and we have placed it there.

So what happens now? We just have to do the same thing for the first N-1 elements. So we apply the same idea to the first N-1 elements of the array, where N is the length of the array then to first N-2 and so on.

(Refer Slide Time: 2:24)

Finding the index of the largest element

```
int posOfMax(float A[], int L){
// Returns the index of the largest element in A.
// Invariant for iteration i:
//     maxIndex = the index of the max in A[0..i-1].
int i=1, maxIndex=0; // invariant holds.
for(i=1; i<L; i++)
    if(A[maxIndex] < A[i]) maxIndex = i;
return maxIndex;
}
```




Okay. So a primitive step in all of this is finding the index of the largest element. Well we said, we want the largest element, but actually it is more useful to find the index of the largest element, where is that largest element present in that array.

(Refer Slide Time: 2:46)

```
Finding the index of the largest element
```

```
int posOfMax(float A[], int L){
// Returns the index of the largest element in A.
// Invariant for iteration i:
//     maxIndex = the index of the max in A[0..i-1].
int i=1, maxIndex=0; // invariant holds.
for(i=1; i<L; i++)
    if(A[maxIndex] < A[i]) maxIndex = i;
return maxIndex;
}
//Number of comparisons: L-1
```



So here is here is a function which does that, so this function takes as argument an array well, an array name, but as you know, it is the starting pointer to the start of the array and the length of the array. But we can think of A as an array name. As we saw that when the parameter passing the values are copied, it really behaves like the name of the array.

This function is going to return the index of the largest element in A. So what is the invariant for this iteration? Well,so for the ith iteration the invariant is that maxIndex which will be a variable, that we will define soon, will equal the index of the largest element in A0 through i-1, okay. Or I should say A[index], if there are multiple it will be one of the, a multiple elements which are maximum, it will be the index of one of those elements.

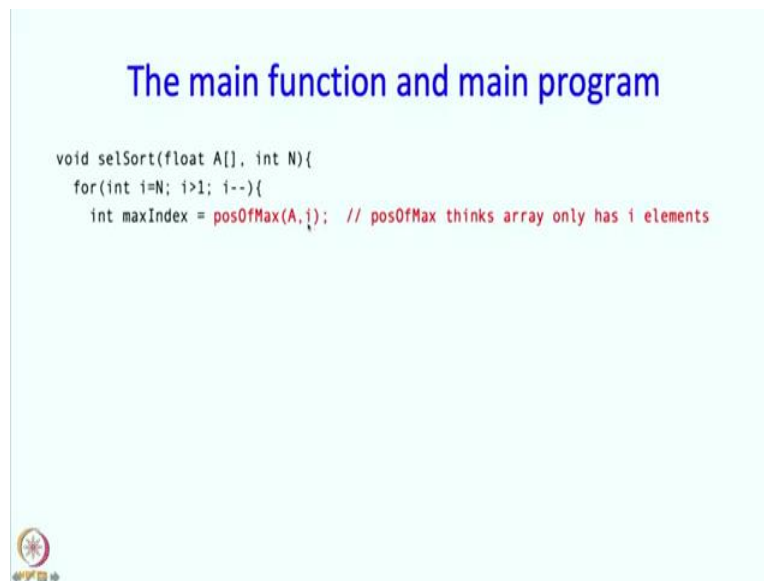
So we start by setting i equal to one and maxIndex equal to 0. So this makes the invariant hold. Why? Because if i equals one, A0 through i-1 is just A0 and maxIndex is indeed the index of the largest element that there is only one element A0 and its index is find index of the largest element in A, kind of trivial, but will do.

Now we are going to increase i until L-1 and as we increase i, we want the invariant to hold. So this is like our Max so far, but instead of keeping track of the Max, we are keeping track of the index. So A[maxIndex] is Max so far and if this is smaller than the new element, then

we are going to set `maxIndex` equal to `i`, okay. So either way `maxIndex` will point to the largest element or be the index of the largest element and so at the end we return `maxIndex`.

I should note that in doing this the number of comparisons needed is going to be $L-1$. So for each value of `i` going from one to $L-1$. So $L-1$ comparisons will be needed, okay. We are, later on going to estimate the time taken for this and this value will be needed over there.

(Refer Slide Time: 5:26)



Okay. So what is the main function and the main program? So the main function is the function selection sort. It again receives the name of the array which is the address of the zeroth element, but which we can pretend is the name of the array and because the name of the array means the same thing and the length of the array. Then we are going to start with N , remember we said that, we are going to place the largest element in the N th position, then the second largest element in the $N-1$ th position and so on, so that is what is loop is for.

We first calculate `maxIndex` which is `posOfMax(A,i)`. If you remember `posOfMax` was going to return an index of the maximum element in the array `A` of length `i`. Now our array `A` actually has length N . In the first iteration this will be N , but in the subsequent iterations this will be smaller. But if you remember according to our discussion in the first iteration, we wanted to find the largest element in the entire array.

In the next iteration the largest element was already in position N . So we wanted to find the largest element in the first $N-1$ positions and that is exactly what this statement will do.


Because in the next iteration, i will have come down one step, so this will be $N-1$. So this is exactly what we want, okay.

So `posOfMax`, this function thinks that the array only has i elements, but that is okay. So it is going to only tell us the index of the largest element in the first i elements, that it is going to consider from this array.

(Refer Slide Time: 7:40)

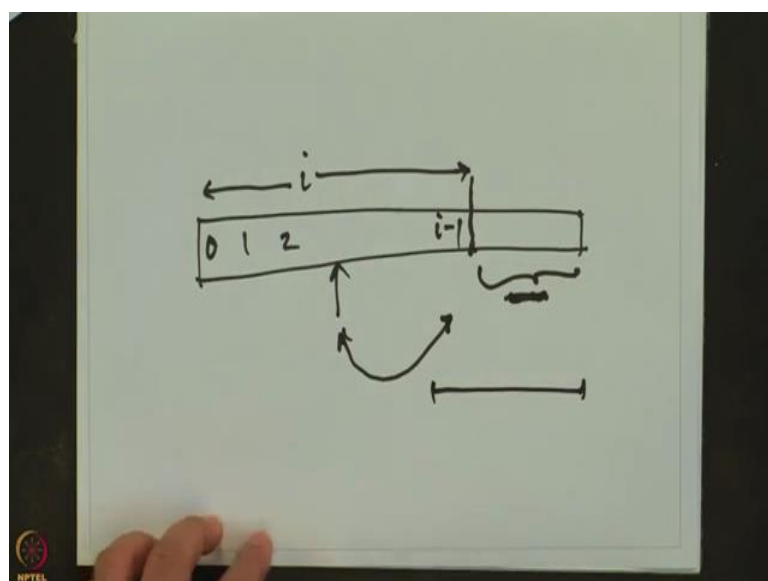
The main function and main program

```
void selSort(float A[], int N){
  for(int i=N; i>1; i--){
    int maxIndex = posOfMax(A,i); // posOfMax thinks array only has i elements
                                // exchange A[i-1], A[maxIndex]
    float maxVal = A[maxIndex];
  }
}
```



Now we want to do the exchange, we want to exchange the $i-1$ th element with `maxIndex` okay, $i-1$ th element.

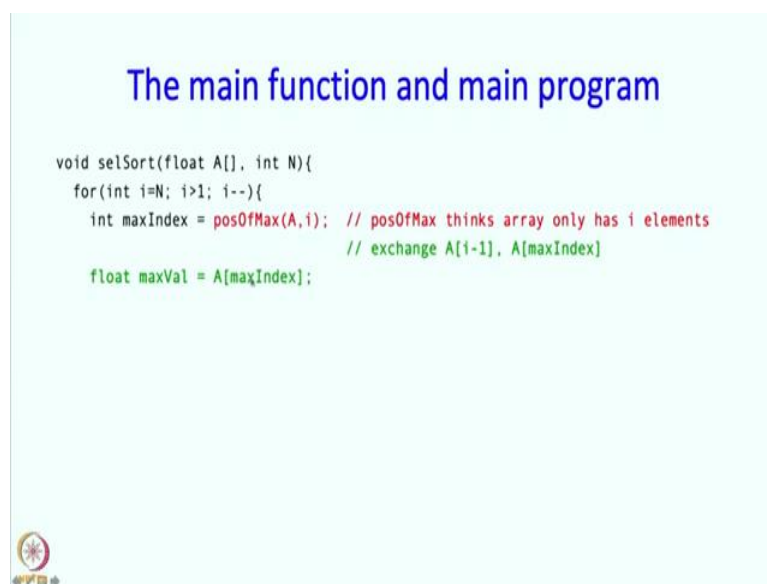
(Refer Slide Time: 7:58)



Well in the i th iteration we are going to consider i elements, okay. So this is element 0, 1, 2 and this last element is $i-1$. So we found some maxIndex over here and we are going to exchange the element over here and the element over here. So what this will do is, we already before starting would have had the correct values over here, okay, the largest values over here. Now the largest value in this entire region is going to be pushed to this point and so the good region will extent a little bit more, that is what this is doing to do.

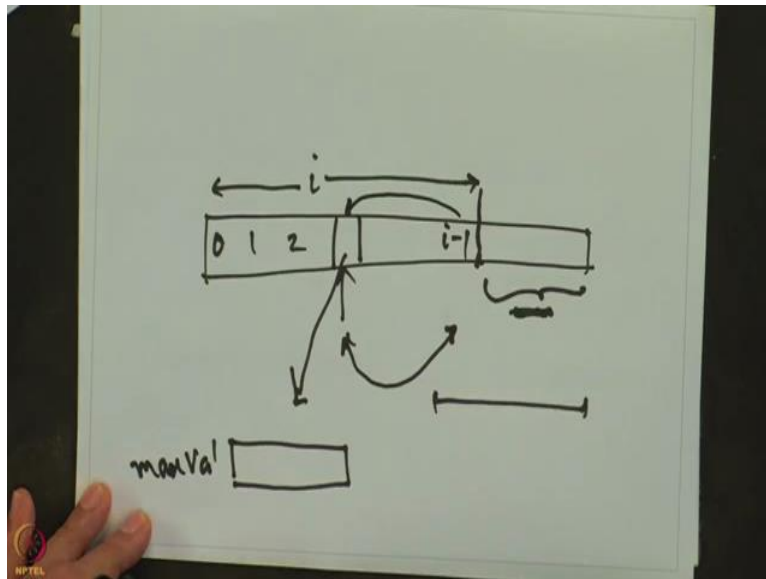
So we need to do this exchange. So how is this exchanged done? Well the exchange is simple enough.

(Refer Slide Time: 8:43)



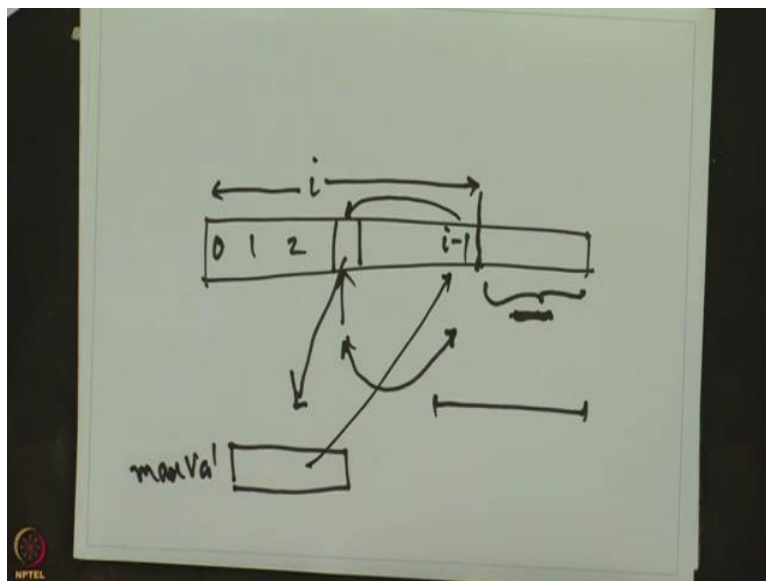
We copy the value at maxIndex to maxVal .

(Refer Slide Time: 8:47)



So this is maxIndex, so this value goes into this variable called maxVal, okay. Then we copy the value in i-1 into this position.

(Refer Slide Time: 9:09)




And finally, we copy the value over here into this position. So the exchange is done okay, so that is it, that is our selection sort.

(Refer Slide Time: 9:20)

The main function and main program

```
void selSort(float A[], int N){
    for(int i=N; i>1; i--){
        int maxIndex = posOfMax(A,i); // posOfMax thinks array only has i elements
                                     // exchange A[i-1], A[maxIndex]
        float maxVal = A[maxIndex];
        A[maxIndex] = A[i-1];
        A[i-1] = maxVal; // exchange done!
    }
}

int main(){
    float a[6] = {35, 12, 29, 70, 18, 29};
    selSort(a, 6);
}
```




So in our main program, we will create this array and then we will call, we will call make us call to sort. Yeah, so we will print it, but I will show that printing step when we do a demonstration.

(Refer Slide Time: 9:36)

Remarks

- posOfMax will perform $L-1$ comparisons, where L = length of array passed to posOfMax.



Okay, all right. So before we do the demonstration, let us review what we have done a little bit. So this is our function and let say, let us try to count how expensive, how much time this whole thing takes. So we said that posOfMax our function will perform $L-1$ comparisons where L is the length of the array passed to posOfMax, okay. So this is what I am talking about.


(Refer Slide Time: 10:11)

The main function and main program

```
void selSort(float A[], int N){
  for(int i=N; i>1; i--){
    int maxIndex = posOfMax(A,i); // posOfMax thinks array only has i elements
                                // exchange A[i-1], A[maxIndex]

    float maxVal = A[maxIndex];
    A[maxIndex] = A[i-1];
    A[i-1] = maxVal;           // exchange done!
  }
}

int main(){
  float a[6] = {35, 12, 29, 70, 18, 29};
  selSort(a, 6);
}
```




So this red part, this red part will require me to do L comparisons, $L-1$ comparisons. If the array has size L . So if the array has size i or if this argument is i , it will do $i-1$ comparisons. The first iteration it will do $n-1$ comparisons, the next iteration it will do $n-2$ comparisons, $n-3$ comparisons and so on, okay.

(Refer Slide Time: 10:41)

Remarks

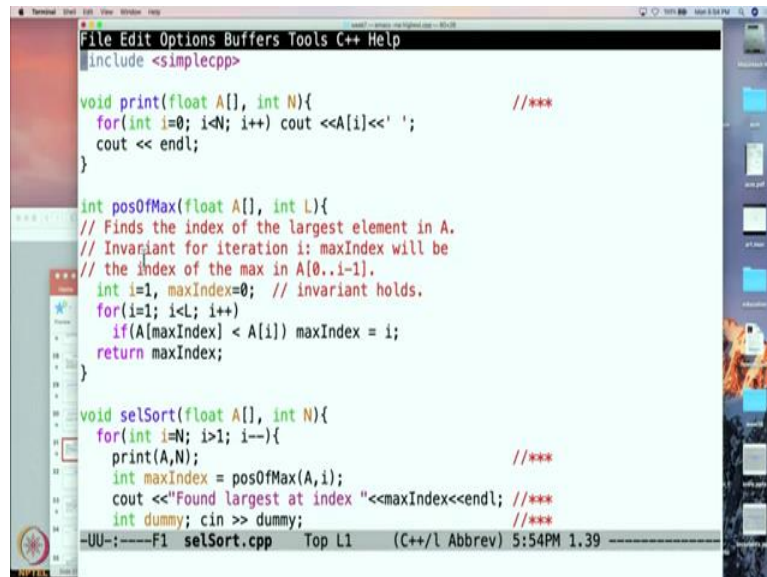
- posOfMax will perform $L-1$ comparisons, where L = length of array passed to posOfMax.
- selSort calls posOfMax for $L = N, N-1, \dots, 2$
- So the number of comparisons =
$$N + N-1 + \dots + 2 = \frac{(N+2)(N-1)}{2} \approx \frac{N^2}{2}$$
- There exist better algorithms which perform far fewer comparisons and thus run faster.
- Demo selSort.cpp



So selSort calls posOfMax for L equals $N, N-1, 2$, so the number of comparisons it does is $N + N-1$ all the way down to 2 so it is $N+2$ times $N-1$ upon 2, okay, so just about N squared by 2 comparisons. So even if I count the number of comparisons, selSort will do about N squared give or take a factor of 2, and N square turns out to be not such a fast algorithm you can do far fewer comparisons and we will see that little a bit later, okay. But anyway, this algorithm

will sort correctly and now we are going to see a demo in which that sorting is going to happen, so let us take a look.

(Refer Slide Time: 11:30)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

void print(float A[], int N){
    for(int i=0; i<N; i++) cout <<A[i]<<' ';
    cout << endl;
}

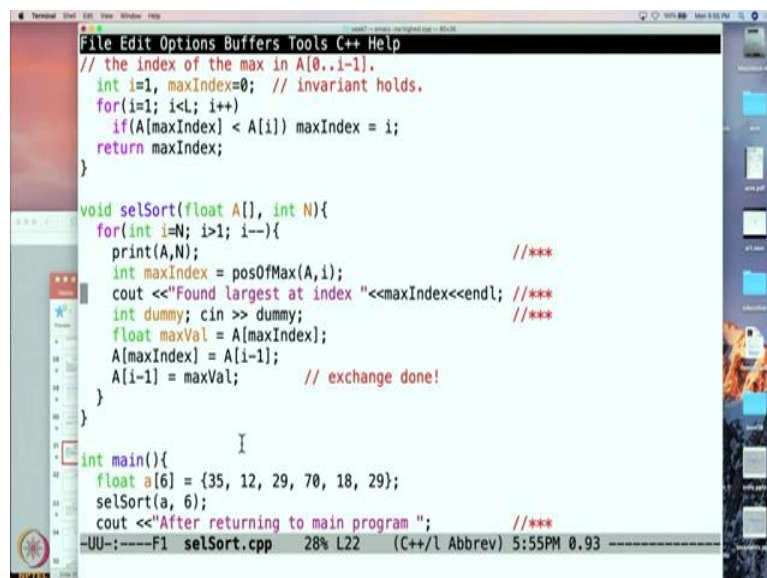
int posOfMax(float A[], int L){
    // Finds the index of the largest element in A.
    // Invariant for iteration i: maxIndex will be
    // the index of the max in A[0..i-1].
    int i=1, maxIndex=0; // invariant holds.
    for(i=1; i<L; i++){
        if(A[maxIndex] < A[i]) maxIndex = i;
    }
    return maxIndex;
}

void selSort(float A[], int N){
    for(int i=N; i>1; i--){
        print(A,N);
        int maxIndex = posOfMax(A,i);
        cout <<"Found largest at index "<<maxIndex<<endl;
        int dummy; cin >> dummy;
    }
}

-UU-:----F1 selSort.cpp Top L1 (C++/L Abbrev) 5:54PM 1.39
```

So this is our function, okay. So this is our code for posMax, I have added a print statement because I would like to print the array at different points, so we will see there this is going to be used.

(Refer Slide Time: 11:49)



```
File Edit Options Buffers Tools C++ Help
// the index of the max in A[0..i-1].
int i=1, maxIndex=0; // invariant holds.
for(i=1; i<L; i++){
    if(A[maxIndex] < A[i]) maxIndex = i;
}
return maxIndex;
}

void selSort(float A[], int N){
    for(int i=N; i>1; i--){
        print(A,N);
        int maxIndex = posOfMax(A,i);
        cout <<"Found largest at index "<<maxIndex<<endl;
        int dummy; cin >> dummy;
        float maxVal = A[maxIndex];
        A[maxIndex] = A[i-1];
        A[i-1] = maxVal; // exchange done!
    }
}

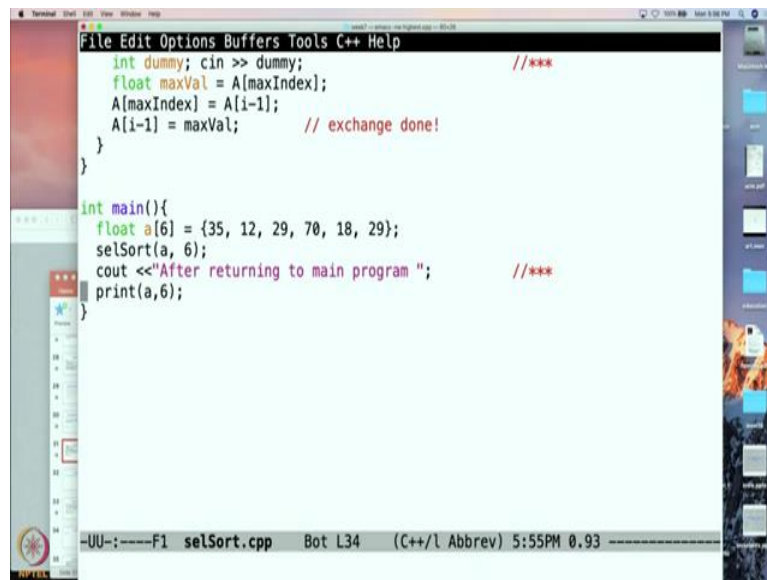
int main(){
    float a[6] = {35, 12, 29, 70, 18, 29};
    selSort(a, 6);
    cout <<"After returning to main program ";
}

-UU-:----F1 selSort.cpp 28% L22 (C++/L Abbrev) 5:55PM 0.93
```

So let us just take a look at the selSort function, okay. So I have modified it a little bit and the modifications are store, shown by these stars. So what I am doing is at the very beginning, at the beginning of the each iteration, I am going to print out the entire array. So this way you

can see how the array is going to change and in, to make sure that, okay, so then this posOfMax is going to be called and I am also going to tell you where that index is, what index was reported. So again, we can check how posOfMax is working, how the entire thing is working. And I do not want the iterations to go of very fast. So here I am going to just have a dummy variable, okay, and I am going to read a number into it, I am not going to use this, but this will just force C++ to stop at this point and wait for me to type in something so that I can continue. So the value of the dummy variable is not important over here. I will just put in so that I can force C++ to stop, okay? So the rest of the code is exactly the same and at the end also, I am going to print out a message and I am going to print out the value of the array, okay.

(Refer Slide Time: 13:10)



```
File Edit Options Buffers Tools C++ Help
int dummy; cin >> dummy; //***
float maxVal = A[maxIndex];
A[maxIndex] = A[i-1];
A[i-1] = maxVal; // exchange done!
}

int main(){
float a[6] = {35, 12, 29, 70, 18, 29};
selSort(a, 6);
cout <<"After returning to main program "; //***
print(a,6);
}
```

---UU---:-----F1 selSort.cpp Bot L34 (C++/L Abbrev) 5:55PM 0.93

So we had the print function at the top and that simply prints out the value of the array, value of all the elements in the array. So let us execute, compile and execute this. So let us run it.

(Refer Slide Time: 13:20)

```
190060037 91
180030001 88
190370051 91
180020041 66
190010045 44
190030101 83
190120077 91
~/Desktop/npTEL/week7 : ./a.out
190370051
91
190380051
Roll number not found.
-1
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ 3poly.cpp
+ g++ 3poly.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : █
```

```
190010045 44
190030101 83
190120077 91
~/Desktop/npTEL/week7 : ./a.out
190370051
91
190380051
Roll number not found.
-1
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ 3poly.cpp
+ g++ 3poly.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ selSort.cpp █
```

```
190380051
Roll number not found.
-1
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ 3poly.cpp
+ g++ 3poly.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

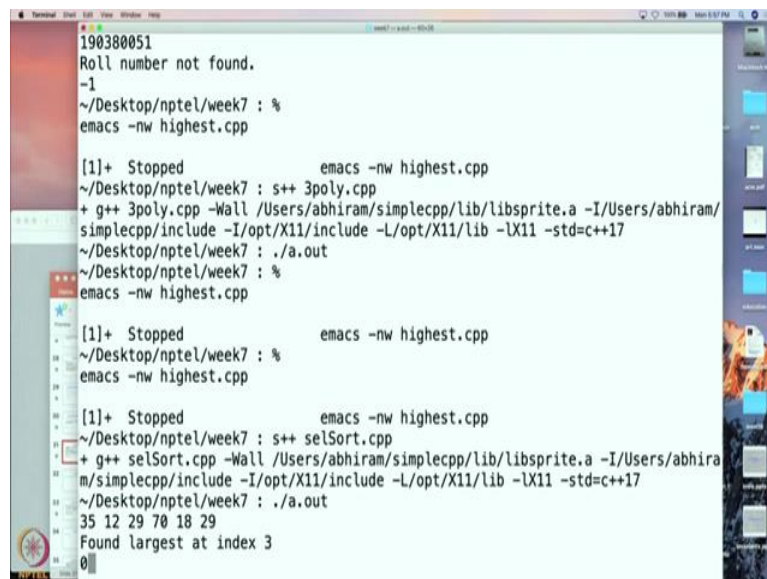
[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
35 12 29 10 18 29
Found largest at index 3
```

So this is the printout at the beginning of the first iteration of selSort. And then it, so this is the, at the at the time when nothing has been done, we are just beginning the very first iteration. And we have issued Max pos and it says that the maximum value is found in index three. Let us check if that is true?

So 0, 1, 2, 3, 70 is at index three and it is indeed the maximum value, so Max pos has behaved correctly. So let us just type in some nonsense value so that the program will continue.

(Refer Slide Time: 14:11)



```
190380051
Roll number not found.
-1
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

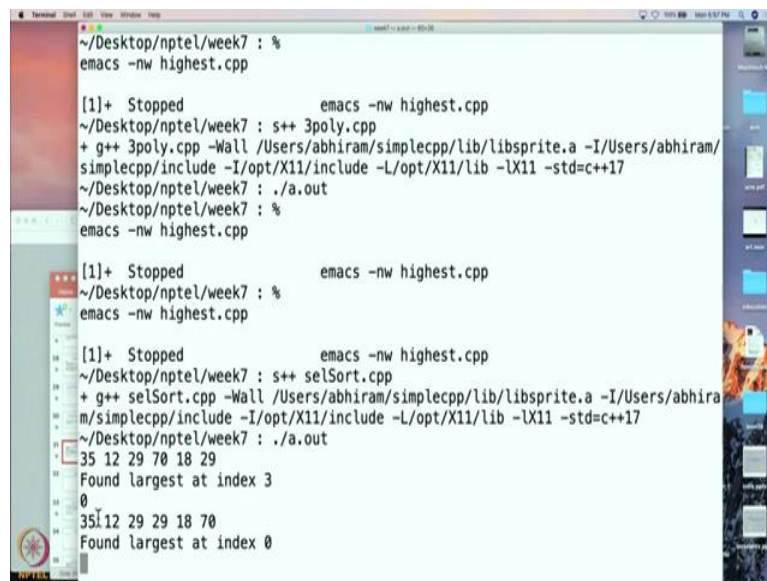
[1]+  Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ 3poly.cpp
+ g++ 3poly.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+  Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+  Stopped                  emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
35 12 29 70 18 29
Found largest at index 3
0
```

So now you see that the second iteration is being executed and at this point 70 has already gone to the end because of our exchange and we have also issued the call to one more Max pos. And that is limited to this region. And in this Max pos is found at index zero which is indeed true. So again let us continue okay.

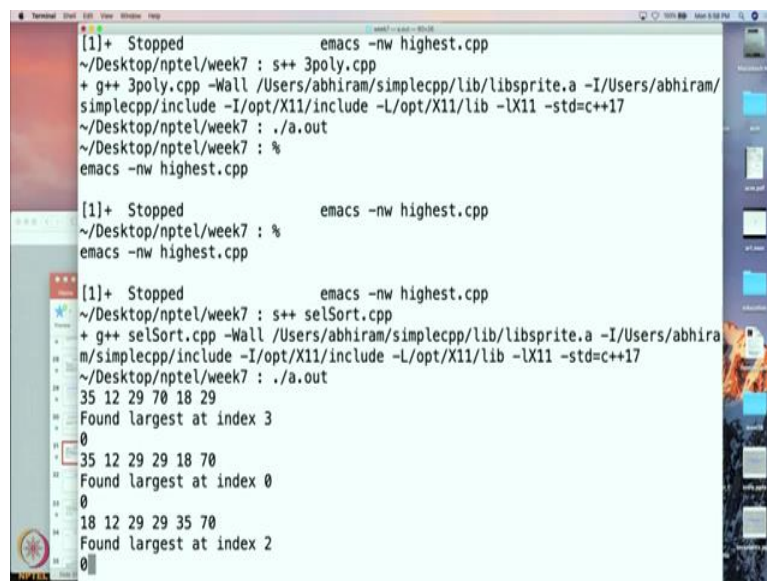
(Refer Slide Time: 14:39)



```
~/Desktop/npTEL/week7 : %  
emacs -nw highest.cpp  
  
[1]+ Stopped          emacs -nw highest.cpp  
~/Desktop/npTEL/week7 : s++ 3poly.cpp  
+ g++ 3poly.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/  
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17  
~/Desktop/npTEL/week7 : ./a.out  
~/Desktop/npTEL/week7 : %  
emacs -nw highest.cpp  
  
[1]+ Stopped          emacs -nw highest.cpp  
~/Desktop/npTEL/week7 : %  
emacs -nw highest.cpp  
  
[1]+ Stopped          emacs -nw highest.cpp  
~/Desktop/npTEL/week7 : s++ selSort.cpp  
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira  
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17  
~/Desktop/npTEL/week7 : ./a.out  
35 12 29 70 18 29  
Found largest at index 3  
0  
35 12 29 29 18 70  
Found largest at index 0
```

And now 35 has gone till the end and now we are finding that Max pos is found at index two, 0, 1, 2. Yeah there are two 29's, but it has picked one of those and so, and it is indeed at this position there is indeed the maximum, A maximum value.

(Refer Slide Time: 15:01)



```
[1]+ Stopped          emacs -nw highest.cpp  
~/Desktop/npTEL/week7 : s++ 3poly.cpp  
+ g++ 3poly.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/  
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17  
~/Desktop/npTEL/week7 : ./a.out  
~/Desktop/npTEL/week7 : %  
emacs -nw highest.cpp  
  
[1]+ Stopped          emacs -nw highest.cpp  
~/Desktop/npTEL/week7 : %  
emacs -nw highest.cpp  
  
[1]+ Stopped          emacs -nw highest.cpp  
~/Desktop/npTEL/week7 : s++ selSort.cpp  
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira  
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17  
~/Desktop/npTEL/week7 : ./a.out  
35 12 29 70 18 29  
Found largest at index 3  
0  
35 12 29 29 18 70  
Found largest at index 0  
0  
18 12 29 29 35 70  
Found largest at index 2  
0
```

```
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week7 : ./a.out
~/Desktop/nptel/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week7 : ./a.out
35 12 29 70 18 29
Found largest at index 3
0
35 12 29 29 18 70
Found largest at index 0
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 2
```

So let us again making go forward, so this time 29 has gone till the end. Well we do not know whether it was exchange or not, but presumably there was an exchange. But now again, a maximum is found at this position, okay.

(Refer Slide Time: 15:18)

```
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week7 : ./a.out
~/Desktop/nptel/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week7 : ./a.out
35 12 29 70 18 29
Found largest at index 3
0
35 12 29 29 18 70
Found largest at index 0
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 2
```



```
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week7 : ./a.out
35 12 29 70 18 29
Found largest at index 3
0
35 12 29 29 18 70
Found largest at index 0
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 0
```

So let us again type 0, so it has gone forward and this time it is really searching within this region Max pos is being found within this region. So the largest value is over here and so index zero is returned by Max pos and so at this, sorry, let me type 0.

(Refer Slide Time: 15:36)

```
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : %
emacs -nw highest.cpp

[1]+ Stopped                  emacs -nw highest.cpp
~/Desktop/nptel/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhira
m/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week7 : ./a.out
35 12 29 70 18 29
Found largest at index 3
0
35 12 29 29 18 70
Found largest at index 0
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 0
^[[C
```

```
Terminal Shell Edit View Window Help
emacs -nw highest.cpp

[1]+ Stopped emacs -nw highest.cpp
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
35 12 29 70 18 29
Found largest at index 3
0
35 12 29 29 18 70
Found largest at index 0
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 0
```

(Refer Slide Time: 15:40)

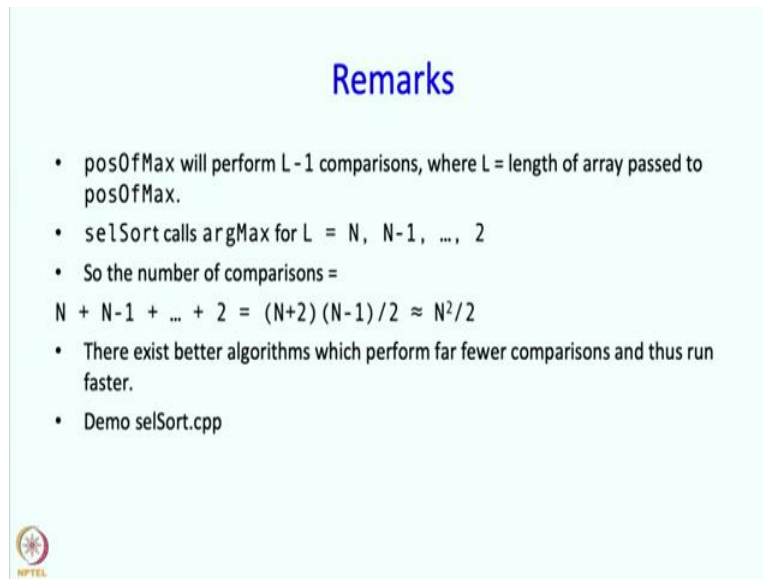
```
Terminal Shell Edit View Window Help
emacs -nw highest.cpp

[1]+ Stopped emacs -nw highest.cpp
~/Desktop/npTEL/week7 : %
emacs -nw highest.cpp

[1]+ Stopped emacs -nw highest.cpp
~/Desktop/npTEL/week7 : s++ selSort.cpp
+ g++ selSort.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week7 : ./a.out
35 12 29 70 18 29
Found largest at index 3
0
35 12 29 29 18 70
Found largest at index 0
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 2
0
18 12 29 29 35 70
Found largest at index 0
After returning to main program 12 18 29 29 35 70
~/Desktop/npTEL/week7 :
```


So at this point the function is returning at after returning to main program 12, 18, 29, 29, 35, 70 is printed and it is indeed correctly sorted.

(Refer Slide Time: 15:58)



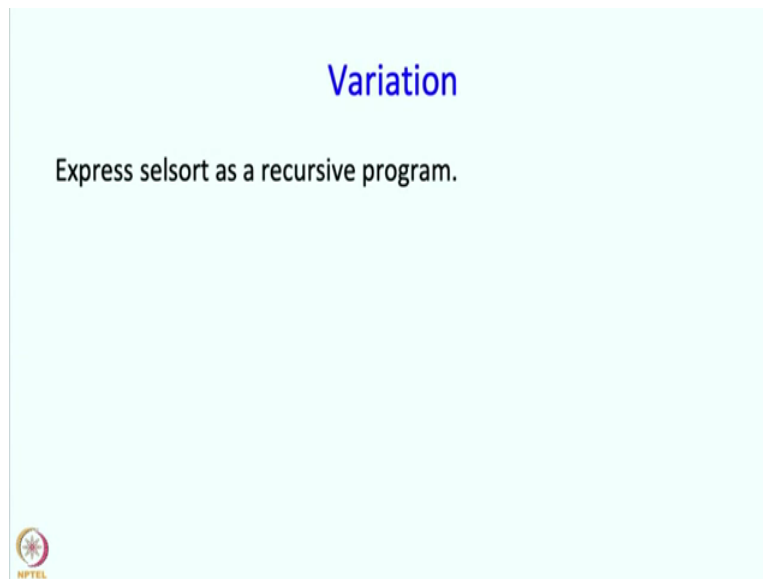
Remarks

- posOfMax will perform $L - 1$ comparisons, where $L =$ length of array passed to posOfMax.
- selSort calls argMax for $L = N, N - 1, \dots, 2$
- So the number of comparisons =
$$N + N - 1 + \dots + 2 = \frac{(N + 2)(N - 1)}{2} \approx \frac{N^2}{2}$$
- There exist better algorithms which perform far fewer comparisons and thus run faster.
- Demo selSort.cpp




Alright. So posOfMax will perform $L - 1$ comparisons where L is the length of the array passed to posOfMax, okay. So we saw all these things, okay.

(Refer Slide Time: 16:10)



Variation

Express selsort as a recursive program.




We saw that demo and we can do a variation. We can express selsort as a recursive program.

(Refer Slide Time: 16:18)

Recursive selsort

```
void RselSort(float A[], int N){
    if(N == 1) return; // base case
    int maxIndex = posOfMax(A,N);
    float maxVal = A[maxIndex];
    A[maxIndex] = A[N-1];
    A[N-1] = maxVal;    // exchange
    RselSort(A, N-1);
}
```



So I am going to leave this, I am going to show you the code, but I am going to leave this as an exercise for you, okay.

(Refer Slide Time: 16:28)

Demo

RselSort.cpp




In fact, I am going to put out the code as well and you can run it yourself and check it. So that essentially concludes this lecture. And so I would like to make some remarks.

(Refer Slide Time: 16:47)

Concluding remarks

- The name of an array can be used in a program without the index, its value is the address of the 0th element of the array.
- Indexing into an array is to be thought of as an operation [] between the array name and the index.
- We can “pass” an array to a function by passing the array name. This does not copy the values but merely copies the address of the 0th element. In addition the length of the array must also be passed.
- Writing functions on arrays is useful, and this is a skill you must master.
- Selection sort runs in time proportional to the square of the number of elements being sorted. Faster algorithms are discussed in Chapter 16.



And in this lecture we saw what sort of happens behind the scenes when we access arrays. And we said what does an array name mean? And what happens for the indexing operator? And how do you pass an array to a function. And we said that when you pass an array to a function, it does not copy values, but it merely copies the address of the zeroth element. And if you want to access the entire, if you want the function to access the entire array you should tell the function how many elements the array has, so that also must be passed.

Writing functions on array is useful and this is definitely a skill that you must master. Selection sort runs in time proportional to the square of the number of elements being sorted and in Chapter 16 we will discuss faster algorithms. I will stop here, but I will urge you to solve the problems at the end of Chapter 14 and of course, read, read Chapter 14. Thank you.