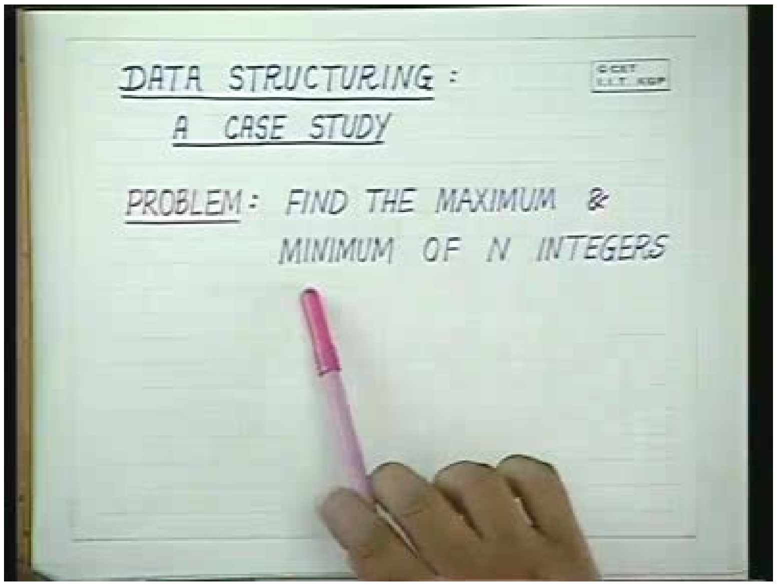
Programming and Data Structure
Dr. P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture # 05
Data Structuring: Case Study - I

Good morning everybody. And today we will start our discussion which is related to data structuring. Till date what we have done is we have just introduce certain constructs of C language and seen how we can do some simple programming using C language. Today we will take case study of a very simple problem and see how data structuring helps to solve the problem. So the problem that we will take today is a very ordering one but we will see that there are several issues in which we shall consider to solve this problem to the best possible way in which we can do it.

(Refer Slide Time 01:52 min)



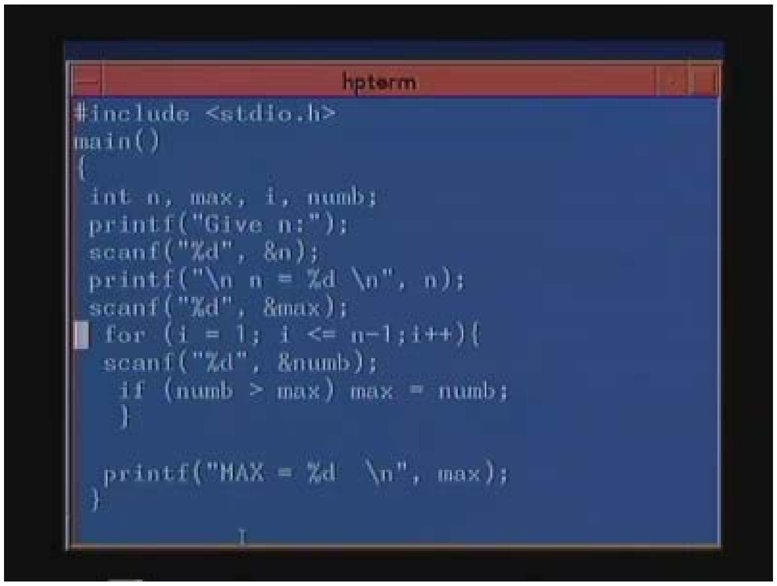
DATA STRUCTURING :
A CASE STUDY

PROBLEM: FIND THE MAXIMUM & MINIMUM OF N INTEGERS

The problem is to find the maximum and minimum of a set of n integers. That is you are given to read a value N and then reading N integers and what you are supposed to produce as output is the maximum and the minimum value. Now let us see how we go about it. There are various places where you can start analyzing the problem and proceeding to solve it. And traditionally I have seen that most of the people will like to start from the program for which they have written to find the maximum or the program which they have written to find the minimum of N numbers. And using that program they try and modify that program to find out the maximum and minimum of N integers. This is something that I have seen nearly everybody doing, whoever has been assigned this problem as nearly everybody has started to do that way. And then when you consider on, whether it is the best possible one or not then several issues come in and the program gets modified to a final stage.

We shall see what that program turns out to be and then we shall see whether analyzing it from a previous stage, analyzing it from what we call the initial solution to a problem and generating other possibilities will lead to a better solution or not. So that's what we will see and we will go through this case study today to find the maximum and minimum of N numbers. So let us first see how a program to find the maximum of N numbers looks like and how anybody would very easily modify that program to find the maximum and minimum of N numbers. So let's go back and see how it works.

(Refer Slide Time 04:11 min)

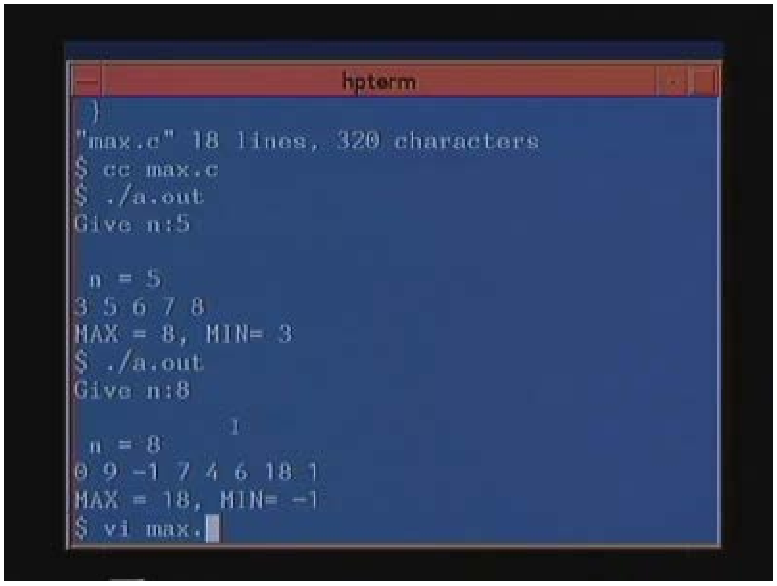


```
#include <stdio.h>
main()
{
 int n, max, i, numb;
 printf("Give n:");
 scanf("%d", &n);
 printf("\n n = %d \n", n);
 scanf("%d", &max);
  for (i = 1; i <= n-1;i++){
  scanf("%d", &numb);
   if (numb > max) max = numb;
   }

  printf("MAX = %d  \n", max);
 }
```

This is the program that we had written to find the maximum of N numbers. If you will recall, this was the program in which we declared integers n, max, i and number or num, you ask to give the value of n then you read n, you printed n. Then you read in the first number and you initialized it to the variable max. And then in this loop which starts from here and ends here, you read in n minus 1 number from i equal to 1 to i less than equal to n minus 1. And every time you read in a number you compare if numb is greater than max then you updated the value of max to numb. This was a program to find the maximum of n numbers and this is in order to see how it works and just revise, we will compile it. Suppose you give 6 and you give these numbers then the maximum is 8.

Now how do you modify it to find the maximum and minimum? What you do is you keep another variable called min like maximum is the temporary maximum. Similarly you keep a valid variable called min and find out the temporary minimum. And you initialize that here and here you compare. So what we do is we declare another variable called min and when you are reading in max, you initialize min to max also that is the first number is read and initialized to both min and max. And here when you read in the new number, what you do is you again compare if numb is less than min then min is assigned now. So this program should work, isn't it? Is that okay? This program should work that you have declared a new variable min, you have read in this, you have done this and if numb is greater than max, max is min numb if numb is less than min, min is min numb and here you have to print both max and min. So I print here min, so I print max and min.

(Refer Slide Time 07:35 min)



```
 }
"max.c" 18 lines, 320 characters
$ cc max.c
$ ./a.out
Give n:5

 n = 5
3 5 6 7 8
MAX = 8, MIN= 3
$ ./a.out
Give n:8

 n = 8
0 9 -1 7 4 6 18 1
MAX = 18, MIN= -1
$ vi max.
```

So let see how this will run. So, again you give 5 numbers. So, let’s run it again, to this how it works and its quite okay. This program is correct because, but the question that we will have to ask is this program the best or can we write a program which is better than. This in order to answer the question of what is better; you have to analyze what we mean by better. What we mean by better is two things whether a program, a program is set to be better than another program, if it runs faster than the other program. So now how will we make one program faster than another? There are formal methods to understand how a program can be faster than another but looking at this one, let us informally see the condition which we can improve, very informally if this condition holds true, this condition will not hold because if numb is greater than max then numb cannot be less than min because max and min can at most be the same value.

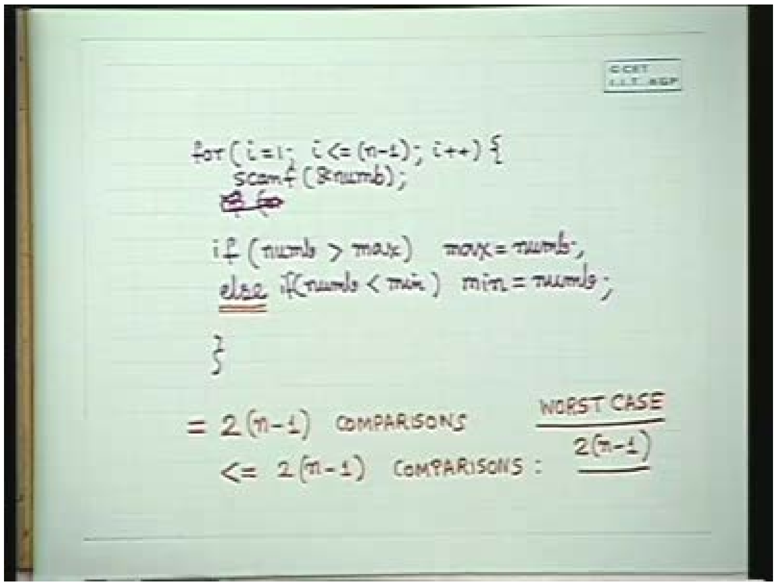
Therefore if we put an else here then at least if this condition is true, this condition will not be evaluated. And if this condition is false only then this condition will be evaluated. so if we modify the program and write and else out here. then obviously this program is expected to run faster than the previous program by whatever mini scale amount it maybe but still it will run faster than the other program. And obviously this program is correct because if a number is greater than max then max is assigned numb, you need not update the value of min. Only if the number is not greater than max then it may be greater and maybe less than min which you may have to check.

So let’s just run this program once. So this is also a correct program which works quite fine in the sense that it is at least better than the previous program in terms of time, all other things remaining the same this program runs better than the previous the program which we wrote earlier without this else. The next question which we will answer is, is this the best possible program or how good is this program. Is there any other possibility in which we can improve this program and make it better? In order to do that we will have to do some more paper and pencil analysis to see how other programs can be written.

Now let's analyze what we did in that program. The main part of the program consisted of a for loop which contain if you did a scanf of numb and the main part was if numb was greater than max then max was updated to numb else if numb was less than min then min was updated to numb. This was the main part of the program. Now in the place where else was not there, in that situation for total of n numbers we had to make 2 into n minus 1 comparisons, if this else was not there because for 1 to n minus 1 we have to make two comparisons and this possible assignment dependent on the value of the comparison. And in the worst case situation, we need both the comparisons and we need only one of the assignments. We cannot make both the assignments, you will make only one of the assignments and we make both the comparisons.

Now let us take this, when we put this else then we always did not make the 2 into n minus, both the comparisons inside the loop. What we did was if these comparisons turned out to be true we did not make this but if this turned out to be false only then we made it. Therefore what we did was we made something which was less than or equal to… The previous one was we made exactly t2wo into n minus 1 comparisons and only one assignment. In the other situation we made less than or equal to 2 into n minus 1 comparisons and one assignment, all other things remaining the same. This matter actually dictated the difference between the two programs. Therefore this program with the else was consider to be better than the other program but obviously you can see there exists examples in which even this modified program will make exactly 2 into n minus 1 comparisons.

(Refer Slide Time 14:04 min)



```
for (i=1; i<=(n-1); i++) {
   scanf(&numb);
   [illegible]

   if (numb > max)  max = numb;
   else if(numb < min)  min = numb;

}
```

$= 2(n-1)$ COMPARISONS — WORST CASE

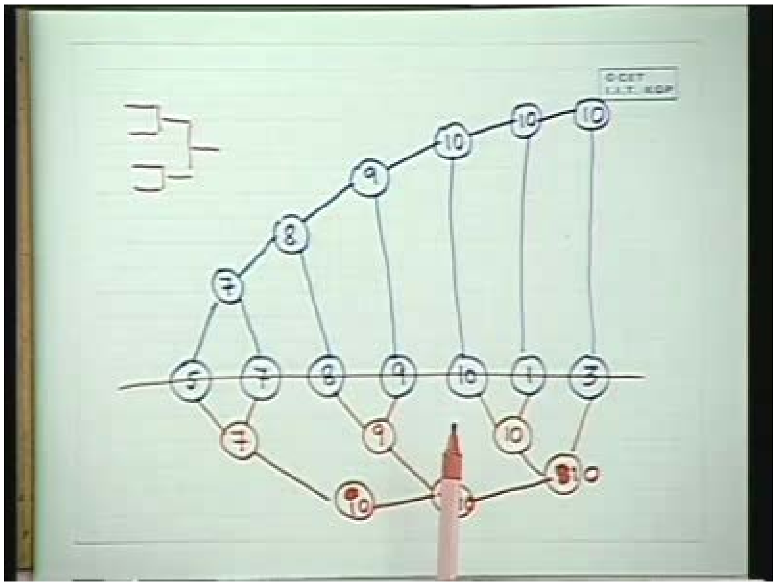
$<= 2(n-1)$ COMPARISONS : $2(n-1)$

Therefore in the worst case both the algorithms are going to make 2 into n minus 1 comparisons. So, in terms of worst case scenario both the algorithms are going to perform equally. The question is can we improve this worst case. The point here is we started with the maximum program and try to change it a little here and there. As I had mentioned in previous classes, we have to go a little more deeper into the problem and try and examined how we have solved the maximum problem itself. And whether we could change the problem to solve the maximum, change the algorithm to solve the maximum so that solving the minimum becomes easier. So that is what we will see now.

So let us see how we solve the problem of finding a maximum of n numbers. We saw a recursion example in a previous class, I will just bring you another flavor and we will analyze that recursive form a little later in a subsequent class and we will see both of them lead to a very similar scenario. Let us take a set of numbers. Now what we did was we compared the first two, then we compared this, then we compared this then we compared this then we compared this and then we compared this and this is the tentative value of the max variable which propagated through. This is how we worked it out.

We could have also worked it out in another way. we could have compared these two, we could have compared these two, we could have compared these two, we could have compared these two, we could have compare these two and we could have compare these two. Sorry I made a mistake here (Refer Slide Time: 16:27). This should be 10, so this should be 10 and this should be 10. So we could have done it this way also and we could have done it in many other ways and you know, can you tell me what this looks like. What does this look like and what does this look like? This looks like a fixture of a knock out tournament.

(Refer Slide Time 17:10 min)



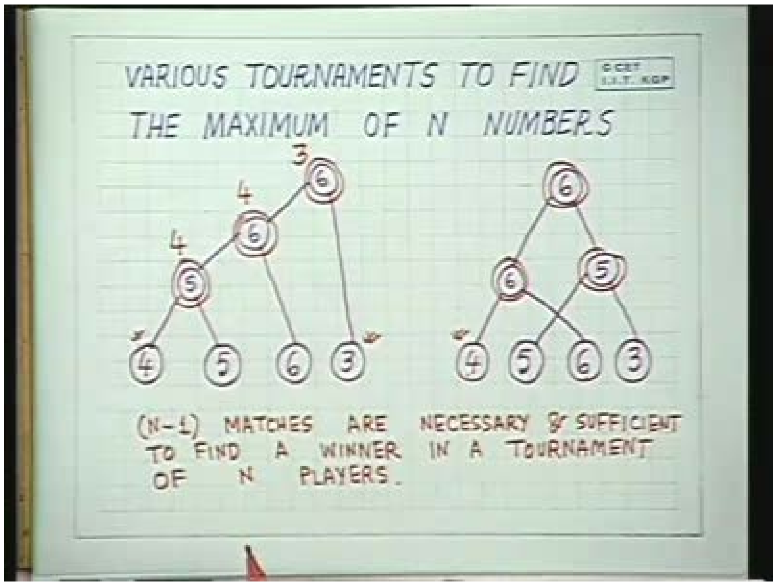
And actually it’s just a knout out tournament, we are just knocking somebody out, bio-mechanism of comparison. And anyway you fixed up this knock out tournament, we are going to get back the winner and whatever where we do it, we are going to get the maximum by this knock out tournament mechanism. So let’s have, let’s see we can have various tournaments to obtain the maximum of n numbers. And the question is why did we choose this tournament and not the other one. So let’s take two solutions side by side and let’s take two tournaments to find the maximum of a set of numbers.

Now very interesting thing about such tournaments is that to declare the winner of a tournament of n players, you must necessarily, see these are the players and these are the matches and the result of the matches, isn’t it. These are the matches which are played to find out the winner, these are the matches which are played to find out the winner. And in any such tournaments structure, to determine the winner of n players you need to have exactly n minus 1 matches. So n minus 1 matches are necessary and sufficient to find a winner in a tournament of n players. So whichever way you set up the tournament, you are going to require n minus 1 comparison to find the maximum of n numbers. And that is why doing it this way or doing it in any other way is equivalent to us.

And you can easily prove that if you do not have n minus 1 comparison, that is you take less than that there are situations in which you are not going to define the maximum value but if, so finding out the maximum of n numbers we took the easy way out and we did it this way because it’s made programming easier and it was possibly the most efficient way in which you can find the maximum of n numbers because you just cannot reduce the number of steps. But since we are finding the maximum as well as the minimum of n numbers, we must see whether another tournament would have been easier because during that tournament, we could have possibly found the minimum faster than the maximum. Because what we are doing here? Here, we are, in a tournament like this, we are finding the maximum and carrying the minimum. So we are carrying the minimum here 4, we are carrying the minimum here 4 and we are carrying the minimum here 3. And this is how we find out the minimum also and we required 2 n minus two into n minus 1 matches or comparisons in the worst case scenario.

The question is when finding the maximum can you have some other way to find out the minimum. Can you give a deeper thought into the problem in which you can obtain a solution to find out the minimum. Can you tell me? Look the maximum is that person who has won everything; the minimum is that person who will lose everything. So now 4 and 5, 4 has lost to 5 so 4 must be there to compare. Now 5 has won, so if 5 has won even a single match, 5 cannot be among the minimum, 6 has won, 3 has lost so you need to compare between 4 and 3 only and that’s what we did around here.

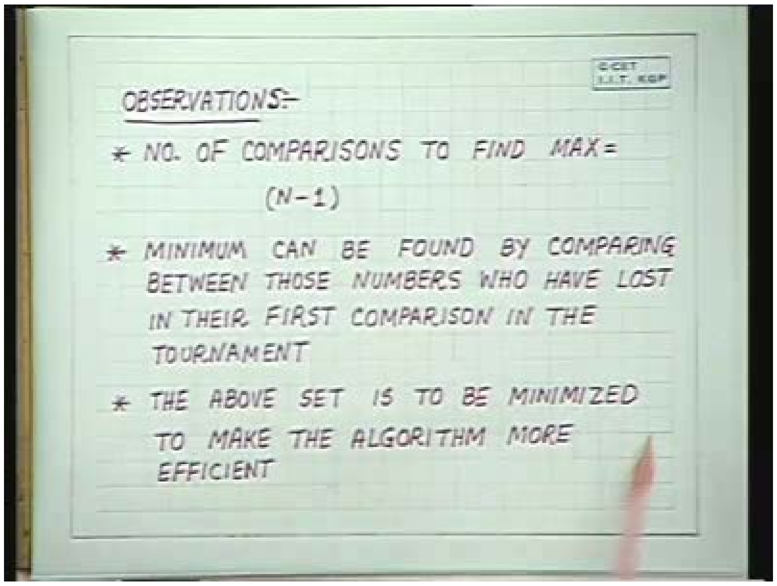
(Refer Slide Time 21:57 min)


VARIOUS TOURNAMENTS TO FIND THE MAXIMUM OF N NUMBERS

(N-1) MATCHES ARE NECESSARY & SUFFICIENT TO FIND A WINNER IN A TOURNAMENT OF N PLAYERS.

Now look at this point, here 4 and 6 were compared, so 4 lost and 5 and 3 where compared and 3 lost. Here 5 and 6 were compared and 5 lost but this fellow will not participate in becoming the minimum because he has won at least one match. So the minimum will be among those people who have lost the first match. It has to be among those people who have lost their first match. So when finding the maximum through a tournament, you have, the winner will have to defeat everybody in some way or the other but the minimum can be found among the losers who have lost their first match only. And if you could minimized this set, if we could set up a tournament which minimizes the set of losers in the first match then only among those losers you will find out the minimum. So that is how you could solve the minimum mode efficiently. So that is the insight that we will use to solve the maximum and the minimum of n numbers.

(Refer Slide Time 23:40 min)


C CET I.I.T. KGP

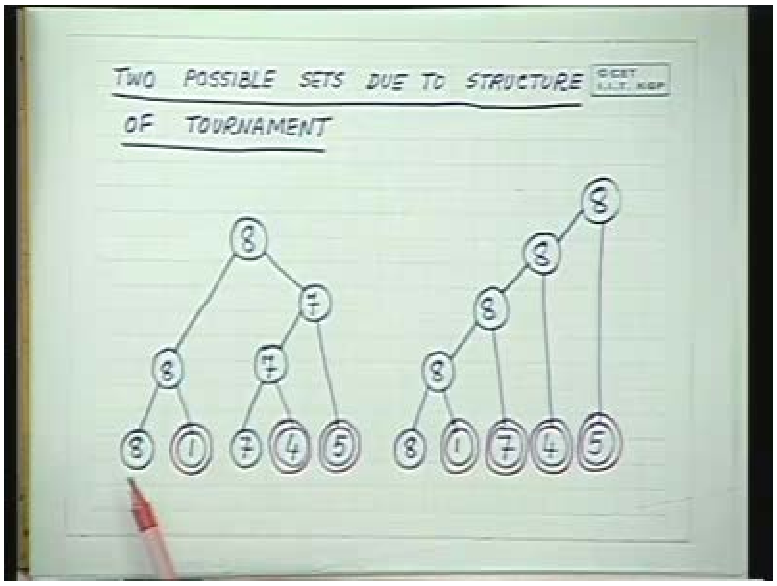
OBSERVATIONS:-

* NO. OF COMPARISONS TO FIND MAX = (N-1)
* MINIMUM CAN BE FOUND BY COMPARING BETWEEN THOSE NUMBERS WHO HAVE LOST IN THEIR FIRST COMPARISON IN THE TOURNAMENT
* THE ABOVE SET IS TO BE MINIMIZED TO MAKE THE ALGORITHM MORE EFFICIENT

So we make three very simple observations. The number of comparisons to find max is n minus 1, that is the first observation that we have made. The second is the minimum can be found by comparing between those numbers who have lost in their first comparisons in the tournament. And in order to make the algorithm more efficient, the above set of number, the set means the set of numbers who have lost in their first comparisons in the tournament is to be minimized to make the algorithm more efficient. So you cannot make less than n minus 1 comparison to find max that's fine but you could do something to find minimum more efficiently.

So let's see what we can do here. Let us set up two possible tournaments, one is the standard one which we have used where here these are the losers in their first comparison. And here these are the losers in the first comparison. Does that give you any idea of how to set up the tournament, so that the losers set of the first comparison is reduced. Can the losers set in the first comparison be less than n by 2. The losers set in the first comparison can never be less than n by 2 because you will have to play them at least two two in order to for somebody to loose. So if we want to minimize the loser set of the first comparison, we must have a tournament in which they play pair wise first.

(Refer Slide Time 25:18 min)

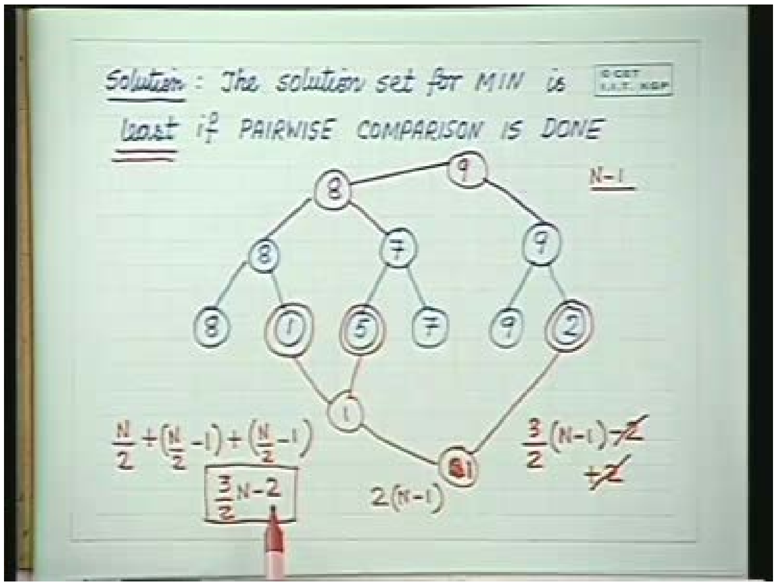


So we make them play pair wise and once they play pair wise, we declare the winners and the losers. And among the winners you can find the maximum and among the losers we can find the minimum, let's see. Suppose we have 8 1 5 7 9 and 2, we compare them pair wise. The winners are 8 here, 7 here and 9 here and these are used to declare the winner and set up, continue the tournament this side. And what you can do is you can have another tournament between the losers in the reverse direction, sorry this will be 1 (Refer Slide Time: 26:15) and declare the minimum. So the solution set for min will be least, I am just giving an informal argument you can just sit down and prove it, it will not take much time if pair wise comparison is done. It will be least in other situations also but at least if pair wise comparison is done, it is definitely going to be least.

And therefore how many comparisons are you going to make? Here you are going to make let us say if n is e1 then in the first round pair wise we are going to make n by 2 comparisons to declare the first round winners and losers. And then among this we have to declare the winner that is maximum of N by 2 numbers, maximum of N numbers is N minus 1, so maximum of n by 2 numbers is this. These are the numbers of comparisons, the rest remaining same. The minimum of N by 2 numbers again this. This is the first round, pair wise comparison. This is the maximum among the winners; this is the minimum among the losers. To find the maximum among N numbers, you required N minus 1 comparisons. Therefore to find the maximum among N by 2 numbers, you require N by 2 minus 1. Similarly to find the minimum among N by 2 numbers, you require N by 2 numbers. And you end up requiring 3 by N minus 2 this is for e 1. For odd you can take the case out and it will work similarly. For odd what do you do is you leave one number out and do it for the even case and then compare this new number, the last number we have both the maximum and the minimum.

That way in the odd scenario if odd, if n is odd then for the even part you will require this and the last one will have to be compared with both the maximum and the minimum. So, these two will cancel out and for all we require this. So, both of them are anyway less than the number which you had before. Therefore if we work out the algorithm this way, we are bound to get a solution which is much better than the previous scenario. This is the point which I wish to stress. The point was where did you start from. The previous algorithm which we discussed, we started from the code of the maximum of n numbers that is the final algorithm of the maximum of n numbers and we tried to improve that, whereas in the second case we went deeper into how the maximum of n numbers were solved. And we built up a conceptual structure in terms of whatever we called it a tournament or a game or something and we had this structure of that is this was the way in which the problem was solved and these were the alternatives in which the problem could have been solved in an equivalent fashion. And then we saw that if you took an alternative approach then finding the minimum of n numbers would have been easier.

(Refer Slide Time 30:32 min)



Solution : The solution set for MIN is least if PAIRWISE COMPARISON IS DONE

$\frac{N}{2} + \left(\frac{N}{2} - 1\right) + \left(\frac{N}{2} - 1\right)$

$\boxed{\frac{3}{2}N - 2}$

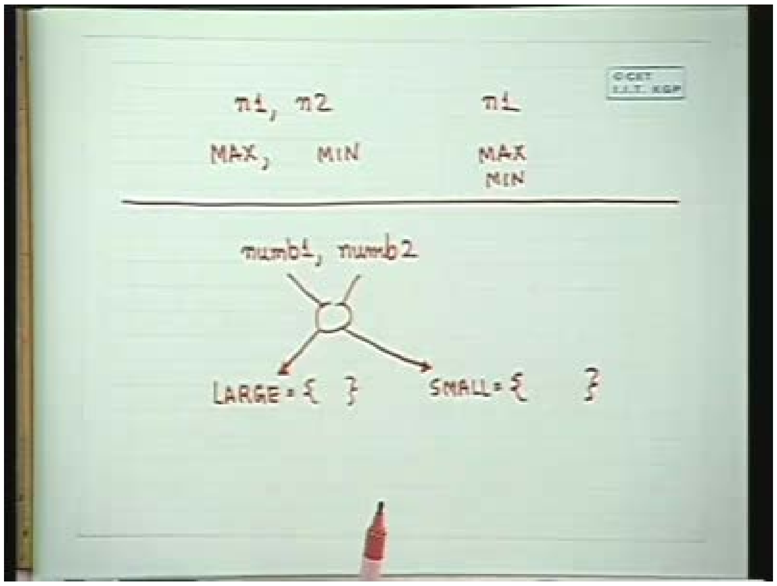
$\frac{3}{2}(N-1) - \cancel{2}$

$+ \cancel{2}$

This is the heart of problem solving. Structuring of information looking deeper into the problem and organizing your data conceptually, this is conceptual data structuring. And this we will see will be converted to your program. So if we start from a program which is the final algorithm, we are going to lead somewhere which is not possibly the best. Though I will not go in to it, you can prove that there can be no algorithm which can take less than so many comparisons, if comparisons is the only measure of solving, but we will not go into that presently. So the second is that now we have got our conceptual data structuring or conceptual algorithm, we have decided, now we will decide how we go and convert it to code. Converting into code is now no problem at all. We read in two numbers n1 and n2 and you initialize your max and min by comparing these two numbers.

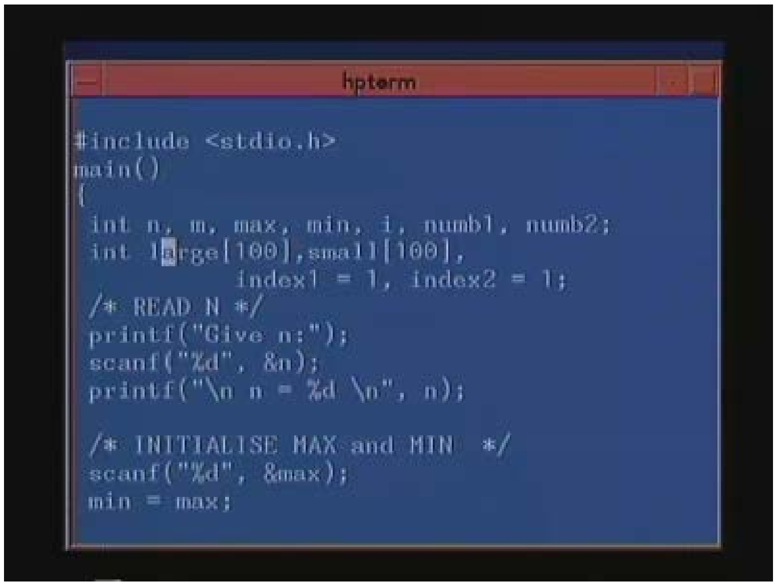
If it's odd you just read in one number and initialize max and min to that okay. That's the first part. And then what do you do is or you could have just read the numbers pair wise, say you read the numbers two at a time numb1 and numb2 and compare them. The larger one, you put in a set large and the smaller one, you put in a set small. This is exactly our tournament structure, you take the numbers pair wise, read them two at a time suppose it's even, read them two at a time, compare them.

(Refer Slide Time 32:01 min)



The winner you put in a set large, the loser you put in a set small and then after you have set all the number then find the maximum from this set large and find the minimum from this set small. So this is how you can easily convert it into code. Any questions out here? So let's see how the code will look like. Yeah, this is the quite a big code but let's see. The following parts are there to the code. The main program in which I have declared n the numbers to be read, m is a temporary variable, max and min are the temporary max and min, i is the loop control, numb1 and numb2 as I mentioned the pair wise numbers will be read.

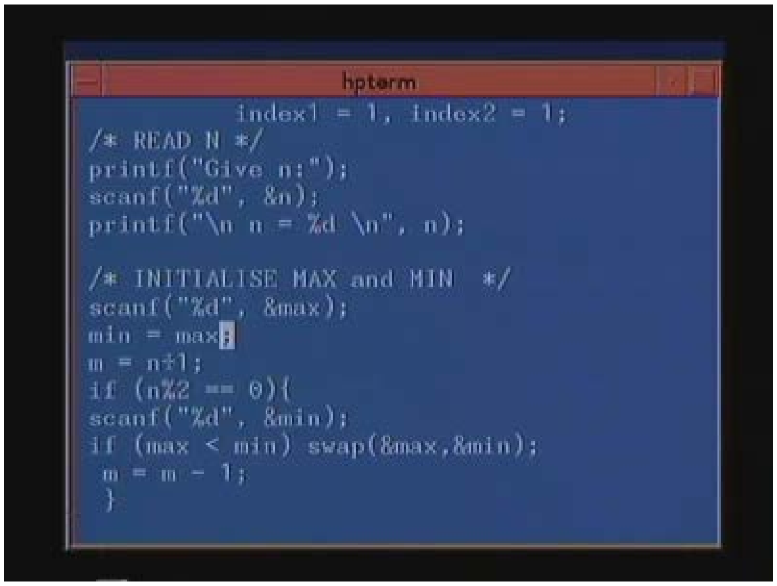
(Refer Slide Time 33:11 min)



```
#include <stdio.h>
main()
{
 int n, m, max, min, i, numb1, numb2;
 int large[100],small[100],
           index1 = 1, index2 = 1;
 /* READ N */
 printf("Give n:");
 scanf("%d", &n);
 printf("\n n = %d \n", n);

 /* INITIALISE MAX and MIN  */
 scanf("%d", &max);
 min = max;
```

I have declared two arrays large and small of integers of size 100 each, these will be the array indexes which I will use. So first I read n and print it and then odd and even case I separate out. If I want to read, if I want to have even numbers then I read into, I am assuming here that we will always get more than two numbers, we could modify the program to get less than two.

(Refer Slide Time 34:03 min)



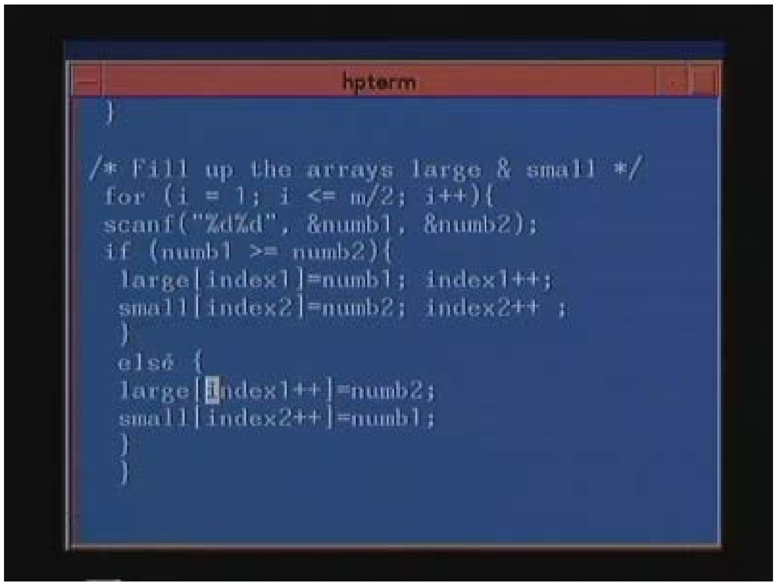
```
           index1 = 1, index2 = 1;
 /* READ N */
 printf("Give n:");
 scanf("%d", &n);
 printf("\n n = %d \n", n);

 /* INITIALISE MAX and MIN  */
 scanf("%d", &max);
 min = max;
 m = n-1;
 if (n%2 == 0){
 scanf("%d", &min);
 if (max < min) swap(&max,&min);
  m = m - 1;
  }
```

So if it is odd then I will read in only one number say max, I initialize max and min to this and max n is equal to n minus 1. And if it is even I will read in another number and then I will swap, I will read it, I just read it into the first number I read it to variable max and I initialize min and max to the same thing. The second I read in the variable min and if max is less than min then I swapped the values of max and min and I decremented m by 1 that is same as that I read in two numbers.

So now here is the main loop which we were working out, fill up the arrays large and small. I will read two numbers pair wise, so I will read i equal to 1 to m by 2 because if it was odd I get in one number and made n equal to n minus 1, if it has even I read in two numbers and made m equal to n minus 2. So now I will read in only the rest of the numbers pair wise. So this loop will go from i equal to 1 to i equal to m by 2 and I will read in two numbers numb1 and numb2 and then if numb1 is greater than or equal to numb2 then I will put numb1 in the array large and I will put numb2 in the array small and after doing it, I will increment the indexes and increment the index here.

Otherwise that is if numb1 is less than numb2 then I will put numb2 in large and numb1 in small. So this is fine and I just bypass here. Here I have written index 1 and then incremented index 1. What I have done here is I have written index plus plus, index 1 plus plus here and index 2 plus plus here. There is a, then what does it mean? Does it increment first here in this one, it must do large index equal to numb 2 and index 1 plus plus, this is equivalent to this in the sense that this index 1 plus plus is a statement which will return something, in C every statement returns a value. So index 1 plus plus first returns the value of index 1 and then increments. Therefore it will work quite well in our scenario where we want large index equal to numb 2 and index to be index plus 1.

(Refer Slide Time 36:35 min)

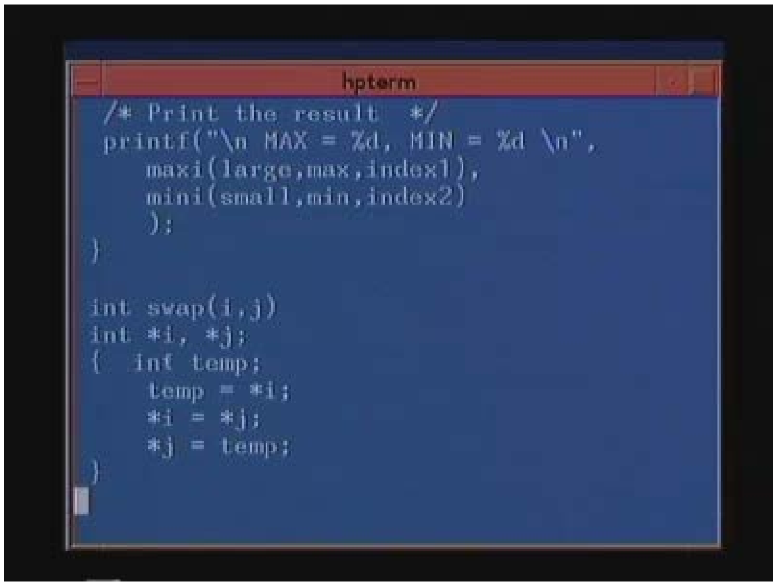


```
 }

/* Fill up the arrays large & small */
 for (i = 1; i <= m/2; i++){
 scanf("%d%d", &numb1, &numb2);
 if (numb1 >= numb2){
  large[index1]=numb1; index1++;
  small[index2]=numb2; index2++ ;
  }
  else {
  large[index1++]=numb2;
  small[index2++]=numb1;
  }
  }
```

So, you can also write the code this way. You could have, there is another way plus plus index, we shall come to that plus plus index 1 means it will first increment and return the value. Therefore if we had returned plus 1, plus plus index 1 we would have first incremented and put it in the value of the next index. But what this means here is that we will first put at index 1 and then increment. We can have a look at books to see the difference between index 1 plus plus and plus plus index 1 but the main idea is that if numb 1 is greater than numb 2, we put the larger in the set large and the smaller in the sets small.

And once we have done that for all the elements we then just print the maximum and minimum by calling two functions. One is maxi which finds the maximum in the set large and mini which finds the minimum in the set small. So in maxi, I pass the array large, I pass the initial max which I had found by comparing 1 or 2 elements and I pass the total number of elements in the index. This is actually total number of elements plus 1.

(Refer Slide Time 38:04)

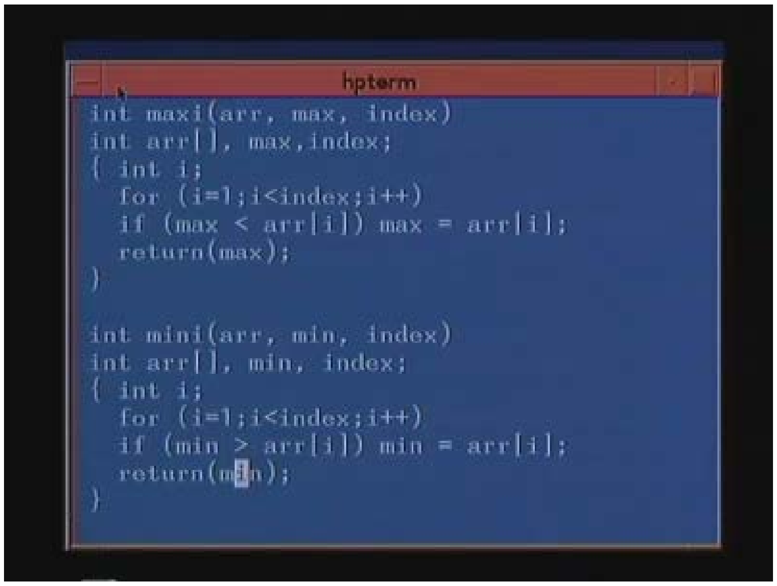


```
 /* Print the result  */
 printf("\n MAX = %d, MIN = %d \n",
    maxi(large,max,index1),
    mini(small,min,index2)
    );
}

int swap(i,j)
int *i, *j;
{  int temp;
    temp = *i;
    *i = *j;
    *j = temp;
}
```

And in mini I pass the arrays small, I pass the current minimum which I had computed earlier and I pass the index 2 and this is the swap routine which we had seen earlier and this is the maxi routine. In the maxi routine, I have declared int though this int was not necessary because this is not returning, it is returning the maximum. So sorry this int is necessary, maxi this is the array this is to be passed, this is the initial maximum and this is the size of the array plus 1. Now look in C language in order to pass the array, you have to just simply pass the array name. We shall come to array passing in more details later on but this is just an introductory example in which to pass the array, you can just have to pass the array name and you have to declare that it is an array of integers, the size need not be given. Why, how it works we shall come to later on.

And therefore this is the format for the time being let us understand. There are other ways of doing it which we shall see later. And if you modify an array index inside here, the value will be modified but though we are not going to do that in this example. So we have declared an array and what we have done is for i equal to 1 till i is less than index we just computed the maximum of elements in the array arr. Similarly mini is identical to maxi except that it just finds the minimum by this and this one returns the max variable and this returns the min variable.

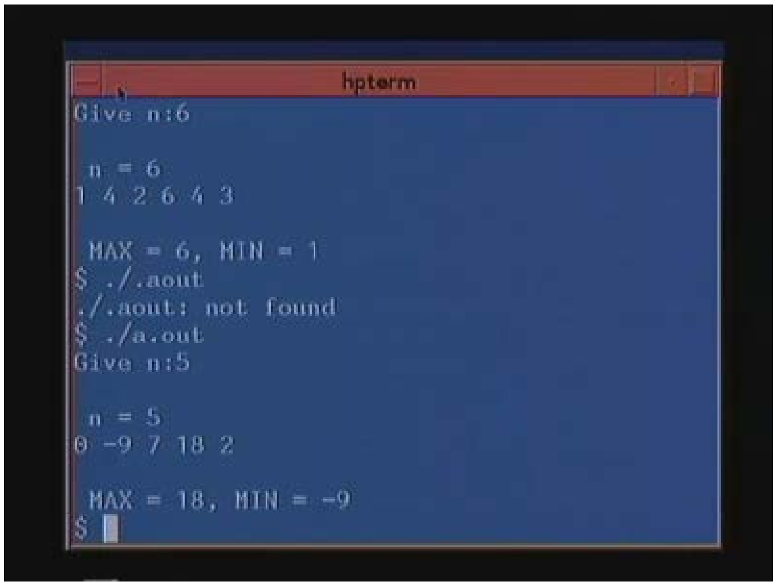
(Refer Slide Time 39:41 min)



```
int maxi(arr, max, index)
int arr[], max,index;
{ int i;
  for (i=1;i<index;i++)
  if (max < arr[i]) max = arr[i];
  return(max);
}

int mini(arr, min, index)
int arr[], min, index;
{ int i;
  for (i=1;i<index;i++)
  if (min > arr[i]) min = arr[i];
  return(min);
}
```

So this way we have computed the maximum of n numbers and the minimum of n numbers by two functions. So after having populated the array is large and small as we discussed through the tournament mechanism, we combined these two and made it into call these two functions and we print the result of maximum and minimum. So this is the algorithm that we obtain finally.

(Refer Slide Time 40:52 min)

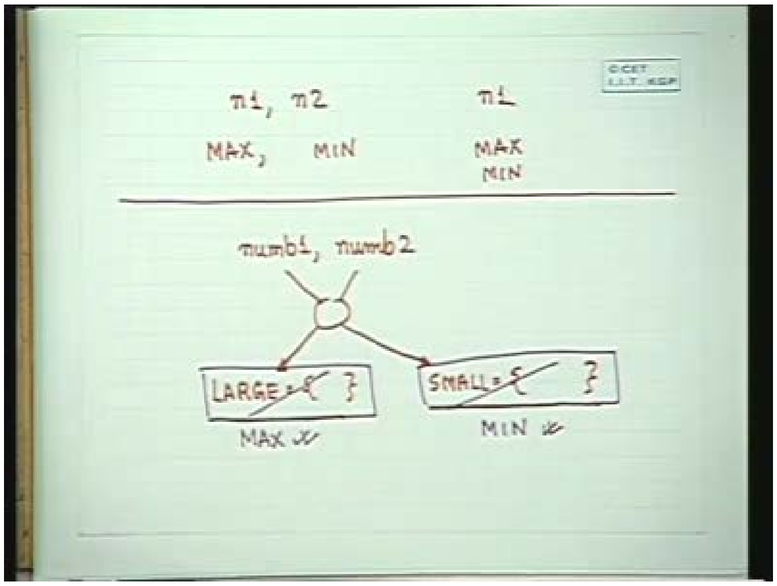


```
Give n:6

 n = 6
1 4 2 6 4 3

 MAX = 6, MIN = 1
$ ./.aout
./.aout: not found
$ ./a.out
Give n:5

 n = 5
0 -9 7 18 2

 MAX = 18, MIN = -9
$
```

So, this way we are going to get the correct program and the program is going to run quite efficiently also. So we have got a program which works efficiently and something which has come out of much more deeper analysis but let us see whether we can even improve this program like we improved the max program to get the maximum and then we improved it. So there is some post optimization which we can do at this point. Can anybody suggest what pose optimization is supposed to do?

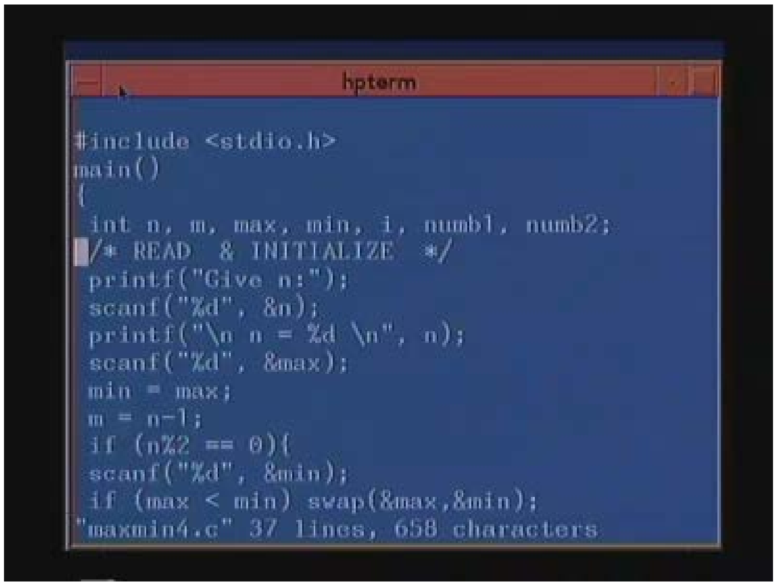
No, no, very simply when we populated these two arrays large and small we need not have put in the arrays. We could have kept a temporary max here and temporary min here and whenever we compare two numbers, the larger one we compare with the temporary max and the smaller one we compare with temporary min. And we could have combined that maximum and minimum finding with this splitting up into these two steps into one short and if we did that together we would not require these two arrays and we would have saved space. We wouldn't have saved on number of comparisons anymore but we could have definitely not required these two arrays at all. So we can get out of these two arrays by saying that when we combine numb 1 and numb 2 and then the lager one you compare with the max and update the value of max and the smaller one you compare with the min and update the value of min.

(Refer Slide Time 42:31 min)



That way you will not require these two arrays. With this simple modification you will reach an algorithm which does not require any arrays any more. Just let's have, I think it's quite clear, just have a look at what the program looks like.

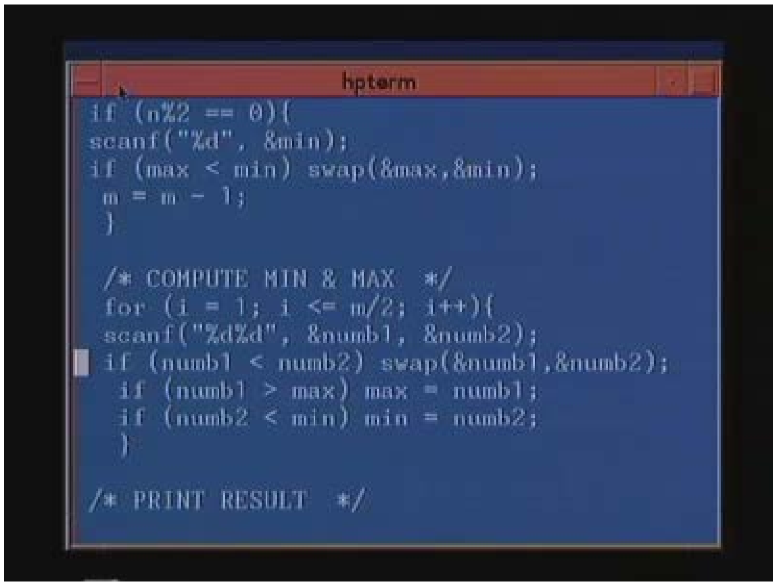
(Refer Slide Time 42:43 min)



```
#include <stdio.h>
main()
{
 int n, m, max, min, i, numb1, numb2;
/* READ  & INITIALIZE  */
 printf("Give n:");
 scanf("%d", &n);
 printf("\n n = %d \n", n);
 scanf("%d", &max);
 min = max;
 m = n-1;
 if (n%2 == 0){
 scanf("%d", &min);
 if (max < min) swap(&max,&min);
"maxmin4.c" 37 lines, 658 characters
```

This read and initialize is the same thing, up to here we read in 1 or 2 numbers and here we compare two numbers and I have written it in short you compare two numbers and then if numb 1 is greater than numb 2 you just swap numb 1 and numb 2. So once you swap numb 1 and numb 2, now you have got the larger n numb 1 and the smaller n numb 2.

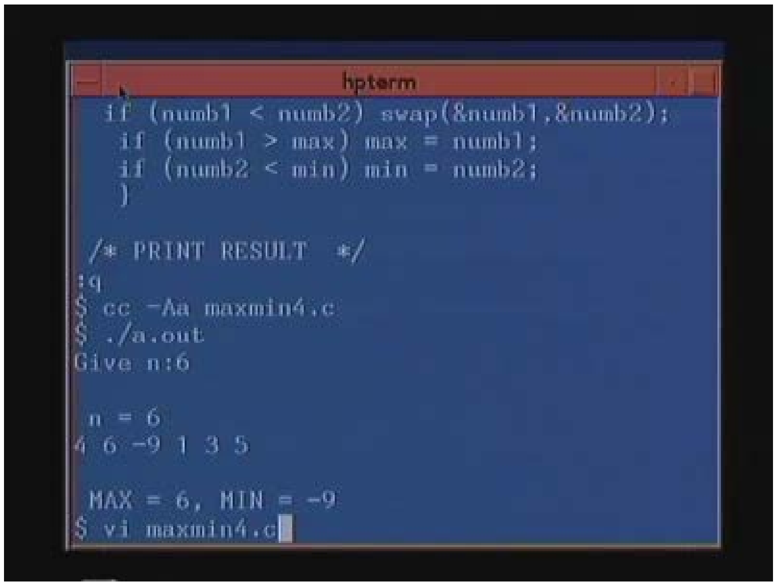
(Refer Slide Time 43:04 min)



```
 if (n%2 == 0){
 scanf("%d", &min);
 if (max < min) swap(&max,&min);
  m = m - 1;
  }

  /* COMPUTE MIN & MAX  */
  for (i = 1; i <= m/2; i++){
  scanf("%d%d", &numb1, &numb2);
  if (numb1 < numb2) swap(&numb1,&numb2);
   if (numb1 > max) max = numb1;
   if (numb2 < min) min = numb2;
   }

 /* PRINT RESULT  */
```

And now you say if numb 1 is greater than max, max equal to numb 1. If numb 2 is less than min, min equal to numb 2 but here you cannot put the else, both of the things may simultaneously exist. So here you cannot put that if numb 1 is greater than max, max equal to numb 1 else that will make it wrong. Here you have to have both the statements.

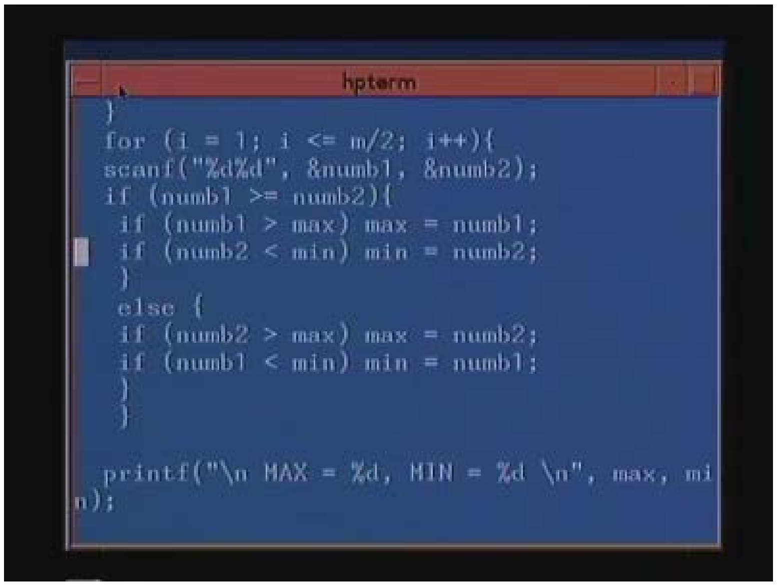
(Refer Slide Time 44:09 min)



```
 if (numb1 < numb2) swap(&numb1,&numb2);
  if (numb1 > max) max = numb1;
  if (numb2 < min) min = numb2;
  }

/* PRINT RESULT  */
:q
$ cc -Aa maxmin4.c
$ ./a.out
Give n:6

 n = 6
4 6 -9 1 3 5

 MAX = 6, MIN = -9
$ vi maxmin4.c
```

So now we have got rid of the array and we got a program in which we have combined the two faces together. So this is the post design face in which you combine all your ideas together and see whether you can even optimize further. This program will work but just for our personal comfort. So here this program works with the swap and well you could have removed the swap also because you could have put in this swap in an additional statement which you will exchange two numbers. Here you could have said if numb 1 is less than numb 2, you could have done this else you could have done something else.

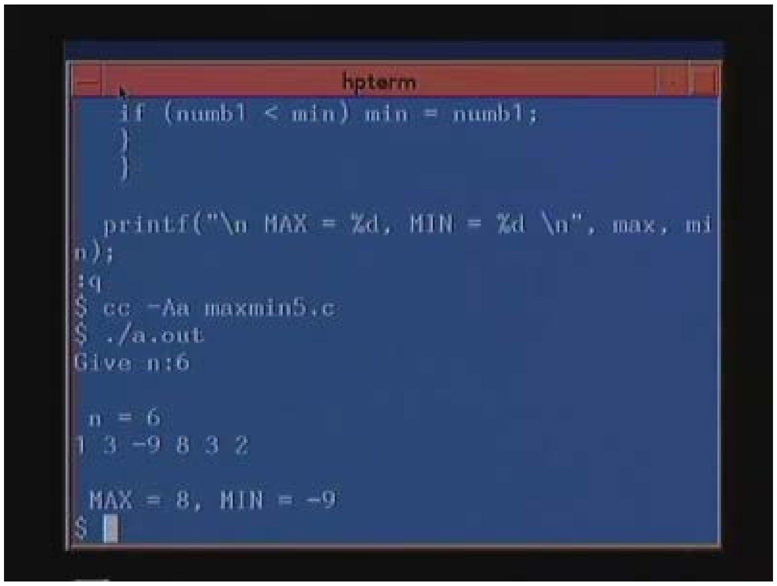
(Refer Slide Time 44:54 min)



```
 }
 for (i = 1; i <= m/2; i++){
 scanf("%d%d", &numb1, &numb2);
 if (numb1 >= numb2){
  if (numb1 > max) max = numb1;
  if (numb2 < min) min = numb2;
  }
  else {
  if (numb2 > max) max = numb2;
  if (numb1 < min) min = numb1;
  }
  }

 printf("\n MAX = %d, MIN = %d \n", max, mi
n);
```

That is else you could have compared numb 2 with max and numb 1 with min. That is the final thing that you could have done and this is what you will end up with a final program in which all the other things remaining same, this part will change. If numb 1 is greater than or equal to numb 2 when you compare numb 1 with max and numb 2 with min else you compare numb 2 with max and numb 1 with min and you get rid of the swap statement as well.

(Refer Slide Time 45:34 min)



```
    if (numb1 < min) min = numb1;
    }
    }

  printf("\n MAX = %d, MIN = %d \n", max, mi
n);
:q
$ cc -Aa maxmin5.c
$ ./a.out
Give n:6

 n = 6
1 3 -9 8 3 2

 MAX = 8, MIN = -9
$ 
```

And finally after doing all your analysis, this is where you can end up with a program which is not only correct and efficient but looks reasonable nice also. So, this is how, this is what we end up with and the whole idea behind this case study was to show that analysis of the problem from the beginning, examining all the possible solutions is what data structuring, problem solving and programming is all about. And that is what is absolutely necessary when you solve a problem. This was a very simple problem and it has so many issues involved. Therefore when you come to a more involved problem, you should always look at it with a fresh mind, examine not only how you solved other problem, what you did for other problem but how you could solve other problems in other way to solve this problem better that was what data structuring is about here. Here look at data structuring, you compare two numbers and you just organized it into two sets. So while finding out the maximum, you structured your data in such a way that finding out the minimum became easier. This is what data structuring is all about. This is the first flavor of data structuring that you will have and we will continue with such things in subsequent classes.