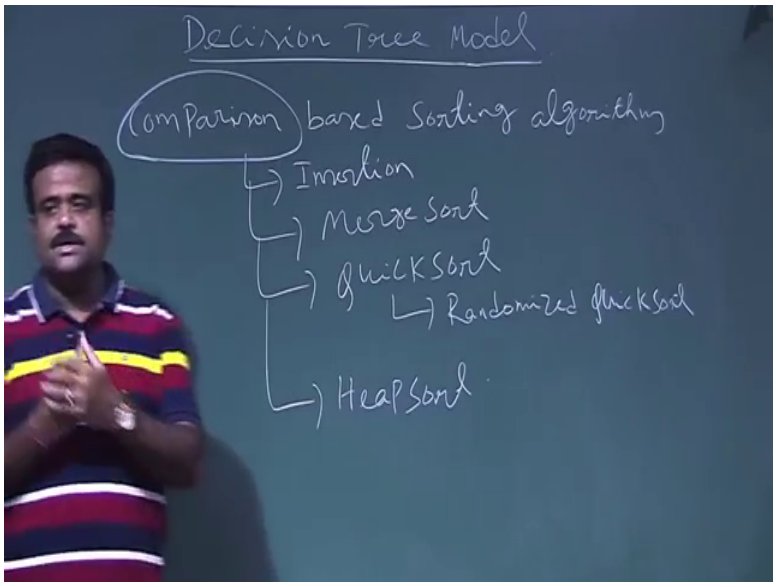
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 15
Decision Tree

So, far we have seen the sorting algorithm which are basically comparison based sort so; that means, we have we compare 2 element to know their ordering.

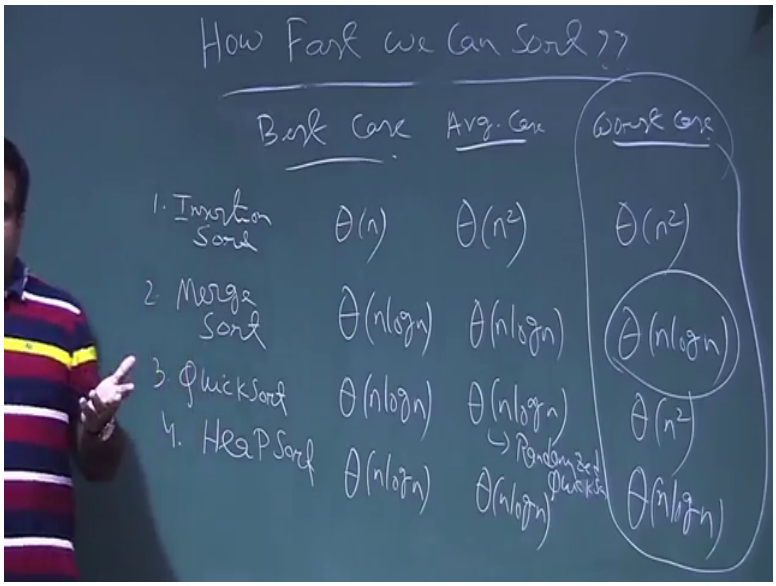
(Refer Slide Time: 00:38)



So, basically we compare the elements to get the ordering. So, those are basically comparison based sort. So, comparison based sort, sorting algorithm. So, like we have seen insertion sort, then we have seen the merge sort, then we have seen the quick sort.

And a version of randomized quick sort, and also we have seen heap sort. So, all these sorting algorithm we have seen are basically comparison based sort so; that means, we compare 2 elements to get their relative ordering. So, these are comparing the elements comparing the input elements. So, these are comparison based sort. So, what are the time complexity for this sorting algorithm.

(Refer Slide Time: 01:57)



So, the question is how fast we can sort? So, that that is our topic today I mean. So, how fast we can sort? Ok.

So far these are the sorting algorithm we have discussed. So, let us have a time comparison trivial for this sorting algorithm. So, for, so let us have best case, average case, and the worst case time complexity of this. So, we talk about we have seen insertion sort. Insertion sort best case we know it is a order of n; that means, when it is when the array is already sorted. And that will give us the best case, and average case we have seen is also order of n square, and the worst case is also order of n square.

And then we talk about merge sort, merge sort we know all the cases it is basically order of n log n. Because it does not matter whether a input is sorted, reverse sorted we go to the middle we just partition, it we just go to the middle we divide into 2 sub array we sort this sub array, sort this sub array then we call a merge sub routine. Then it is a all the cases is order of n log n. And it is not a n plus sort because to call the merge sub routine we need to have a extra array.

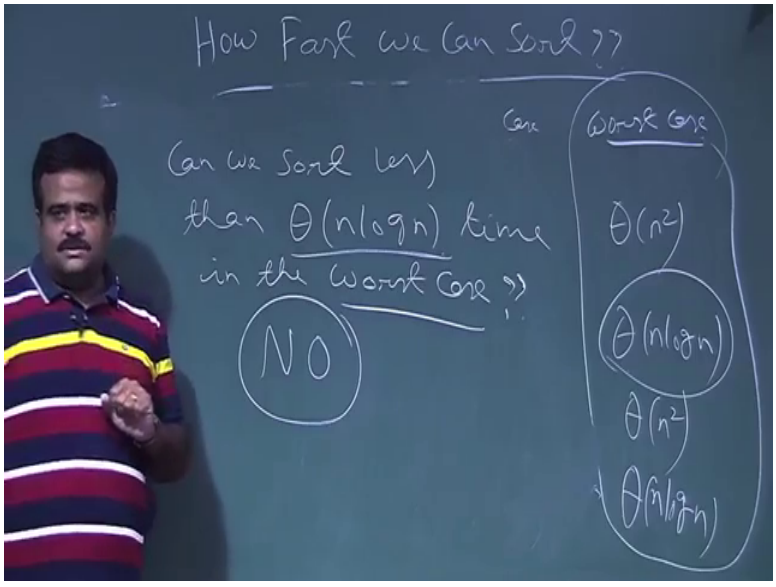
So, then we discuss the quick sort. And the best case we have seen it is n log n, when we choose the pivot is maximum or minimum always. And the average case this average case analysis we did for randomized version of the quick sort which is the expected run time, this is the randomized version of the quick sort, randomized quick sort. And then the worst case we know order of n square for quick sort because, worst case it is always,

always we choose the pivot is, best case we do not choose the pivot is minimum or maximum, but for the worst case we always choose the pivot is to be maximum or minimum. Then it will be partition is 0 is to n minus 1 always.

And then we talked about the heap sort in the last class, which is also all the cases it is order of n log n. So, basically we make use of the data structure heap. So now, the question is, in the, we are basically concerned about worst case. So, our question is, how fast we can sort in the worst case? Because worst case is the most usual case one should go for, because it is give us a guarantee. So, the question is how fast we can sort? So, so far we have seen this is the best way, the order of n log n which we are achieving by merge sort or in the heap sort in the worst case. And merge sort is not a in place sort, but heap sort is a in place sort everything, we are doing in that array we do not need any extra storage for heap sort.

So, now the question is this the faster way or we can reduce the worst case time complexity further? So, the question is, can you sort, can we sort, sort less than order of n log n time In the worst case worst case?

(Refer Slide Time: 06:03)



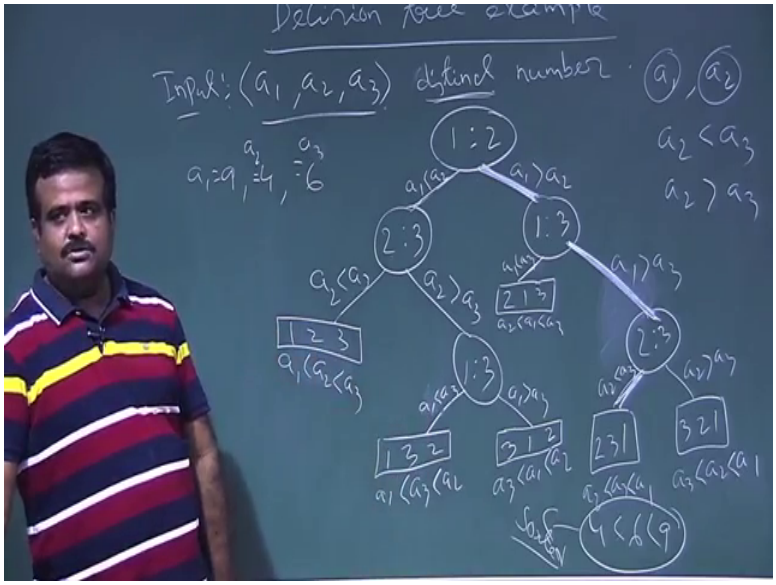
So, that is the question in the worst case. So, so the question is this the best one in the worst case scenario or we can have some better algorithm which will give us the worst case time complexity better than, order of n log n or better than order of n log n means it

could be linear it could be log n, it could be n log, log n something like that. So, it is should be time complexity should be lesser than less than this. So, that is the question.

So, the answer is if we are using the comparison based sort so; that means, if we are comparing 2 element to get their relative ordering then the answer is no. So, there is no comparison based sort which can give us the worst case time complexity lesser than this. So, for any comparison based sort if we consider that will give us the worst case I mean this is the lowest one. We cannot reduce this time further. So, that we have to convince by the help of what is called decision tree. So, will take the help of decision tree model to convince this answer that if we use some comparison based sort, any comparison based sorting algorithm we cannot do better than n log n in the worst case.

So, let us convince this with the help of a decision tree. So, what is the decision tree? So, basically, decision tree example.

(Refer Slide Time: 08:03)



Suppose we have given 3 number to sort. a 1, a 2, a 3, 3 numbers are given a 1, a 2, a 3 and we need to sort. So, what we do we will form a tree decision tree. So, for that for making this tree we assume this number at distinct all are distinct. A 1 they are distinct number these are input and we need to sort. So, we have 3 distinct numbers ok now how to form a decision tree. So, first we compare a 1 with a 2. So, 1 is to 2, this we write in this way. So, we are comparing a 1 and a 2. So, a 1 is a number a 2 is a number and this 2 are distinct. So, if we compare 2 number which are distinct. Then what are the

possibilities? We have only 2 possibilities, because we are not taking their they may be equal. So, equality possibilities is gone.

So, we have 2 possibilities, either a 1 is less than a 2 or a 1 is greater than a 2. So, if a 1 is less than a 2 will go to some branch left branch. And if a 1 is greater than a 2 we will go to right branch and we will do the subsequent comparison again. So, this branch will go if a 1 is less than a 2. And this branch will go if a 1 is greater than a 2. There is only 2 option because, these are distinct. Now if you follow this branch again we compare a 2 and a 3. So, again if we compare a 2 a 3, so there are 2 possibilities either a 1 a 2 is less than a 3 or it will greater than a 3.

So, again we have 2 possibilities. So, this way a 2 is, so we will go for further comparison if a 2 is this is the branch for to follow if a 2 is less than a 3, and this is the branch if a 2 is greater than a 3. Now if we reach to this branch a 1 is less than a 2 a 2 is less than a 3. So, this will reach to a decision. What is the decision? This is telling us a 1 is less than a 2 less than a 3, this is the decision.

So, this we denote by 1, 2, 3. Suppose we have a example such that a 1 is less than a 2 then we will follow this branch again we compare 2 3 a 2 is less than a 3 then this is the branch we follow and this will reach a decision. So, these are all this is the leaf node. Now here if we go for this branch, we have to compare again a 3 with a 1 with a 3 to reach to a decision. So, if you compare 2 number again. So, we have 2 possibilities a 1 is less than a 3 or a 1 is greater than a 3. So, if a 1 is less than a 3, So we know the a 1 is less than a 2 and a 2 is greater than a 3 and here, a 3 is greater a 3 is greater than a 1.

So, this is giving us the decision like 1 3 two; that means, a 1 is less than a 3 less a 2. Because a 1 is less than a 3 coming from here and then a 2 is greater than a 3 and then again a 1 is greater than a 3. So, if we follow this branch it is reaching us to a decision that, a 1 is less than a 3 less than a 2. And similarly if we follow up this branch then it will reach to a decision 3 1 2; that means, a 3 is less than a 1 less than a 2, ok.

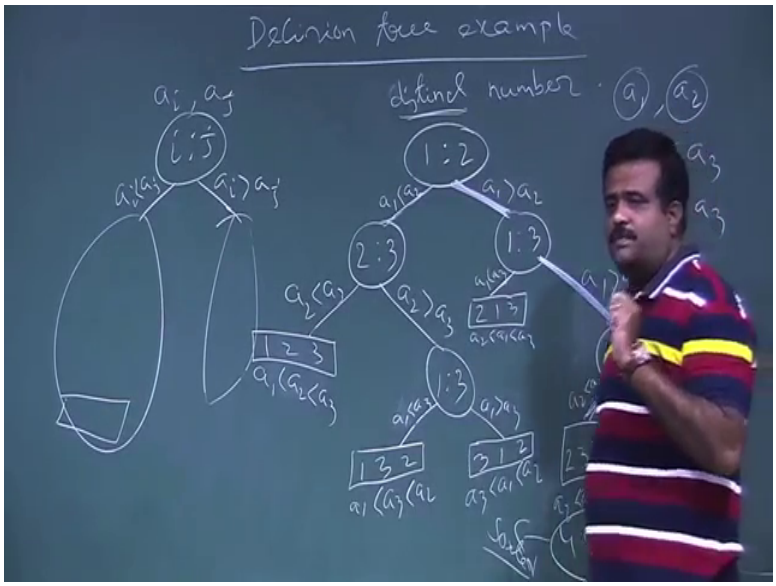
Now, come here. So, if a 1 is greater than a 2 then we have to do further comparison. So, we compare a 1 in the a 3. So, if a 1 is less than a 3 and we know the a 1 is greater than a 2, So this will reach to a decision 2 1 3 so; that means, a 2 is less than a 1 less than a 3. Now if we have to follow this branch, so then we have to compare again a 2 and a 3. So, these will reach us to a, so this is basically, this should come down here.

So, this is basically 2 a 2 a 3. So, this is giving us 2 decision. So, if a 2 is less than a 3. So, this is basically a 1 if greater than a 3, and this is a 1 is less than a 3. So, this is reaching to a decision 2 3 1 and this is reaching. So, this is a 2 is greater than a 3. This is reaching to a decision 3 2 1. So, this is basically a 3 is less than a 2 less than a 1 and this is a 2 is less than a 3 less than a 1. So, this is this is called decision tree, decision tree.

Now, we can take an example. Suppose we take say 9, 4, 6 suppose this is our a 1, this is a 2, this is a 3. A 1 is so, a 2 is 4 a 3 is 6 ok. So, we want to follow this model. So, now, we start with here we compare. So, this is the model we have this tree we have. This is the decision tree. Now first we compare a 1 with a 2. So, a 1 is 9 a 2 is 4. Now 9 is greater than 4, we must follow this part, and then because we have to follow this part because 9 is greater than 4. Then we have to again do the subsequent comparison to reach a decision. These all leaves notes are basically giving us the decision. So, they are basically permutation. So, leafs are basically all possible permutation. So, there are 3 notes. So, there are factorial 3 permutation, so factorial 3 6, 6 permutation ok.

So, here 9 is greater than 6. So, we follow this path, then again we after reaching here we have to compare a 1 a 3. So, a 1 is 9 a 3 is 6 we compare 9 and 6. So if so, 9 is greater than six so again we follow this path. Now again we have to compare a to a 3 a 2 is 4 a 3 is 6 now 4 is less than 6. So, we follow this path and we reach to a decision, what is the decision? That a 2 is less than a 3 is less than a 1. So, what is a 2? A 2 is basically 4, 4 is less than a 3 is 6 and less than 9 which is correct basically. So, this is sorted. So, every path is if we so for a given input basically we are following a path to reach to a decision. So, to reach to a to get a sorted array. So, this is our input we need to sort it. So, we start with the root and we compare 2 element and then based on the comparison either if these are distinct element.

(Refer Slide Time: 16:35)



So, basically what we have we have say nodes are basically i is to j. So, basically here we are comparing a i and a j. And these are distinct nodes distinct element. So, if we compare 2 element then a i is either less than a j or a i is greater than a j.

So, if a i is less than a j, then we will go to the this sub tree. And we will do the subsequent comparison until we reach to a decision. And then if a i is greater than a j we will go to this sub tree right sub tree until if and we keep on comparing again until it reach to a decision, until it reach to a leaf note and every leaf contain a permutation. And that is the ordering between the elements. So, any comparison based sorting algorithm we have seen is basically execution of any comparison based on sorting algorithm we have seen basically, execution of this decision tree.

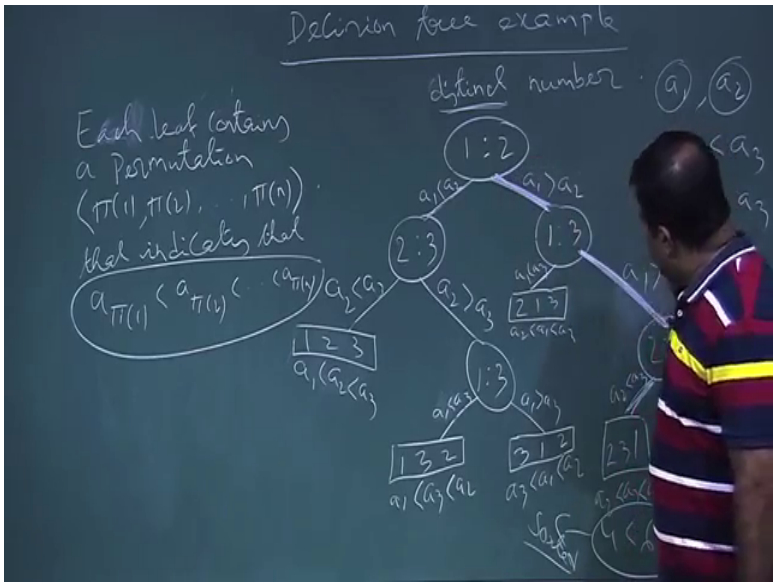
So, we basically every comparison based sort if you compare, if you think like insertion sort. Insertion sort what we are doing? We are comparing 2 elements and then we have taking a decision. So, basically same thing, so any comparison based sort basically we are comparing between the elements and we are going and further comparing, again we are further comparing and finally, we are reaching to a decision finally, we are reaching to a permutation which is giving us the ordering of that element. So, this is basically the decision tree model.

And which is basically the any comparison based sort execution of any comparison based sort is basically the path execution of the path of the decision tree from root to the

leaves. So, here any note is basically of this form i is to j. So, this is basically comparing the a i with a j now if a i is less, than a j we will go for this path and we do subsequent comparison and if a i is greater than a j we go for this path, we do the subsequent comparison. So, so this is the model of decision tree and any comparison based sort is basically fit under this model, ok.

So, and what are the leaves? So, leaves are basically, each leaf yeah.

(Refer Slide Time: 19:05)

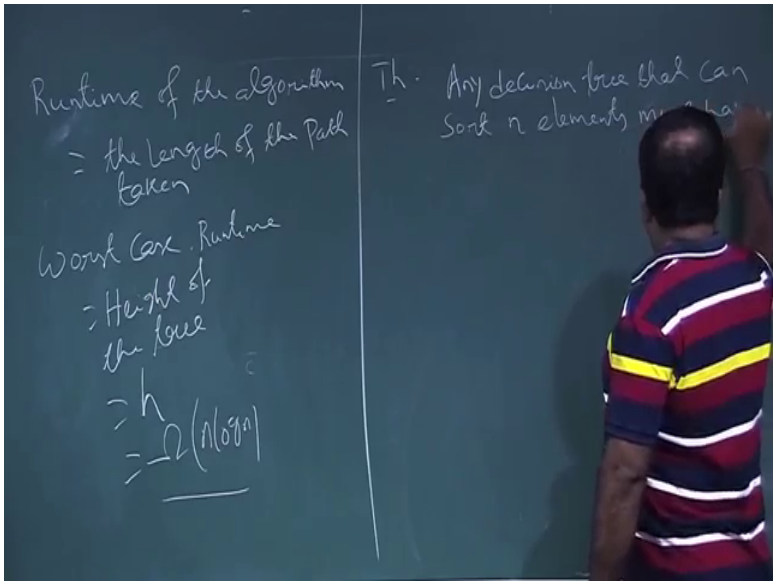


So, each leaf contain a permutation say, pi 1, pi 2, pi n. That indicates that, they are they are in n that order that, indicates that a pi 1 is less than a pi 2 less than dot, dot, dot a pi n. So that means, they are in the order. So, we reach to a decision. So, this is a decision, this is a decision depending on the input we reach to that particular I mean, one of this part, one of this leaf and that is a decision.

So, if each leaf contain a decision. So, basically any execution of any comparison sort is basically we just execute the root to the leaf, so basically the run time. So run time depend on the length of the path. Now here if the input is say 9, 4, 6 then we follow this path. Now if the input is a instead of 9, 4, 6 if the input is say 9, 6 and then say 10, then what we do? So, we just this is our a 1, a 2, a 3. If the input is this then what is the path? The path is basically; let us erase this earlier one.

So, path is basically we compare a 1 with a 2. So, we follow this path then we again compare a 1 and a 3, a 1 is 9 a 3 is 6 so we follow this path. So, we reach to a decision, what is the decision? A 2 is less than a 2 is 6, which is less than a 1 and which is less than 10 which is correct. And length of this path is small. So, depending on the input I mean, the time complexity will be the length of the path decision tree. So, let us write this, ok.

(Refer Slide Time: 21:58)

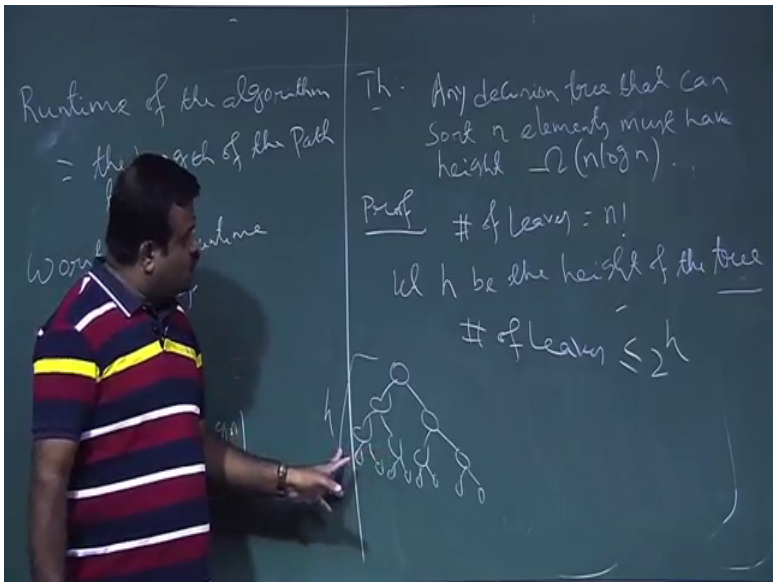


So, the run time, run time of this decision tree model of the algorithm is equal to the length of the path taking.

For this input we go to the less length, but then what is the worst case run time? Worst case run time will be the height of this tree, because that is the worst case, worst case means maximum path we follow. And that is the height of the tree. In the worst case, worst case means we have to follow the root to the maximum length part. And that is the height of a tree. So, the worst case run time is basically, is basically height of the tree. And this is denoted by h and we prove that height of this tree is n log n.

And that will give us the answer that in the worst case any comparison based sort we cannot have a comparison based sort which can be better than n log n. So, let us talk about how the worst case, how the height of this decision tree is n log n ok.

(Refer Slide Time: 23:52)



So, this we prove by a theorem. So, this theorem is telling any decision tree that can sort n element must have height. So, height must be bounded by n log n lower bound. So, height must be greater than equal to n log n.

So, these we have to prove. So to prove that, suppose we have n elements. So, in the example we are having 3 elements and we form the tree. Now suppose we have n element and we form the tree. So, so what is the number of leaves? So, number of leaves is basically all possible permutation, basically they have the decision we do not know which input will come. So, the number of decision will be, so we should have a decision for all type of inputs so; that means, and each leaf is basically the decision. So, for a given input we execute the path and we reach to a decision.

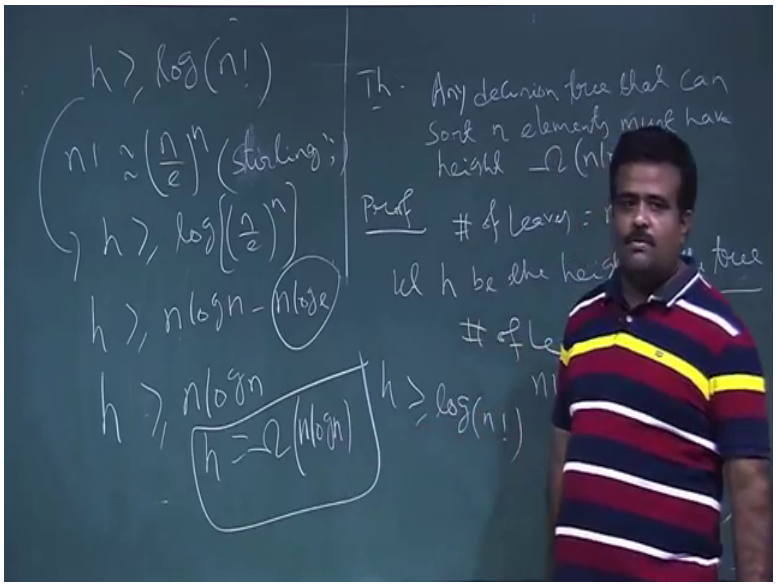
So, number of leaves must be factorial n, and this is the all possible permutation I mean permutation of n inputs. So, this is the number of leafs. Now this is the binary tree so; that means, we have each mode have 2 notes. So, if the height is h, let h be the height of the tree. Then 2 to the power h must be greater than equal to factorial n because, then the number of leaves must be, number of leaves must be 2 to the power h. Why? Because, so how it is a h?

So, if we have a complete binary tree like this. Then this is the height means maximum depth if the height is h. So, at this level what is the number of notes? 2, this level 2 square. So, if the height is h. So, if is a complete binary tree then the number of leaves

will be 2 to the power h, but it is not a complete binary tree. There are some leaves some branches are ending fast so; that means, if it could complete then it is 2 to the h so; that means, the 2 to the number of leaves must be less than 2 to the power h

So, from here we know the number of leaves is factorial n it is basically 2 to the power h.

(Refer Slide Time: 27:05)



So, now from here we can just write h is basically greater than equal to log of factorial n. Now, we will use a formula which is stirlings formula. So, let us. So, what we have? We have h is greater than equal to log of factorial n. Now factorial n can be approximated by, n by e to the power n. And this is called stirlings formula, this formula is stirlings formula.

So, if we use here this, so h is greater than equal to log of n by a to the power n. So, this is basically h is greater than equal to n log n minus n log e. Now this is a low ordered term we can ignore this. So, this is basically height is basically bounded by n log n. So, this is basically height is omega n log n. So, the height of a decision tree cannot be less than n log n it is a lower bound by n log n. So, omega big omega of n log n so; that means, worst case run time of any comparison based sort is bounded by lower bounded by n log n.

So, we cannot, we cannot sort n element faster than n log n if we are using comparison based sorting algorithm. And this we have seen by the help of decision tree. So, next

class we will talk about linear time sorting algorithm, but those are not comparison based sort, we are not comparing the between the elements we are just seeing value of the element and we are putting into the bucket. So, those are basically buckets are we will discuss in the next class.

Thank you.