

**Programming and Data Structure**  
**Dr. P.P.Chakraborty**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture # 07**  
**Data Structuring: Case Study - III**

We shall continue our discussion on the tournament data structure. And in the previous two classes, we had studied the following two problems the max and min of n integers and the max and the second max or the largest and the second largest of n numbers.

(Refer Slide Time 01:42 min)



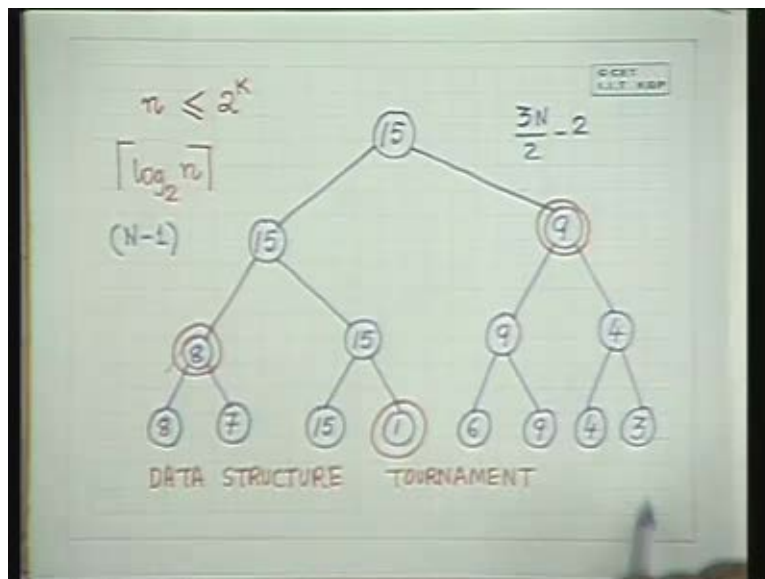
And we saw that to solve these two problems, we imagine out our structure of a comparison tree which we called a tournament structure. And the tournament structure was only implicitly used solve the max min problem but it was explicitly used to solve the maximum and second maximum problem. And we shall also see how it can be used to solve the sorting problem, also much more efficiently than the traditional techniques. So, that's what we discuss today in more details, the tournaments structure. First, a quick revision. The tournament as we said in the previous tree is a comparison tree. The leafs of the tournament are the numbers to be compared and for which we require to find the maximum and second maximum or the maximum and minimum or possibly sort them. And we compare the numbers pair wise and then compare the results pair wise till we build up what we call a tournament tree or a comparison tree.

Now the maximum and minimum problem can be solved in the following way. These are the winners of the comparison at the first level. Using these winners if you compare the winners, we will get the maximum and if we compare the losers for example 7 here, 1 here, 6 here and 3 here and a comparison of the losers will give us the minimum. And that would solve the problem in approximately  $3N/2 - 2$  comparisons for even and  $3 \times (n/2) - 3$  by  $2 \times (n/2) - 1$  comparisons for odd unlike the traditional one which we would have done when we converted

the standard maximum algorithm. In the case of the maximum and the second maximum of the largest and the second largest problem, the inside that we obtained was for such a tournament structure, the second largest number must be that which has lost out to the largest in the comparison tree or in the tournament structure. Therefore in this particular example, the second largest would lie between 9 8 and 1. So if we compare 9 8 and 1 and found out the largest of these 3 numbers only then we would have founded the second largest number. And this comparison, the number of elements to be compared to find the second largest number was the sealing of long n to the base 2 which is the height of the tree or which is the length of the longest path in the tree in the worst case scenario.

Therefore finding the largest number would have taken n minus 1 comparisons and finding the second largest number would have taken only log n to the base 2 comparisons sealing obviously. Therefore this is the much more efficient way to find out the maximum and the second maximum of n numbers. Then we went into the issue of implementing the tournament because we have to find out this second largest, therefore we have to construct this data structure and implement it in our implementation language called C. Our language does not provide an in built data structure for a comparison tree from which you could have manipulated that data structure. It provides you array's, it provides you integers, characters and other things but it does not provide you a structure called a comparison three.

(Refer Slide Time 04:55 min)

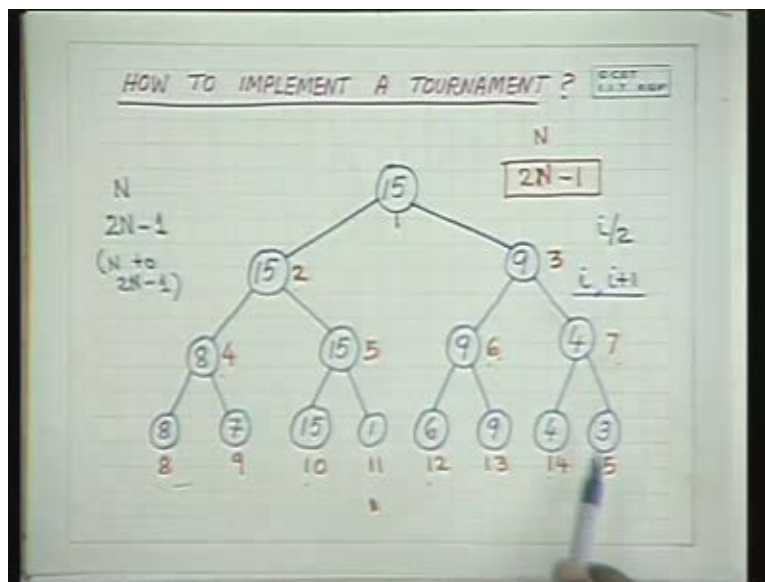


So this data structure of a comparison tree has to be built using the available data types and structures available in the particular language. If there was a language which provided you a comparison tree with functions built tree, with function getnext element, with functions get the second largest element then it would have been matching easier for you to solve the problem. Therefore data structuring has been to be done on the language at hand and in C we will now discuss how to do it.

We had seen the previous day that a simple numbering technique gives us an easy way to implement a tournament tree on to an array data structure. For example if you number this 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 that is if there are n numbers then we would require 2 n minus 1 elements of the array. And the numbers which we are reading in would have been stored from n to 2 n minus 1 in these locations. For examples if there are 8 numbers here and we are going to store it from location 8 to 15.

Now the comparison would come as follows, 8 and 9 we would be compared and the larger one would be stored in 4, 10 and 11 in 5, 12 and 13 in 6, 14 and 15 in 7. Similarly, 6 and 7 in 3, 4 and 5 in 2 and similarly the comparison of 2 and 3 in 1.

(Refer Slide Time 07:14 min)

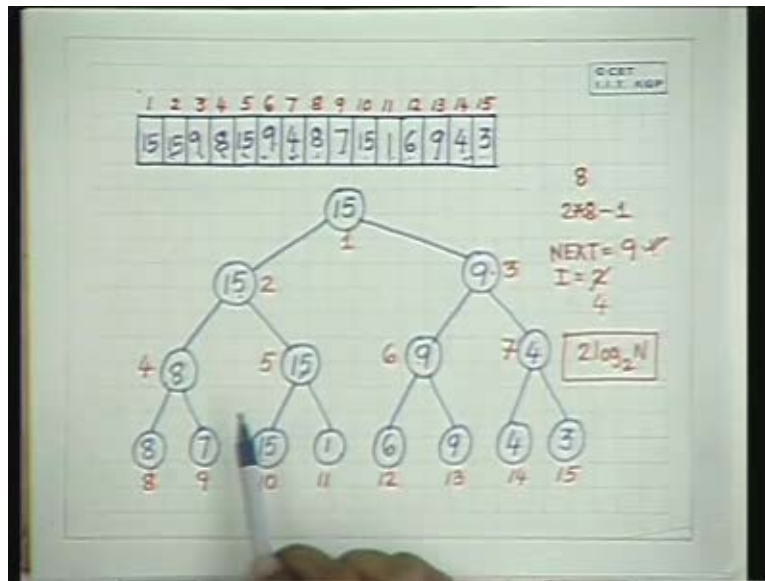


Therefore the comparison of the  $i$ th and the  $i$  plus oneth index number would be stored in the  $i$  by twoeth location by when we do integer division. So  $i$  and  $i$  plus 1 would be stored in  $i$  by 2 that was the idea. And in order to implement it very easily to write a good simple program, we could start from the end that is what we saw. We first compare these two and put it in 7, these two and put it here, these two and put it here, these two and put it here. Then we come back and compare these two, put it here, these two put it here and then 3 and 2 compare and put it here. That's also what we saw in this example where we constructed out the array and we stored out the elements from the location  $n$  to  $2n$  minus 1 8 7 15 1 6 9 4 and 3 then we compared from the end 4 and 3 and stored it in the location 7.

6 and 9 we stored in the location 6 because 12, half of 12 is 6 then those in location 10 and 11 we stored in location 5, those in 8 or 9 we stored in location 4 and we continued those in 7 and 8 we stored in location 3 sorry those in 6 and 7 we stored in location 3, those in 4 and 5 we stored in 2 and comparison of 2 and 3 we stored in 1. Therefore the algorithm to build up a tournament was a very simple algorithm. Next we were interested in finding out the second maximum. The maximum element will lie at the head of the tournament. The second maximum you have to compare between those who lost to the maximum.

So first let us see how we will get to the indices of these two elements. So you can start with 2 and 3 and then you find out which them is smaller and you store the smaller one, initialize the current minimum, current second largest to the smaller of the 2. And then you go to the larger of the two and see its index. You double the index to get to the other element which may have been compared with the largest elements say 15. And here the smaller of the two is 8. So this element has to be compared with the tentative second largest which is stored in a variable. So this will be compared with tentative second largest and the comparison of these two elements would give me the tentative, the new tentative second largest.

(Refer Slide Time 09:52 min)



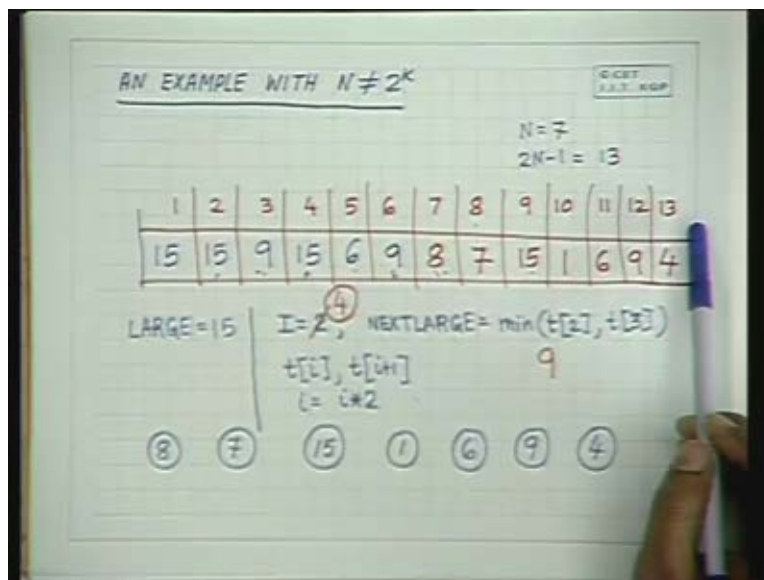
And again I will double this location and come to these two positions and then again the smaller of the two, I will compare with the tentative second largest. So let's work it this out on an example and we will see how we will be able to get it. Let us take this one, we have got 7 elements 8 7 5 15 1 6 9 and 4. And how did we build up the tournament? 9 and 4, 12 and 13 is stored in 6. 10 and 11 is stored in 5, comparison of 8 and 9 that is 7 and 15, the locations 8 and 9, the numbers are 7 and 15 and the comparison is stored here. The comparison of these two elements in 6 and 7 is stored here. The comparison of 15 and 16 in location 4 and 5 is stored here and the comparison of these two is stored here.

So now we have found out our largest is equal to 15. Now we proceed to find out the second largest element. Now how do we do that? We initialize I to 2, let's us assume for the time being that there are more than one element, if there is only one element then the problem is absolutely simplified. And we nextlarge element that is the next largest element, we initialize to minimum of the tournament, let us say let us call it  $t_2$  and  $t_3$ . So this is what we initialized I to 2 and the nextlarge is telling into initialized to minimum of 2 and 3. So in our example, the next largest element here will be initialized to 9. So the initial value of this will be 9 because the minimum of  $t_2$  and  $t_3$  is 9. Next we are at 2, so now we compare which of these two elements is larger, I  $t_2$  is larger than  $t_3$ . So we will compare  $t[i]$  and  $t[i + 1]$ . Now whichever is larger, if  $t[i]$  is larger we will double this index.

If this is larger we will double this index. In this case this one is larger, so we double the index  $i$  to  $i$  into  $2$  and  $i$  was, which was to will now become  $4$ . And then once we come here, again we do, we find out the smaller of the two. The smaller of the two is  $6$ . Now this is a candidate for the second minimum. So the smaller of the two that is the one which has lost to the largest element is the candidate for the second minimum. So this candidate is compared with the tentative second minimum. Once it is compared with the tentative second minimum, the tentative second minimum is updated. In this particular case it is  $6$  and  $9$ , so it remains to be  $9$ .

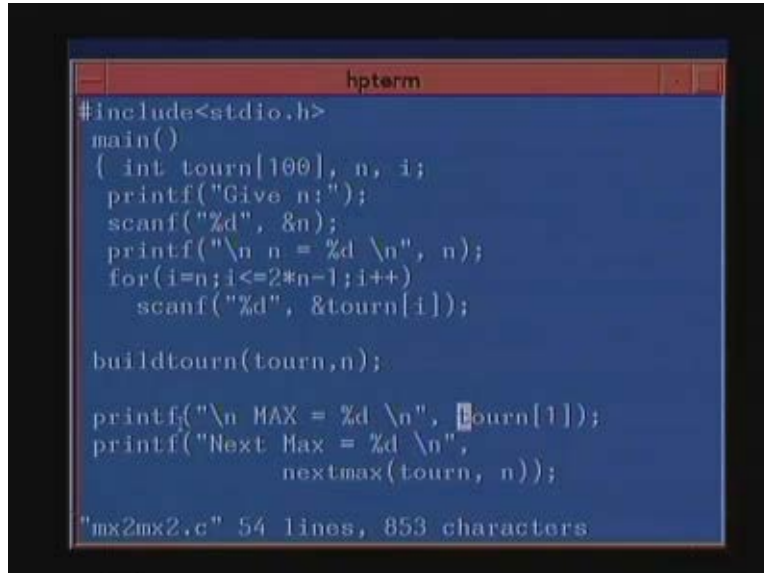
Now again we are at  $i$  and we will think of doubling of this index. So we are now **at 4 and** at index  $i$  and  $i$  plus  $1$ , the larger of the two is here which is the maximum element. So we double  $4$  and we come to  $8$ . Therefore the larger of the two that is the maximum element must now be between  $8$  and  $9$ . So again we find out the element which is the larger one, is this one and we find out the element which lost to it, that is  $7$ . So since this element lost to it, this is compared with the tentative second largest and this tentative second largest remains  $9$ . Again  $i$  is now the larger of the two is  $15$ , so  $i$  is  $9$  this is doubled, one this is doubled it becomes  $18$ .

(Refer Slide Time 14:29 min)



Now since it crosses to  $n$  minus  $1$ , we can stop because we have reached all the elements. So this way once we cross this up out beyond the range of  $2n$  minus  $1$ , we can stop. So this is the idea of producing the second largest element, this is the complete implementation. Any questions? We will now go and see how the program looks like to build the tournament and to print the, to find out the next largest element. So let's go and see the program. This is the complete program, it declares an array called `tourn` of size `100`,  $n$  is the number of elements to be read,  $i$  is the array index. Then it asks to give the value of  $n$  then it reads in  $n$ , it prints the value of  $n$  and this is where it reads in the elements of the array. And as I mentioned before, it reads in from the index  $n$  to  $2n$  minus  $1$ , these are the places where the  $n$  elements are reading. So it reads into tournament  $i$  from  $i$  equal to  $n$  to  $2n$  minus  $1$  then it calls a function `buildtourn` which builds the tournament, it passes the array index that is the minimum of the array and it passes the size of the array that is the number of elements  $n$ .

(Refer Slide Time 16:25 min)



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

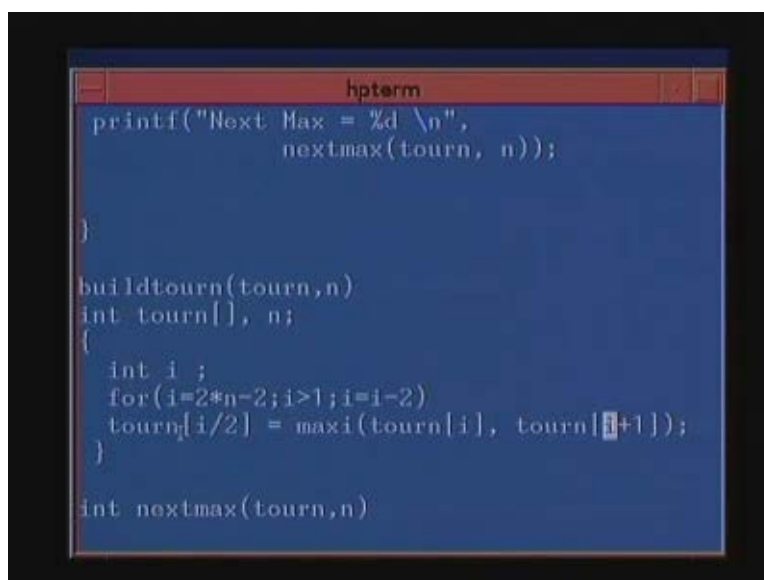
  buildtourn(tourn,n);

  printf("\n MAX = %d \n", tourn[1]);
  printf("Next Max = %d \n",
        nextmax(tourn, n));

"mx2mx2.c" 54 lines, 853 characters
```

And after the tournament is build up tourn one contains the largest element, that's quite obvious. And then to find the next largest it calls another function called nextmax in which it passes the tournament and that there are n base elements in the tournament, those are tournaments of size n minus 1. So this is the complete program. Now we will have to go and see the two functions buildtourn and the nextmax. We had seen buildtourn in the previous class and now we will see its version as a function.

(Refer Slide Time 18:04 min)



```
hpterm
printf("Next Max = %d \n",
      nextmax(tourn, n));
}

buildtourn(tourn,n)
int tourn[], n;
{
  int i ;
  for(i=2*n-2;i>1;i=i-2)
    tourn[i/2] = maxi(tourn[i], tourn[i+1]);
}

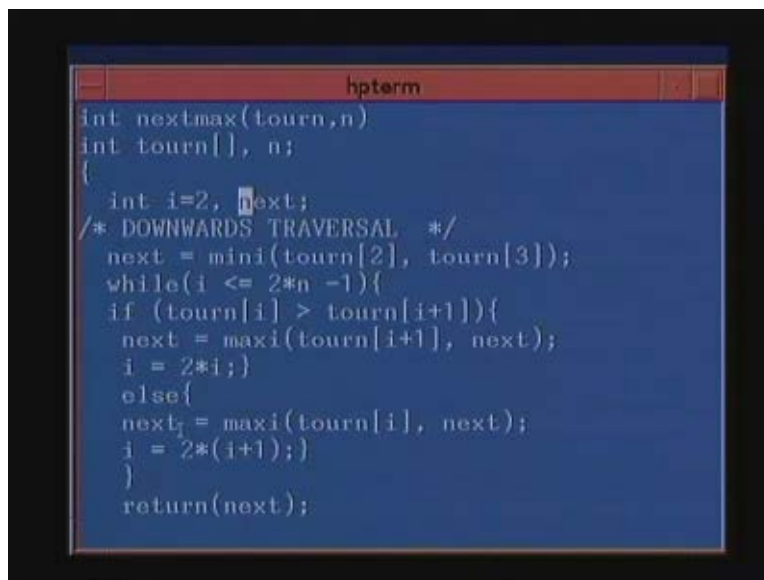
int nextmax(tourn,n)
```

In a function we have passed to the array tourn and the number of elements n. Here as I had mentioned in a previous class possibly but we will read like this that when you declare an array

here, you don't have to declare the size of the array. You have to just declare the array name, the type of the array and that it is a one dimensional array. There are other ways to do it and we shall come to details of declaring arrays in functions in an subsequent class. But tentatively if you want to pass an array, you have to just pass the array name and here you have to declare what type the array is and that it is an array of one dimension. You need not specify the size of the array here. so we declare a tentative variable and from the end of the array, we were just comparing two elements and put in, the idea was compare element  $i$  and  $i$  plus 1 and put the maximum of the two in the index  $i$  by 2. And we do this from the end of the array till  $i$  is as long as  $i$  is greater than 1 and while we do this we will be easily be able to build up the tournament. We had seen this example in the previous class and we saw that tournament can be properly build up using such an example.

Now we come to the question of next max. This is the function for next max. The function for nextmax takes in tourn, the array and the number of elements and it declares as mentioned before intl tourn one dimensional array, n.

(Refer Slide Time 18:58 min)



```
hpterm
int nextmax(tourn,n)
int tourn[], n;
{
    int i=2, next;
    /* DOWNWARDS TRAVERSAL */
    next = mini(tourn[2], tourn[3]);
    while(i <= 2*n -1){
        if (tourn[i] > tourn[i+1]){
            next = maxi(tourn[i+1], next);
            i = 2*i;}
        else{
            next = maxi(tourn[i], next);
            i = 2*(i+1);}
    }
    return(next);
```

It declares two integers  $i$  which we said was initialized to 2 and the next maximum which is the tentative current maximum. And before we proceed to the downwards traversal, we initialize the next largest to the minimum of tourn [2] and tourn [3] as we mentioned. And then we continued in a loop, so this is a while loop which starts here and ends here. And we continue as long as  $i$  is less than or equal to  $2n$  minus 1 because if  $i$  exceeds  $2n$  minus 1, we have finished. And there are two cases now, if tourn  $i$  is greater than tourn  $i$  plus 1 and if tourn  $i$  is less than or equal to tourn  $i$  plus 1.

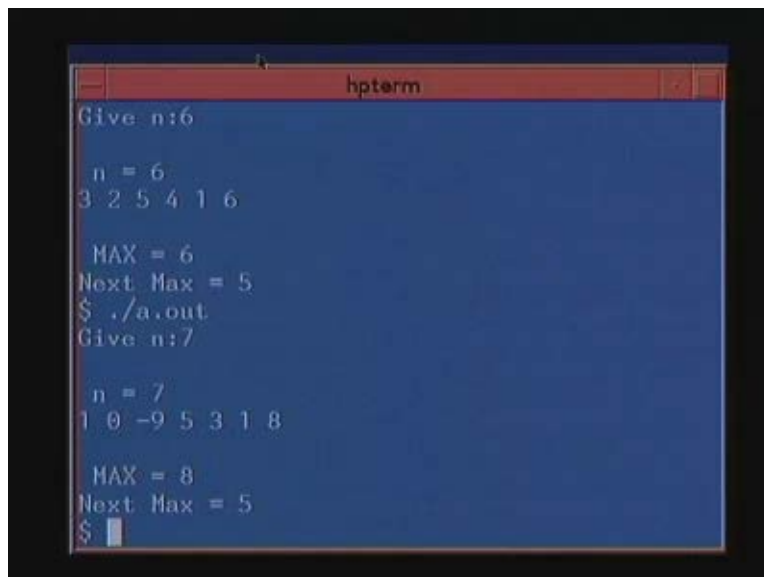
If tourn  $i$  is greater than tourn  $i$  plus 1 then obviously the smaller of the two is tourn  $i$  plus 1. say this if tourn  $i$  is greater than tourn  $i$  plus 1, the smaller of the two is tourn  $i$  plus 1, this will be compared with tentative maximum next and the larger of the two will be stored in the tentative maximum next. Maxi is a function which compares two numbers and returns the maximum. So

tourn i plus 1 which is the smaller of tourn i and tourn i plus 1 is compared with next because this was the one who lost out to the maximum element. So the larger of the two is kept in the tentative max and since tourn i was larger than tourn i plus 1, the next maximum will lie between twice i and twice i plus 1.

Therefore i is made into twice i and the loop continues else, this is the else part. Else means what? tourn i is less than or equal to tourn i plus 1 that means the maximum element is possibly now tourn i plus 1. So, the one which lost to the maximum element is in tourn i. So this is compared with the second largest, with the current next largest element and the larger of the two is and that updates the next largest element value. And since we again have to double our index, this time we have to double or index with 2 star i plus 1 because now it is in the else part of this if which means the tourn i plus 1 is the larger of the two and once we double the index, we have to double it here.

So this ends our condition and once we come out of this loop, what all we have to return is the value of the next largest element. So this is the function get next max where we are calling this function in the printf statement, mind you in the printf statement also we can put in a function name because every statement and every function returns the value and this will return the value of the next maximum.

(Refer Slide Time 22:48 min)



```
hpterm
Give n:6
n = 6
3 2 5 4 1 6

MAX = 6
Next Max = 5
$ ./a.out
Give n:7
n = 7
1 0 -9 5 3 1 8

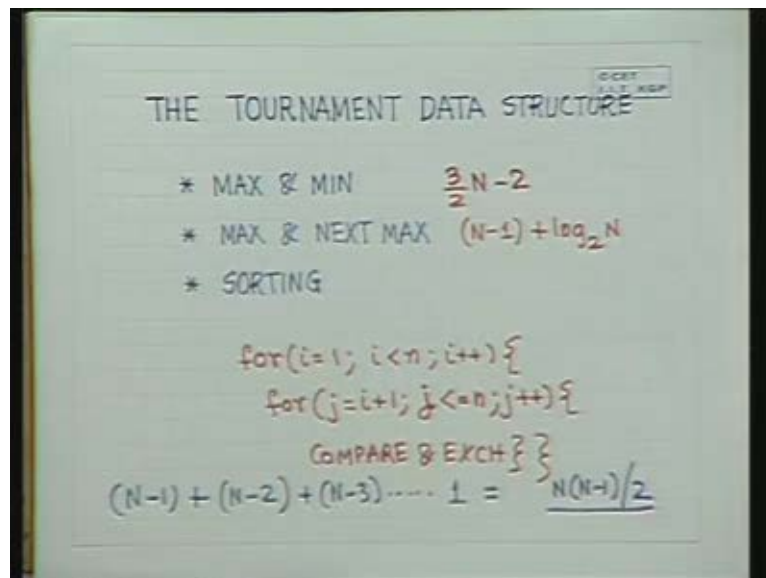
MAX = 8
Next Max = 5
$
```

So now we can compile and run it. What was this? This was mx2mx2. 3 2 5 4 1 6, so the maximum is 6 and the next maximum is 5. Maximum is 8 and the next maximum is 5. So this way using this program, we can find out, we can build the tournament, so we now know how to build the tournament and we now know how to traverse downwards to find out the next maximum in a much more efficient manner than we would have done it in the traditional way of comparison which, the example of which we have seen before. So we have seen now how to find out the largest and the second largest element in a much more efficient way in time which is  $n \text{ minus } 1 \text{ plus } \log_2 n$  and this one we saw it is something like this.



Now we come to a very interesting example, the example of sorting. The traditional double sort or the exchange sort which all of us do have got two loops. Isn't it? One is for (i equal to 1; i less than n; i plus plus) and for (j equal to i plus 1 to j less than equal to n; j plus plus) and then you compare and exchange. This is the program to do bubble sort or exchange sort. And how many comparisons and exchanges do you do in this sorting routine? you do, this loop goes from 1 to n minus 1 and this i plus 1 to n, so when this is 1 this does n minus 1, when this is 2 it does n minus 2. So the total number of comparisons is n minus 1, when i is 2 which is... (Student: Question). This is N into N minus 1 by 2. So many comparisons we are going to do in all the traditional sorting examples that we have done.

(Refer Slide Time 25:28 min)



Now let us see whether this tournament which helps us to find the largest, the second largest in log n time. Can it help us to find out the elements? And we shall see how easily we can do that. Let us go back to this example. So this is the tournament which you have built up and the largest element of the tournament we saw lies here. So the largest element can be found from tournament one, so tour 1 is contains the largest element in the tournament. Now once you have build up the tournament, you can also find out the smallest element because the way we did our largest and smallest, we can also find out the smallest element. So let us say the minimum element here can be found out and is one. So there is no numbers smaller than one here. So let us define a value called low equal to min minus 1. That means in this case it is 0 and in every case it will be smaller than the smallest element. And now let us revisit what we were going to do to find out the second maximum. What we did was we found out which of these two is smaller and then we kept dot tentative second maximum.

I will just show you some data structuring that we can do while going down, we were going downwards isn't it while finding the second maximum. Now let us see what data structuring we can do to go downwards and then we will see it becomes very easy to find out the second maximum in another way. slightly longer way that it will help us as we mentioned before if we take longer way out to do one thing, the subsequent things maybe done much more easier. So

this one we replace by low which is 0. We compare which is larger of the two and when we go down, we just convert all numbers which are having the largest value to low. So we come here, compare these two, the larger of the two we assign to low. Again we double this up, the larger of the two we assigned to low. Again we doubled this up, this one, the larger of the two we assign to low.

So what we have done is we have gone down, we have not found the second maximum, we have just gone down like this and may come up to here and when we came here, we assign all the numbers which were the largest element to zero. Now if we rebuild the tournament, here what will we get? The second largest will come. If we rebuild the tournament, the second largest will come here but do we need to rebuild the tournament from the beginning. We don't, we just need to go up this link now. If we go up this length, we can rebuild the tournament because this part is done, we don't need to do it anymore, this part is done you don't need to do anymore.

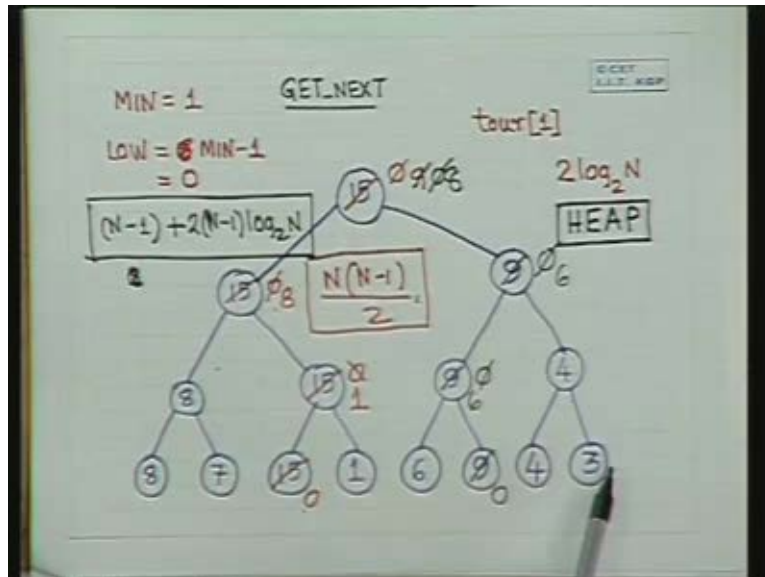
All we need to do is to rebuild the tournament, we don't have to build the tournament like this as we were doing. We can again go up along the path which we came down and just compare these elements. For example let us look at what we will do. We will compare 0 and 1 and put 1 here, we will compare 1 and 8 and put 8 here, will compare 8 and 9 and put 9 here. And once we go up, the second largest will pop up that is the current largest will come up here and tour 1 will contain the second largest element. So in  $\log_2 N$  steps where we made one comparison each, we went down and in another  $\log_2 N$  steps we went up. So in  $2 \log_2 n$  steps, we got the second largest. So we spend a little more of effort but once we have been done this, the principle of induction follows. The third largest can again be formed in another  $2 \log n$  steps. Isn't it?

So how do we find out the third largest? Go down and replace by zero's again. This becomes minimum compare the two, the larger of the two is this. This becomes min, jump, compare the two, the larger of the two is this, this becomes min. jump, compare the two, the larger of the two this becomes min, so downwards traversal is done up. Up means, what? Double half the indices you will go up, double the indices you will go down, half the indices you will go up. So, again 6 and 0 6, 6 and 4 6, 6 and 8 8. So again the third largest comes up. So next time you go down and up, you will get the fourth largest. So if we call, if we now define a function `get_next` and what does this function get match do? It goes down and up and after finding the tournament if we call `get_next`  $n$  minus 1 times, we are going to get the numbers in sorted order one after another. And how much time are we going to take to do that, to build up the tournament? We took so much time. and for each sets step going up and down in `get_next`, we are going to take  $2 \log_2 n$ . so the total amount of time that we are going to take is  $2 n \log_2 n$ . Compare this with the value  $N$  into  $N$  minus 1 by 2, when  $N$  is large which one is bigger? This one is bigger. This is much bigger this is, this is of the order of  $N \log N$  and if you do this divided by this when  $n$  tends to infinity, you are going to get zero.

Therefore this will take much less time than this, when you are going to have reasonably large value of  $N$ . As  $N$  increases, the effect of this becomes more pronounced compared to the effect of this. Therefore using this tournament data structure and using a very similar idea to the next maximum of which we go down and go up, you can get a sorting algorithm which is much better than your exchange sort or bubble sort. So this is the idea of sorting using the tournament structure. We shall see how we can improvise the tournament structure much later on because we

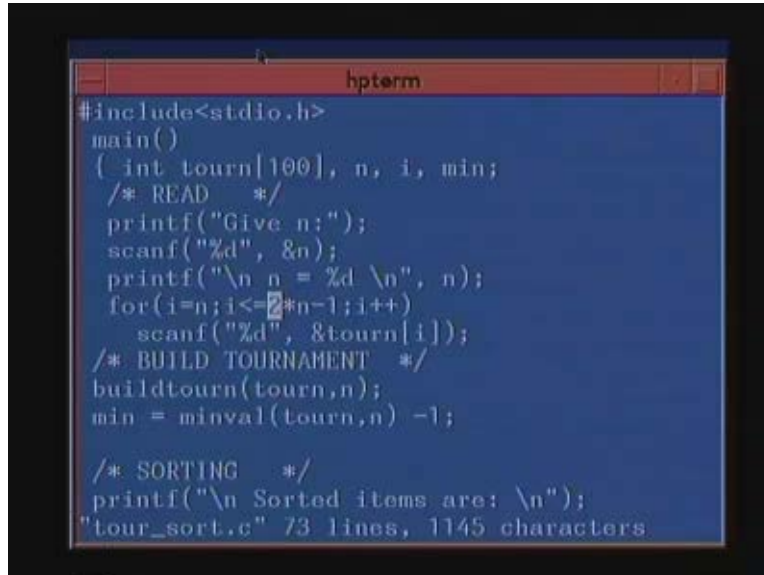
can build up a tournament structure with only n elements with an array of size only n. We have build up a tournament structure with an array of size 2 n minus 1. We can build up a more complicated tournament structure with an array of size n. Their again we can find out the largest and second largest quite efficiently but that algorithm is much more involved.

(Refer Slide Time 33:48 min)



And if you look at any text book, you will see that data structure is called a heap. A heap is a data structure which builds up a binary comparison tree of size, with an array of size n elements only and it can, it is capable of finding out the largest and the next largest by a similar sort of traversive. It is slightly more involved because it uses less space but it will also take a very similar amount of time. And that sorting algorithm will be called heap sort maybe you have heard of this name before. So, we will come to these algorithms much later on. This, in this introductory idea is where we try to introduce the concept of what is data structuring. I try to emphasize how data structuring can help us to solve a number of problems. So before concluding we will just have a quick look at the code for tournaments sort. So this is the code for tournament sort, this is very similar to the code for finding out the second maximum.

(Refer Slide Time 34:55 min)

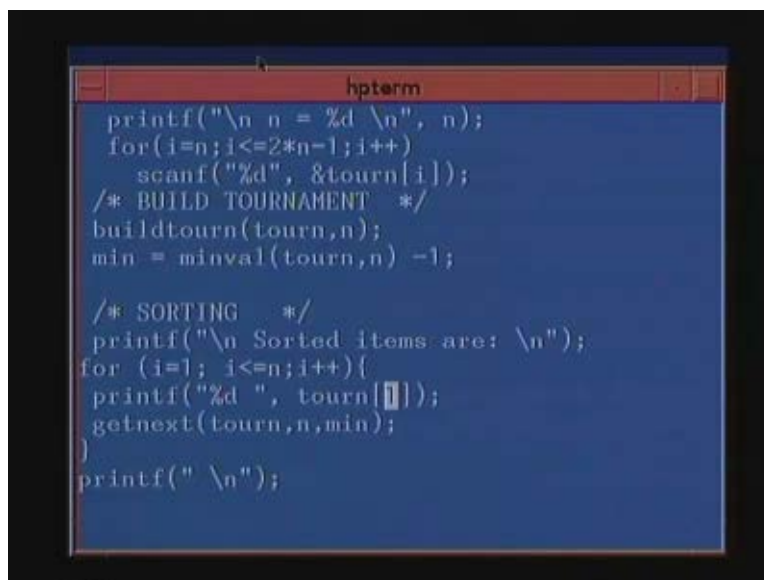


```
hpterm
#include<stdio.h>
main()
{ int tourn[100], n, i, min;
  /* READ */
  printf("Give n:");
  scanf("%d", &n);
  printf("\n n = %d \n", n);
  for(i=n;i<=2*n-1;i++)
    scanf("%d", &tourn[i]);
  /* BUILD TOURNAMENT */
  buildtourn(tourn,n);
  min = minval(tourn,n) -1;

  /* SORTING */
  printf("\n Sorted items are: \n");
"tour_sort.c" 73 lines, 1145 characters
```

Here you read in the n, you read in the n elements from n to 2 n minus 1 just like before, you called buildtourn just like before and here you find out the minimum element. So I just put in a routine which finds out the minimum value but you can integrate it with the building of the tournament, I just did not do it for purposes of easy explanation and you subtract one from it so that this value becomes smaller than the smallest element available. And then we start our sorting and in the sorting we put in a loop which works from 1 to n. The first thing you do is we print tourn 1.

(Refer Slide Time 35:34 min)

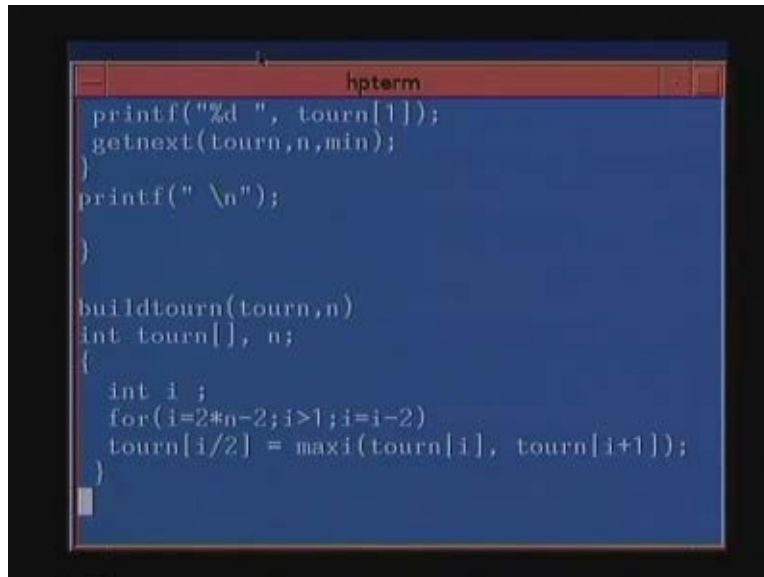


```
hpterm
printf("\n n = %d \n", n);
for(i=n;i<=2*n-1;i++)
  scanf("%d", &tourn[i]);
/* BUILD TOURNAMENT */
buildtourn(tourn,n);
min = minval(tourn,n) -1;

/* SORTING */
printf("\n Sorted items are: \n");
for (i=1; i<=n;i++){
  printf("%d ", tourn[i]);
  getnext(tourn,n,min);
}
printf(" \n");
```

So, after we have build the tournament, tourn 1 contains the largest value and again we called get\_next. So every time we do this, we have started out so we will print this tourn 1 n times and we will be able to get the number of elements. Build tourn is identical to what we did before, so we come to get next.

(Refer Slide Time 35:57 min)

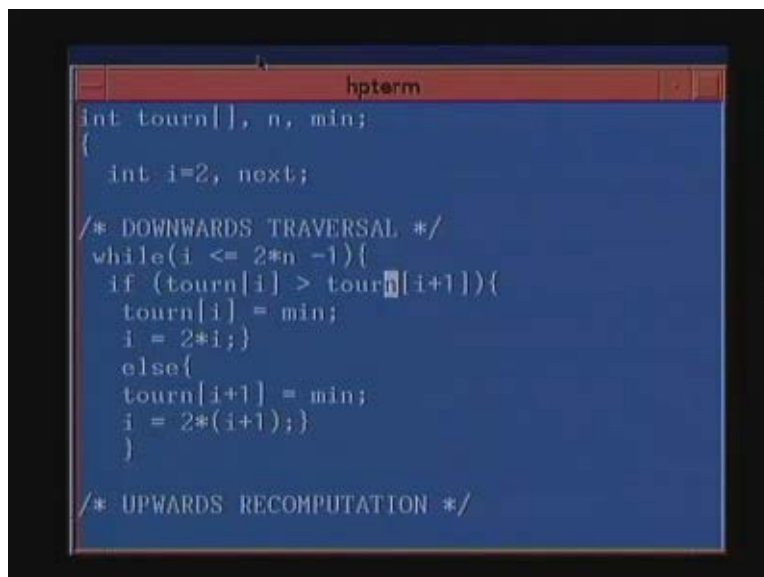


```
hpterm
printf("%d ", tourn[1]);
getnext(tourn,n,min);
}
printf(" \n");
}

buildtourn(tourn,n)
int tourn[], n;
{
    int i ;
    for(i=2*n-2;i>1;i=i-2)
    tourn[i/2] = maxi(tourn[i], tourn[i+1]);
}
```

Get\_next has two parts as I mentioned, downward traversal where we compare the two elements and the larger one is assigned the value of that list which is min and the upward computation where we recompute the values of and redefine the tournament properly.

(Refer Slide Time 36:09 min)



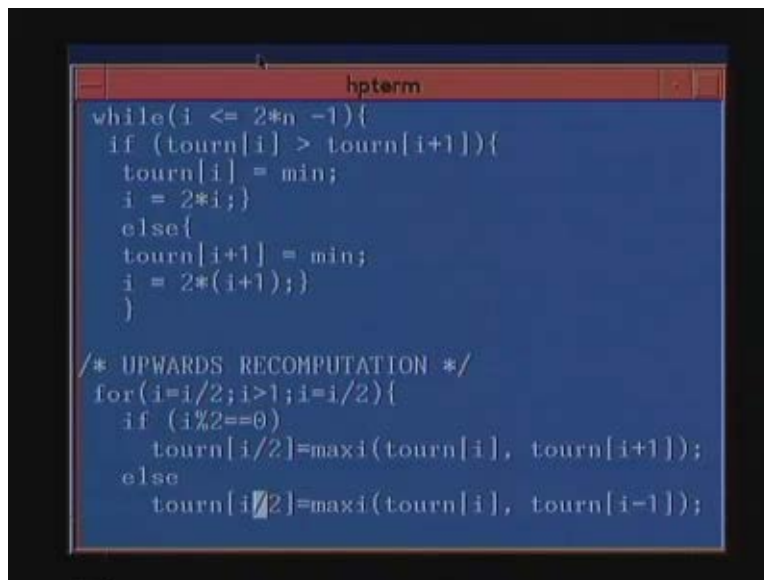
```
hpterm
int tourn[], n, min;
{
    int i=2, next;

    /* DOWNWARDS TRAVERSAL */
    while(i <= 2*n -1){
        if (tourn[i] > tourn[i+1]){
            tourn[i] = min;
            i = 2*i;}
        else{
            tourn[i+1] = min;
            i = 2*(i+1);}
    }

    /* UPWARDS RECOMPUTATION */
```

So what do you do here just like we did in the second maximum, we initialize  $i$  equal to 2. This variable is legacy variable, it's not used anywhere and  $i$  we continue as long as  $i$  is less than  $2n$  minus 1. We compare if  $i$ ,  $\text{tourn}[i]$  is greater than  $\text{tourn}[i+1]$  that means  $\text{tourn}[i]$  is the current largest. So we assign it  $\text{min}$  which is that low value and we doubled the index otherwise, otherwise means else means this one was the largest element. So we assign this tournament and the double the index. So this was our simple downward traversal and the upward computation was like this. We know where we are at  $i$ , so when did we come out of the loop when this  $i$  exceeded  $2n$  minus 1. So, if we half that we will get back to the place maybe wanted to start it. So we start with  $i$  by 2, we continue till  $i$  is greater than 1 the only difference with the tournament computation is that we just divide  $i$  by 2 instead of minus  $n$  by 2. And we have to take two cases if it is odd or even, if it is odd we compare  $\text{tourn}[i]$  and  $\text{tourn}[i+1]$  and put it in  $\text{tourn}[i]$  by 2.

(Refer Slide Time 37:46 min)

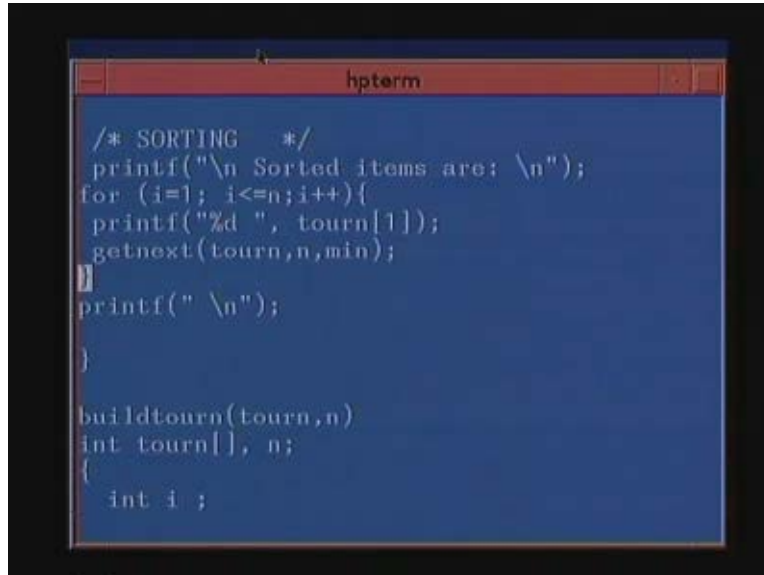


```
hpterm
while(i <= 2*n -1){
  if (tourn[i] > tourn[i+1]){
    tourn[i] = min;
    i = 2*i;}
  else{
    tourn[i+1] = min;
    i = 2*(i+1);}
}

/* UPWARDS RECOMPUTATION */
for(i=i/2;i>1;i=i/2){
  if (i%2==0)
    tourn[i/2]=maxi(tourn[i], tourn[i+1]);
  else
    tourn[i/2]=maxi(tourn[i], tourn[i-1]);
}
```

And if it is even we compare  $\text{tourn}[i]$  and  $\text{tourn}[i-1]$  and put it in  $i$  by 2. So this was the upward computation and once we finish the downward and the upward computation, we have got the next element in  $\text{tourn}[1]$ .

(Refer Slide Time 38:00 min)

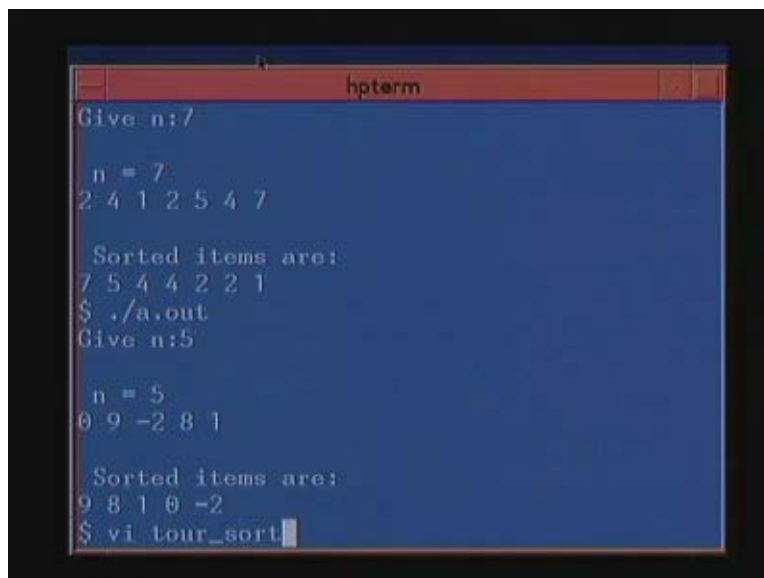


```
hpterm
/* SORTING */
printf("\n Sorted items are: \n");
for (i=1; i<=n;i++){
printf("%d ", tourn[i]);
getnext(tourn,n,min);
}
printf(" \n");
}

buildtourn(tourn,n)
int tourn[], n;
{
int i ;
```

So, this is the routine, the next element in this loop here starting here, here (Refer Slide Time: 38:02), this loop will print tourn 1 get next again print get\_next, print get\_next and print it n minus 1 times, n times to find out the sorting.

(Refer Slide Time 39:00 min)



```
hpterm
Give n:7
n = 7
2 4 1 2 5 4 7

Sorted items are:
7 5 4 4 2 2 1
$ ./a.out
Give n:5
n = 5
0 9 -2 8 1

Sorted items are:
9 8 1 0 -2
$ vi tour_sort
```

So this way we have now got an algorithm which is the much more efficient sorting algorithm than our traditional exchange sort or bubble sort. And this sorting algorithm takes  $n \log n$  to the base 2, so which is much better than  $n^2$ . We will address the issue of complexity of sorting and study others sorting routines as well but this whole series of 3 lectures was devoted to tournaments which is the very very simplified structure

just to emphasize how data structuring can help problem solving and how data structuring is absolutely essential to make problem solving much more efficient. Thank you.