Programming and Data Structure

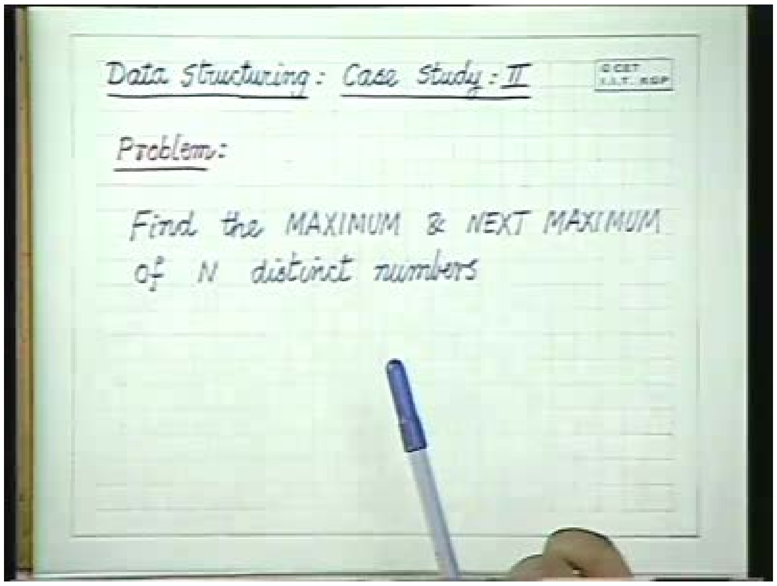
Dr. P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture # 06

Data Structuring: Case Study - II

We continue our study of data structuring and we have seen the problem of solving the maximum and minimum of n numbers. Now we examine the second problem, very similar problem, the problem of finding the maximum and the next maximum of n distinct numbers. That is the largest and the second largest of n distinct numbers.

(Refer Slide Time 01:37 min)



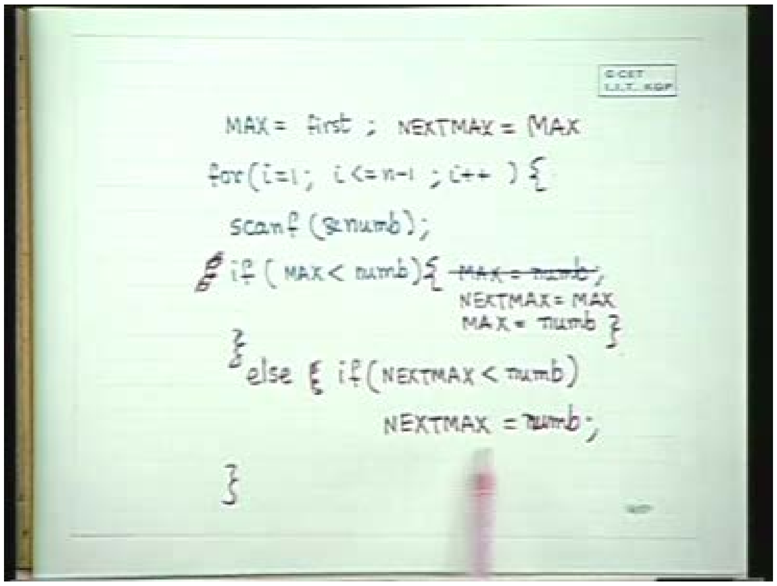
Data Structuring : Case Study : II

Problem:

Find the MAXIMUM & NEXT MAXIMUM of N distinct numbers

So like the maximum and the minimum, we will go through the mechanism of finding out the largest and the second largest by modifying the largest, the program for finding out the largest n numbers. And let us see what we will achieve if we do that. In the largest of n numbers, we initialized the value of max to the first number and then in the for loop what we did was we read in a number and if max was less than numb then we made max equal to numb. That was the simple thing that we did to find out the maximum of n numbers. And in order to find the maximum and the second maximum, we will also take a very similar approach that we took to find the minimum. And I have seen nearly everybody do it this way, when I asked anybody for the first time, next max is equal to max. And here this comparison if max is less than number then this is not the only thing that you do instead of this, what you do is you first make nextmax equal to max and then you make max equal to numb.

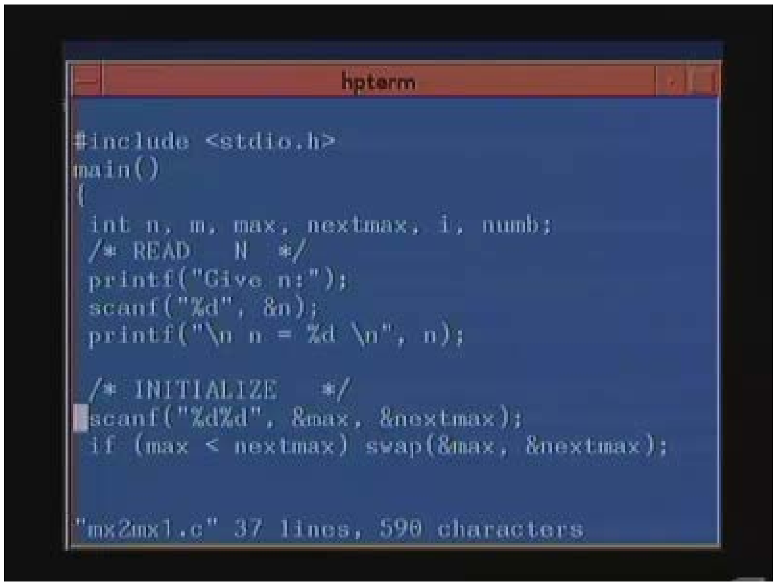
(Refer Slide Time 04:41 min)



```
MAX = first ;  NEXTMAX = MAX
for(i=1; i<=n-1 ;i++ ) {
  scanf (&numb);
{ if ( MAX < numb){ ~~MAX = numb~~;
                     NEXTMAX = MAX
                     MAX = numb }
  }
  else { if(NEXTMAX < numb)
              NEXTMAX = numb;
  }
```

If this is not the case then this number maybe still greater than the second maximum. So what you do is you write an else and then you write the if nextmax, else is essential is less than numb then you just assign nextmax to numb because max remains unchanged. So this is like this, these two brackets go into together. Please be careful about uppercase and lower case, they are different in C language. So this is how you would modify the program that if max is less than numb, next max is equal to max, max equal to numb else if nextmax is less than numb then nextmax is equal to numb. So, I have a program for that and just lets run that which does exactly the same thing, it reads in n here and then reads in the two values max and nextmax.

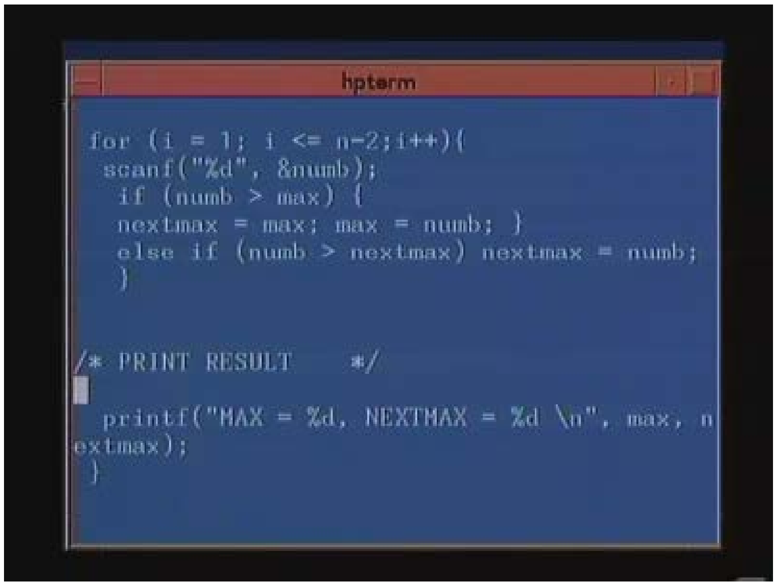
(Refer Slide Time 05:11 min)



```
#include <stdio.h>
main()
{
 int n, m, max, nextmax, i, numb;
 /* READ   N  */
 printf("Give n:");
 scanf("%d", &n);
 printf("\n n = %d \n", n);

 /* INITIALIZE   */
 scanf("%d%d", &max, &nextmax);
 if (max < nextmax) swap(&max, &nextmax);


"mx2mx1.c" 37 lines, 590 characters
```

So we assume here that there are more than two numbers obviously and the mechanism of initializing those two, I have done it in this way that if max is less than nextmax just we swap max and nextmax, so we have initialized the two numbers. And here is exactly the code I read down in the piece of paper but here there is a slight modification, since we are reading two numbers I have done i less than equal to n by minus 2. You could easily modify to do the other things and put all the checks. Scanf numb, if numb is greater than max then nextmax assign max and max is assigned numb and obviously you cannot do this statement after you do this statement then you would lose the value of max. So you must do in proper order else if numb is greater than nextmax, nextmax is assigned now and simply print the values of sorry print the values of max and nextmax by this state, this is the swap routine.

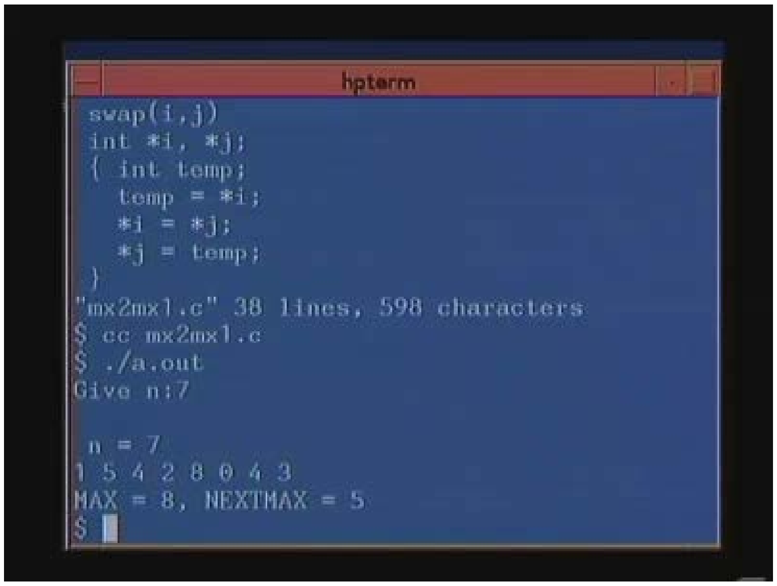
(Refer Slide Time 06:05 min)



```
 for (i = 1; i <= n-2;i++){
  scanf("%d", &numb);
   if (numb > max) {
   nextmax = max; max = numb; }
   else if (numb > nextmax) nextmax = numb;
   }


/* PRINT RESULT    */

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

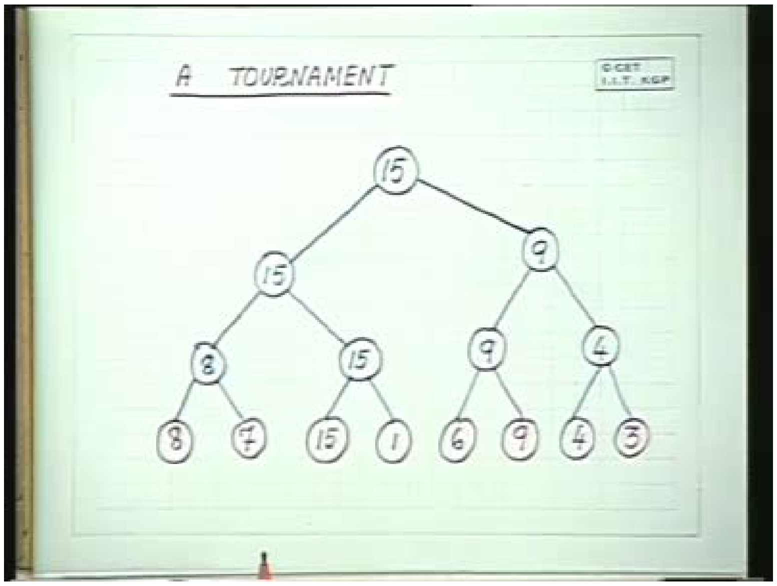
So this program is quite obvious that it will work correctly, if you get more than two numbers.

(Refer Slide Time 06:57 min)


```
swap(i,j)
int *i, *j;
{ int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}
"mx2mx1.c" 38 lines, 598 characters
$ cc mx2mx1.c
$ ./a.out
Give n:7

 n = 7
1 5 4 2 8 0 4 3
MAX = 8, NEXTMAX = 5
$ 
```

So now let’s again go back to paper and pencil analysis to see if you could be anything better. So this is what our, the core of our program was and in any case to do this if you look at comparisons only then you are definitely going to do how many comparisons at this point. This comparison will work and in the worst case, you have to do both the comparisons because actually both may fail and you will have to do both the comparisons. So you will end up doing something like 2 into n minus 1 or 2 into n minus 2 comparisons finding out the maximum and second maximum of n numbers. The question is could you do better. Then anybody have any suggestions one more to do. Again we will go back to how max was solved and we will see whether we can do anything while solving max to get the next maximum.

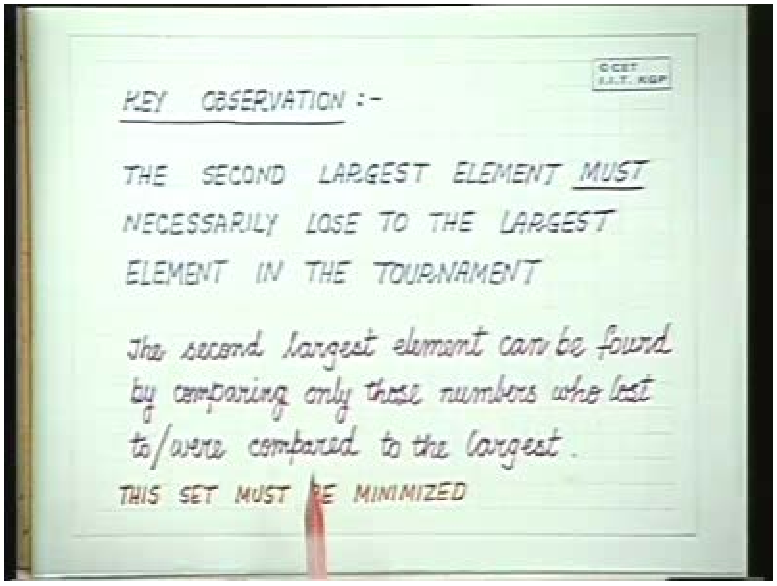
(Refer Slide Time 08:38 min)



And the whole solution structure of finding out the maximum of n numbers, whichever way we work it out actually inherently lies in the tournament structure. So let's take any tournament of 1 2 3 4 5 6 7 8 numbers I have taken and this is the tournament to find the maximum. Now while finding the minimum, we compared only the losers. Now is there any key observation to find out the second largest? Can 4 be the second largest? The person who is the second best player in this tournament must have lost to the best player sometime. That is the core idea. The second largest must definitely lose to the largest in a comparison; otherwise he is not the second largest. You cannot declare a person winner by, unless you defeat the second largest possible, somewhere along the tree. For example here the second largest will occur between this fellow, this fellow and this fellow, must occur.

The second largest must occur among those people who have lost out to the largest; this is an absolutely key absolute key observation. And these are the necessary and sufficient items which are to be compared to find out the second largest. That is you have to compare among the losers those who lost to the largest in the tournament and among then you find out the maximum, you will get the second largest element. For example here between 1 8 or 9 if you find out the maximum, you will find the second largest element, the rest are unnecessary to compare. So we come to the key observation here. The second largest element must necessarily lose to the largest element in the tournament point number one.

(Refer Slide Time 10:52 min)


KEY OBSERVATION :-

THE SECOND LARGEST ELEMENT MUST NECESSARILY LOSE TO THE LARGEST ELEMENT IN THE TOURNAMENT

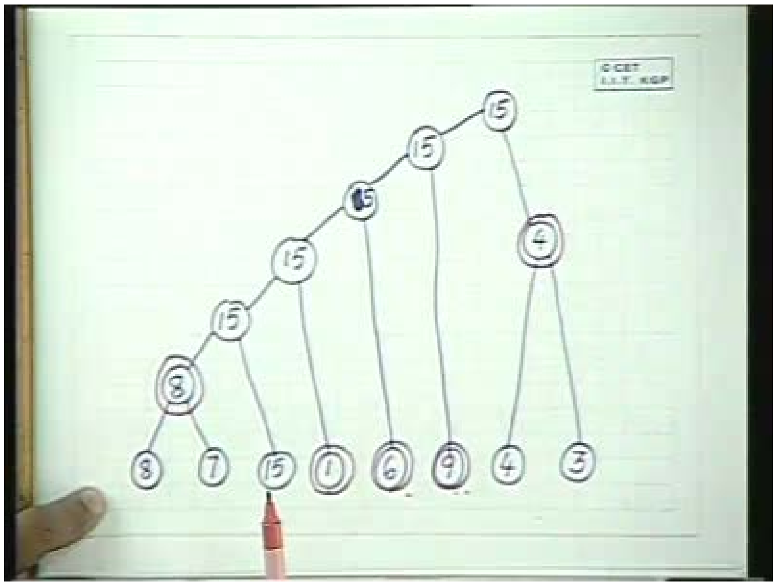
The second largest element can be found by comparing only those numbers who lost to/were compared to the largest.

THIS SET MUST BE MINIMIZED

And now we come to the question of optimizing, the second largest element can be found by comparing only those elements who lost to or were compared to the largest. Along the tree, now not only at the first level as in the max and min, in the max and min we satisfied by first time losers but here it is not first time losers. You must have lost somewhere in the full tournament and what do we want to do. Finding out the maximum of n numbers will require n minus 1 comparison. We must minimize this set to find out the second largest. So how will be minimize this set depends on how we set up the tournament. This is one case of this set as we just saw.

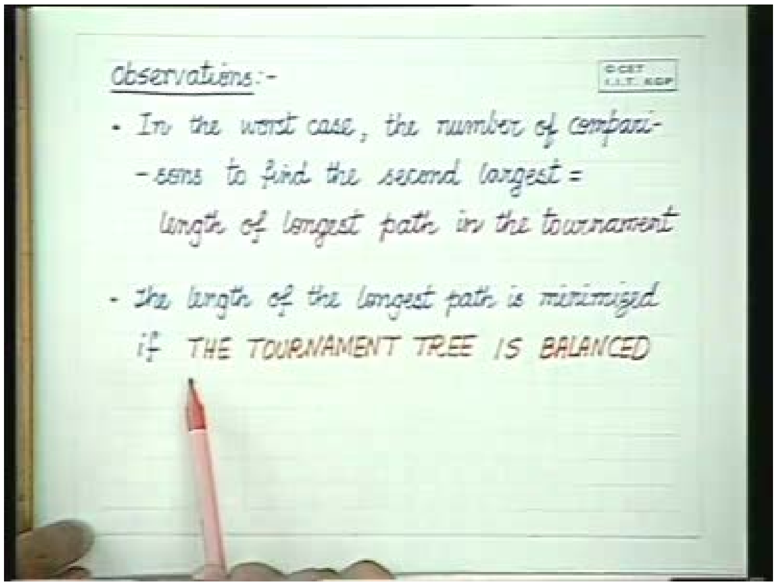
If we solve it, likely solved our maximum of n numbers, the program which we just swapped previously was something like this. Isn’t it? We went along like this. This was sorry this was not the tournament. The tournament was cube, even this was compared with this and this was compared with this. And this tournament if you set up a tournament like this then these are the elements which are to be compared because they had lost out to the maximum. If it is like this then only these elements have to be compared this one, this one and this one. Leaves not, it is not necessary that leaves have to be compared. Now can you tell me what decides the number of elements that have to be compared to find the second largest. The path is decided by the path of the winner. The length of the path of the winner decides that is how many tournaments the winner has to play besides how many opponents that person plays with. So you have to minimize the number of tournaments that the winner has to play. Now we don’t know the winner apriori therefore in the worst case, if you put in a tournament like this and the winner was here then all the others would have to be compared.

(Refer Slide Time 13:53 min)



Therefore the length of the path of the winner in the tournament decides it. And now the length of the path of the winner can be the longest path in the tournament. Therefore in the worst case, the longest path in the tournament can decide the number of elements which are going to be compared to find the second largest. Therefor your idea of minimizing the set to be compared for the seconds largest is to set up a tournament in which the length of the longest path is minimum. Therefore we come to the solution that is the observation in which the solution is to be obtained.

(Refer Slide Time 14:54 min)



Observations:-

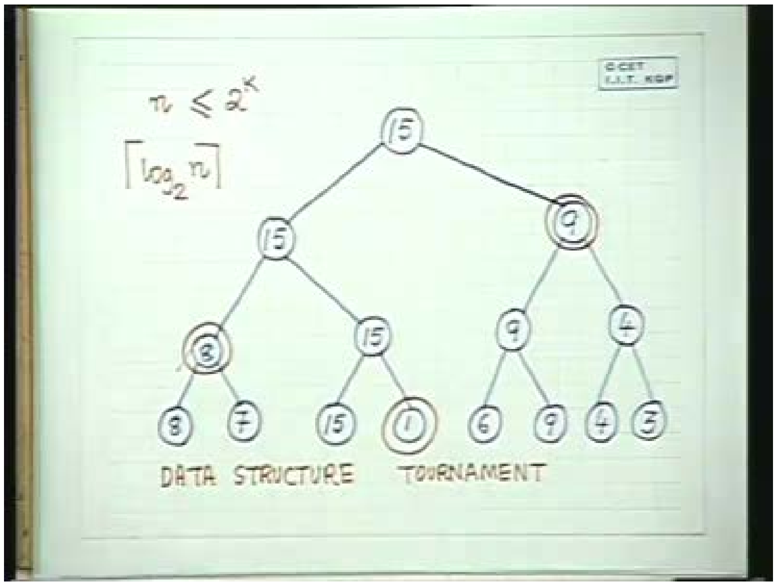
- In the worst case, the number of compari-
  -sons to find the second largest =
  length of longest path in the tournament
- the length of the longest path is minimized
  if THE TOURNAMENT TREE IS BALANCED

In the worst case the number of comparisons to find the second largest is equal to the length of the longest path in the tournament, in the worst case because that maybe the path of the winner. Therefore the length of the longest path of the tournament is minimized, must be minimized and we say that it is minimized when the tournament tree is balanced. What do we mean by balanced? There are many meanings of balanced but for our tournaments structure, let us argue it this way. If the set up pair wise comparison at the leaves and then pair wise comparison, if we continue to do the pair wise comparison as it done in a standard tournament, you will see that if there are 2 to the power n numbers they are like 8, for example if there are only two players then there will be a only one. If there are 4 players, the length of all the parts will be 2. If there are 8 players then you can set up the tournament by playing pair wise so that the length of the path will be 3. That is you can make n players play in such a way but the length of the path will be k such that k is the smallest integer which is just greater than such that 2 to the power k is just greater than or equal to n.

In other words the length of the path, the length of the longest path can be reduced to this value, the ceiling that is the largest integer or you round off this into or you truncate plus 1, truncation of log n to the base 2 plus 1 which is called the ceiling that it is the next integer which is closest to n. So if you balance, we will come to balance trees, balanced tournaments much later on but let us understand that if we play pair wise here and then pair wise here and then pair wise here and continue, we are just going to get the minimum length of the longest path. And if we get the minimum length of the longest path then all we have to do is compare amongst the losers, among the losers who lost to the winner that's all we have to do. So this is the idea of solving the problem.

Now we have to get it down to programming. That is are we going to store up this structure because unlike the maximum and minimum where we could just put them into arrays very simply, we do not know the winner till the complete tournament is finished. Therefore only when the tournament is over, can we know who is the winner and only based on that can we decide on who are going to be compared in the next phase. So we have to set this structure out that is we have to implement the result of the tournament somehow.

(Refer Slide Time 18:45 min)



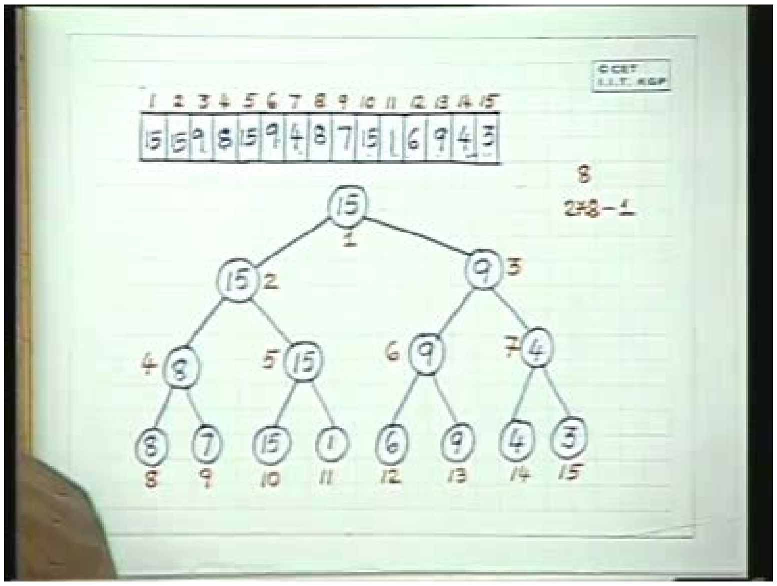
We not only have to store these n numbers, we have to store this structure. That is now we have the data structure tournament which was not so obvious in the previous case but here we are now reached a situation in which we have to implement this tournament structure which we conceptually drew up with circles and links. So the question is this is the data structure which we have in mind, the next phase is how do we implement it. So we come to the next question, how to implement our tournament structure. We can put our read element, these are the elements 8 7 15 1 6 9 4 3 into an array and then we can set up the tournament. Without going into much more detail, I will quickly go into the solution because I do not want to start discovering how to set up the tournament but if you have some idea about numbering then you will see suppose I did a numbering like this 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15. Let's see this numbering, the comparison now suppose my read in elements are put in array locations 8 9 10 11 12 13 14 15. Then the comparison of 14 and 15 goes into 7. The comparison of 12 and 13 goes into 6. The comparison of 10 and 11 goes into 5. The comparison of 8 and 9 goes into 4. The comparison of 4 and 5 goes into 2. The comparison of 6 and 7 goes into 3. The comparison of 2 and 3 goes into 1.

So now do we have a mechanism of knowing exactly where the comparison of two elements will go into. If two elements are in locations, we have made sure the two elements which are to be compared are in consecutive locations point number one. And the result of their comparison goes into that index which is half of their index, 8 by 2 is 4, 9 by 2 is 4 if you do integer division. Similarly 10 and 11 by 2 is 5, here 14 and 15 by 2 is 7, 6 and 7 by 2 is 3, 4 by 2 or 5 by 2 is 2, 2 by 1 2 or 3 by 2 is 1. So if we put all elements here then we can easily set up the tournament structure in an array. So if we declare, if you have got n elements and we declare and we can put them out in an array of size 2 n minus 1.

We can easily build up the tournament in an array of size 2 n minus 1. So how do we build up the tournament in an array of size 2 n minus 1 is the next question. So, I will just we will just re-work the example, how do we start where do we start with, we can compare these two. So we read in the elements and we store them. Suppose you are reading in 8 elements like you did here, we store them from locations n to 2n minus 1. So when we are reading 8, we will start storing them from 8 to 2 into 8 minus 1. We will just store them first. So we will read in these elements and we store them 8 7 15 1 6 9 4 and 3 and then what we can do is we can compare these two and put it here, compare these two put it here, compare these put it here, compare these two put it here. And then do from here to here, compare these two put it here, compare these two put it here and then compare these two and put it here. A much more easy way of doing the whole thing is if you start backwards.

You compare these two and put the larger here, compare these two and put the larger here, compare these two and put the larger here, compare these two and put the larger here and just continue compare these two and put the larger here, compare these two and put the larger here, compare these two and put the larger.

(Refer Slide Time 24:41 min)

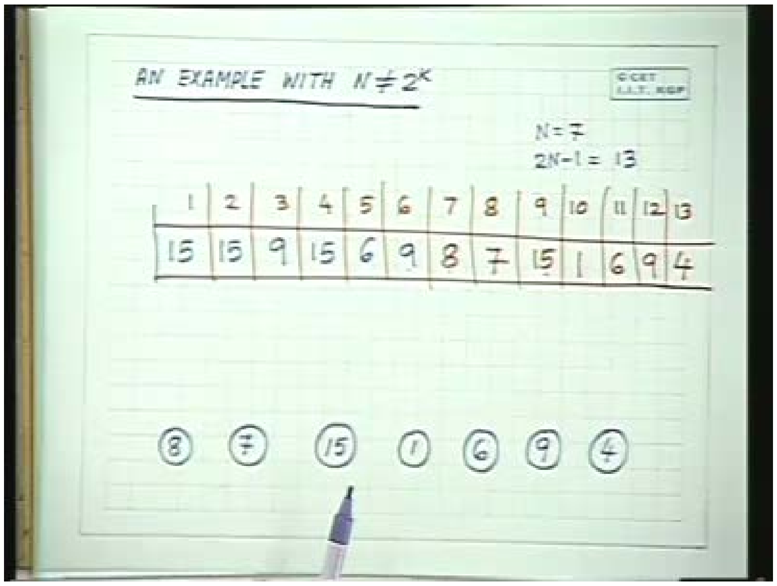


So for i equal to 2 star n minus 1, right up to here in one loop you can simply set up the tournament. Is that okay? So now this is an example with 8 elements which is the power of 2, let us just do another example which is not a power of them. So we have got 1 2 3 4 5 6 7 elements. So n is equal to 7, so 2 n minus 1 is equal to 13. So in our array we have got 13 locations, 1 2 3 4 5 6 7 8 9 10 11 12 and say 13. The first thing that we do is store the numbers from 7 to 13, 8 7 15 1 6 9 4 and then starts backwards, 9 and 4 will be put in 6. So 9 is put here, 1 and 6 is going to be put in 5, these two, these two are going to be put here.

Now note since it was an add number, there was a problem but that problem is taken care off. These two are going to be compared and this is going to be put here, these two are going to be compared and this is going to be put here.

These two are going to be compared and this is going to be put here. You can a slightly different structure compared to the tournament that we would have set up but this is also a tournament. So doing it this way will give you the full tournament structures set up.

(Refer Slide Time 26:41 min)



AN EXAMPLE WITH $N \neq 2^K$

©CET I.I.T. KGP

$N = 7$

$2N-1 = 13$

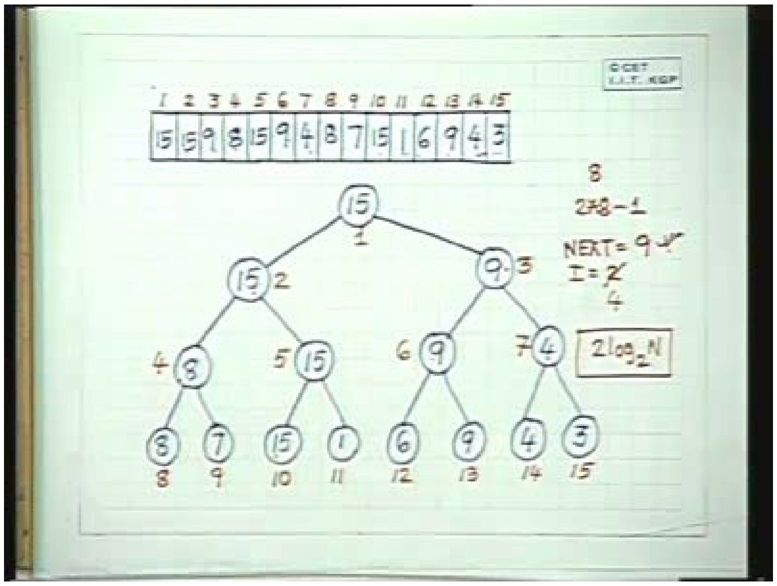
1	2	3	4	5	6	7	8	9	10	11	12	13
15	15	9	15	6	9	8	7	15	1	6	9	4

8 7 15 1 6 9 4

Now once you have the tournament structure, you know which is the largest. Now what do you do? Now how do you find out the second largest? Suppose you got more than two elements okay, then we start between two and three and you initialize the second largest with the smaller of the two. Now go to the larger of the two. If this index is two, so if this index is two then jump to 4 and compare these two. The smaller of the two must be compared with the tentative second largest. So first you start with 15 and you initialize the second largest to the smaller of these two, So you keep a variable is called next say which is initialized to 9. And index i is made two because this is larger and this is smaller. Then you make i is equal to 2 into i and so i becomes 4.

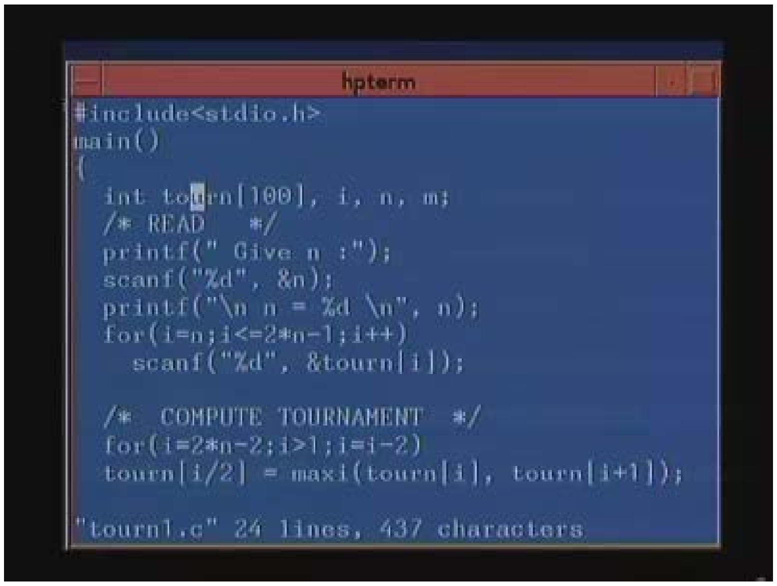
Once i is make 2 into i and i becomes 4, now compare 4 and 5 i and i plus 1. The smaller of the two is compared with this that is smaller of the two is 8 is compared with this. The index of the larger of the two is doubled to find out the other two locations. So we first compare with, we saw these two compared with this one then we came here jumped, found out the other two which are to be compared, saw this one and then again we jumped and saw this one. So we compared 9 8 and 11 and we compared these two obviously again and again and we compared these two and we compared these two. So we made more no more than some 2 into log of 2 to the base n comparisons and with these indexing and setting up of the tournament you can easily find the largest and the second largest.

(Refer Slide Time 29:18 min)



The largest will be at the head of the tournament and the second largest can be found via routine. So lets quickly see how the program of finding a tournament and finding of the largest and second largest. Yeah, I will come to that. So what we will do is we have declared an array for the tournament.

(Refer Slide Time 29:48 min)



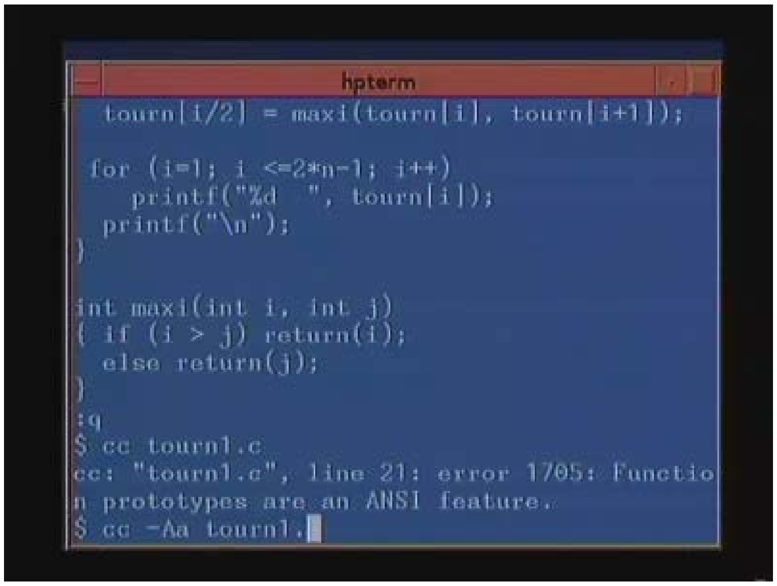
```
hpterm
#include<stdio.h>
main()
{
   int tourn[100], i, n, m;
   /* READ   */
   printf(" Give n :");
   scanf("%d", &n);
   printf("\n n = %d \n", n);
   for(i=n;i<=2*n-1;i++)
     scanf("%d", &tourn[i]);

   /*  COMPUTE TOURNAMENT  */
   for(i=2*n-2;i>1;i=i-2)
   tourn[i/2] = maxi(tourn[i], tourn[i+1]);

"tourn1.c" 24 lines, 437 characters
```

This is a program which just sets up the tournament, it does not do anything else. So, today we will just see how to, we will see the program of setting up a tournament. This will have large implication in solving other problems as well. So we will visit this tournament again in the next class.

So we read in n then we read in the n numbers and stored as I said from n to 2 n minus 1 in the array tournament. So initially the n elements are stored from locations n to 2 star n minus 1 as I mentioned. So that's why you read them all into. And what is the loop? The loop starts from the end, so its starts from comparing initializing i to 2 star n minus 2 as long as i is greater than 1 and at each step we are comparing pair wise and moving back so the loop control is i equal to i minus 2. And what you do is you put the tourn i by 2 index to the maximum of tourn i and tourn i plus 1. That's what we were doing when we did it by own hand calculations. For example first the index 2 n minus 2 and 2 n minus 1 will be compared and the maximum will be stored in that i by 2. Then I will become i minus 2. So if there were total of 8 elements then we would have started comparing 14 and 15.

The next we would have and stored it in 7. Next I would be i minus 2, so we would have compared 12 and 13 and stored the larger of 12 and 13 in 6 and continue till I continued as long as i is greater than 1 when i becomes 1 or 0 we stop. So this is how we just set up the tournament and this maximum maxi program, it just takes a very simple thing, it finds out the maximum of two numbers and returns the larger number.

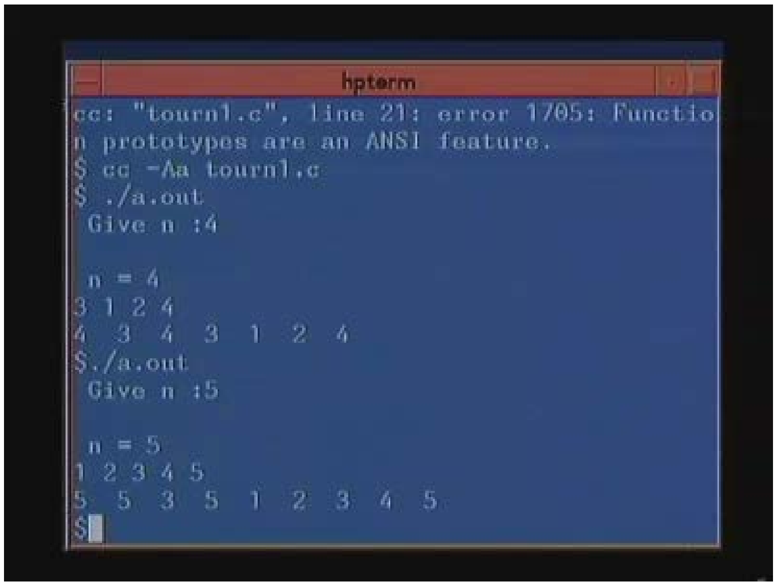
(Refer Slide Time 32:20 min)



```
  tourn[i/2] = maxi(tourn[i], tourn[i+1]);

 for (i=1; i <=2*n-1; i++)
    printf("%d  ", tourn[i]);
  printf("\n");
}

int maxi(int i, int j)
{ if (i > j) return(i);
  else return(j);
}
:q
$ cc tourn1.c
cc: "tourn1.c", line 21: error 1705: Functio
n prototypes are an ANSI feature.
$ cc -Aa tourn1.
```

That's what we did, we took two numbers and return the maximum of two numbers. So let's see how the setting up of the tournaments works. So let's give say 4 elements, so we have given 4 elements 3 1 2 4, there was stored in locations 4 5 6 and 7, 2 and 4 were compared and 4 was returned here, 3 and 1 are compared and 3 was return here, 4 and 3 are compared and 4 was put here. So this is how we got that result. If you run it again with say n equal to 5 and we give a simple data 1 2 3 4 5, so from 5 6 7 8 and 9th location these values we are stored, 4 and 5 were compared and 5 was stored here, 2 and 3 were compared and three was stored here, 5 and 1 was compared and 5 was stored here, 5 and 3 was compared and 5 is stored. So whatever you give, you are going to get back the value of the tournament.

(Refer Slide Time 33:49 min)



hpterm

```
cc: "tourn1.c", line 21: error 1705: Functio
n prototypes are an ANSI feature.
$ cc -Aa tourn1.c
$ ./a.out
 Give n :4

 n = 4
3 1 2 4
4  3  4  3  1  2  4
$./a.out
 Give n :5

 n = 5
1 2 3 4 5
5  5  3  5  1  2  3  4  5
$
```

We will stop here today for you to just digest what we have done to the tournament. Try and work out the program to now, given such a tournament data structure how we are going to find out the second largest. And we shall examine, we shall revisit the tournament next day see the exact code for finding out the second largest and see the ramification, see what else we can do with this. And we shall visit this in more details because though very simple, this structure has got lot of very interesting properties and helps to solve some of the core problems much better than what we could have imagined then to do.