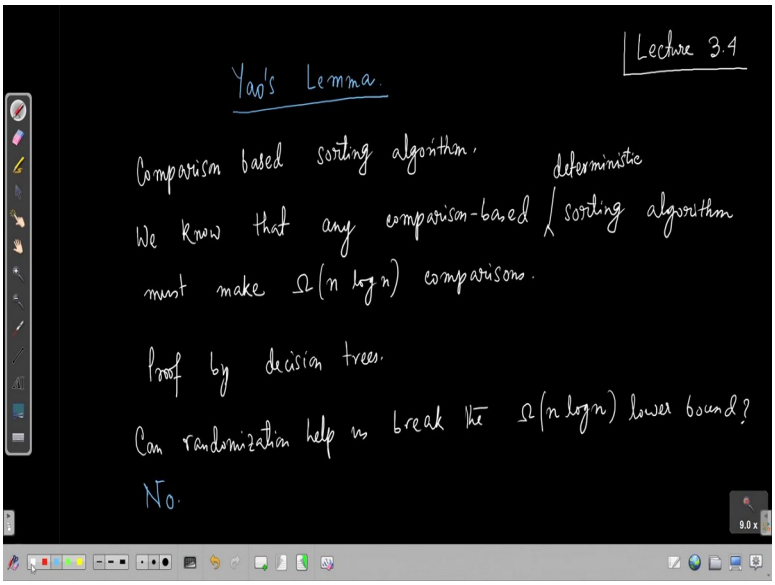
Algorithmic Game Theory
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 14
Yao's Lemma and its applications

(Refer Slide Time: 00:38)



So welcome, so today we will do an very important application of metric schemes which is famously called Yao's Yao's Lemma. So, let me explain the background this Yao's lemma is very useful for proving lower bounds for algorithms. So, let us take a concrete example. Let us take comparison based sorting algorithms.

So, any sorting algorithm is called a comparison based if the only way to infer to know whether to among two elements which one comes before and which one comes after is to compare them, that is it there is no other ways. Examples of comparison based sorting algorithms are say quick sort, march sort, insertion sort, bubble sort, selection sort and everything, hip sort.

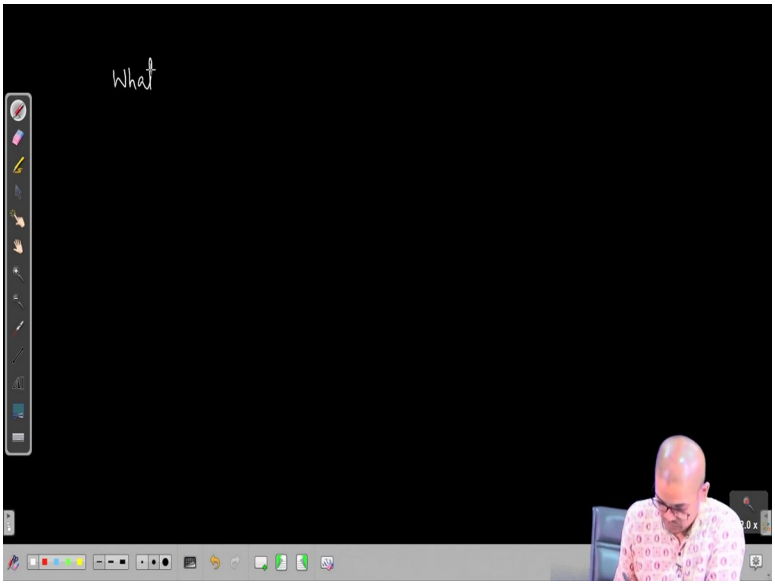
There are sorting algorithms which can sort integers which are not comparison based like counting sort, radix sort. So, we are not talking about those sorting those sort non-comparison based sorting algorithms we are only focusing on comparison based sorting algorithms. And the base running time that we know is $O(n \log n)$ to sort n items.

And we also know so we know that any comparison based sorting algorithm must make omega n log n comparisons. Any comparison based deterministic sorting algorithm not probabilistic and the proof typically goes by via decision tree method. So, proof by decision trees I am not going into the proof if you have not seen the proof I would strongly recommend you to see the proof it is in any standard textbook of algorithms you will be you will find plenty of YouTube videos and online materials. So, this elegant and a nice proof.

Now, we can ask ok so maybe any deterministic algorithm may not breach the $\Omega(n \log n)$ barrier how about a randomized algorithm. So, what is the main question. So, can randomization help us break the omega n log n lower bound. So, we have seen that the randomized quick sort its worst case running time is $O(n^2)$, but its expected running time or average case running time is much faster its big o of n log n which is significantly faster than $O(n^2)$.

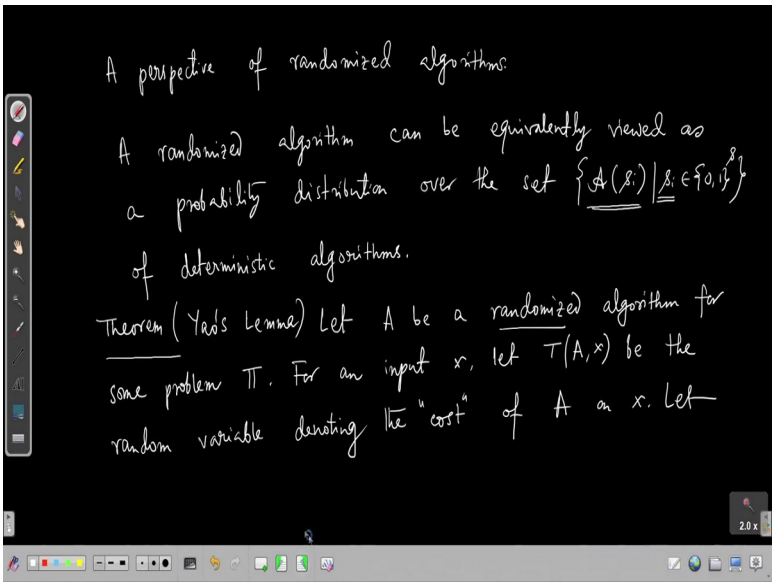
So, that way can there exist a randomized algorithm whose expected number of comparisons is less than $\Omega(n \log n)$. And the answer is no and this is the main topic of our discussion we will prove using Yao's lemma. But before that let us see let us dig into what do we mean by randomized algorithm.

(Refer Slide Time: 05:52)



So let me write this way.

(Refer Slide Time: 05:59)



A perspective of randomized algorithms you know randomized algorithms are like deterministic algorithm except every now and then it has an access to some random bits. So, it has an access to a random fare or fare coin which it can toss and depending on the outcome of the random event it can decide its course of action.

So, by some magic if we can fix the outcome of those random events beforehand then its a deterministic algorithm. So, here is the perspective. So, a randomized algorithm can be viewed can be equivalently viewed, randomized algorithm can be equivalently viewed as probability distribution over the set $A(s_i)$ such that $s_i \in \{0,1\}^n$ set of deterministic algorithms what do you mean by that.

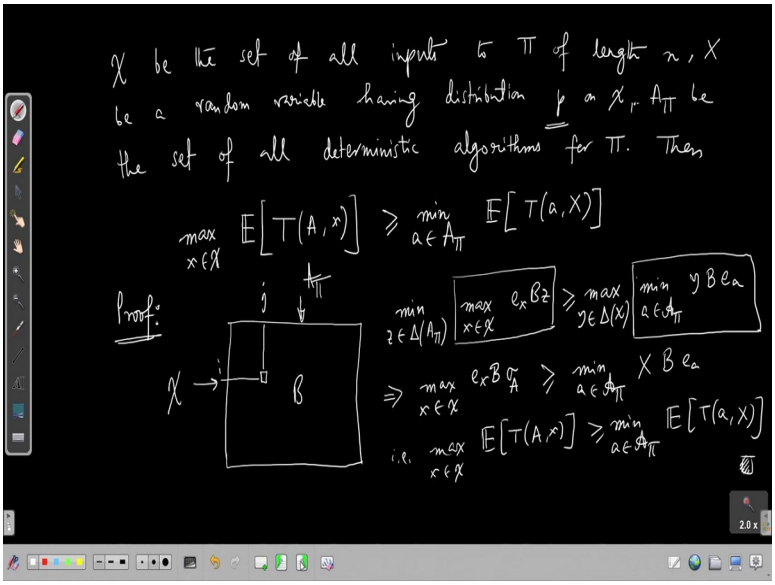
So, suppose the algorithm makes small s number of random coin tosses the outcomes are 0 1. So, s_i is A string of length s a binary string of length s and if you fix the outcome of those random coin tosses to be s_i, then let calligraphic $A(s_i)$ is the deterministic algorithm which is which you obtain by fixing the outcome of the coin tosses of the randomized algorithm to be s_i.

So, there is certain probability of happening the outcome of that those s spin random coin tosses to be s_i. So, randomized algorithm a is this deterministic algorithm $A(s_i)$ with that probability. So, this is what we mean by it is a probability distribution over a set of deterministic algorithms and this perspective will be extremely useful.

So now, let us state Yao's lemma. So, its a mouthful of statement, but let me write theorem popularly noses known as Yao's lemma. So, let A be a randomized algorithm; A be a randomized algorithm for some problem Π. For example, Π could be the sorting problem and A could be a quick sort ok. So, its a randomized algorithm.

So, randomized pick it picks the pivot uniformly at random state. For an input x, let T of A comma x be the random variable denoting the cost. Cost is a general phrase to denote anything like time or space or say number of comparisons for sorting what we will use. Let $T(A,x)$ be the random variable denoting the cost of A on x. Why random variable because A is not a deterministic algorithm, so it is not a fixed one number its a random variable its it takes certain values with certain probabilities ok.

(Refer Slide Time: 12:32)



Let cal X be the set of all inputs for that problem be the set of all inputs to Π. For example, for sorting for the sorting problem it is the set of all subset of n integers on of length n, X be a random variable with having distribution say p on x. So, I am also defining another distribution probability distribution p on the set of all inputs is, this is independent of the given algorithm.

The given algorithm does not have any distribution on the input I am defining a distribution I am not defining it let X be a random variable and let p be a distribution ok. And some more terminology let A Π be the set of all deterministic algorithms for Π, then we have the following.

What is the expected running expected cost of the algorithm A. So, the random variable was T of A comma x this is the random variable denoting the cost of algorithm A on input s on input x. I want to find the expected cost because its cost is not fixed it is a random variable it takes certain values with respect to certain probabilities. And I want to take max of this max over x over all possible inputs I take that input which maximizes this expected utility expected cost. So, this is the worst case expected cost.

The Yao's lemma says that this is greater than equal to min a in A Π . So, you consider an algorithm a small a for the problem deterministic algorithm small a for the problem pi. And you look at its average cost when you vary the input according to the probability distribution p. So, T a comma capital X is the is the random variable denoting the cost of a on X.

Now here a is a deterministic algorithm, but here input is random and. So, as I vary input the cost also varies it takes certain values with certain probabilities and says that this is minimum greater than equal to minimum of a in a Π expectation of $T(a, X)$. And this looks very abstract and not very useful we will see its use soon by when we prove that there does not exist even a randomized algorithm whose expected number of comparisons is is better than $\Omega(n \log n \log n)$. So proof, but let us prove it this is very easy.

It is the easier direction of that max min theorem. So, consider a matrix scheme consider a matrix B where this rows are indexed by this set x the set of all inputs and columns are indexed by the set of all algorithms ok. The columns are indexed by set of all algorithms for the problem. And of course, the entries here this say i and this is j i comma jth entry is the performance of the jth algorithm on the ith input the cost ok. Now, we know that min max is greater than equal to max min, min $z \in \Delta$ $A(\Pi)$ minimum over columns max over rows x in.

The inner one could be written as written with respect to pure strategies if you recall because of the averaging principle. $e_x B_z$ recall ex is the vector indicator vector where the x-th entry is one all are 0 and we will assume that e x is a row vector and z is a column vector instead of writing transpose and those sort of things we will make assumptions. So, that this matrix multiplication makes sense. This is greater than equal to

max y in delta x max over rows and you should allow randomizations min over a in A_{Π} is cost.

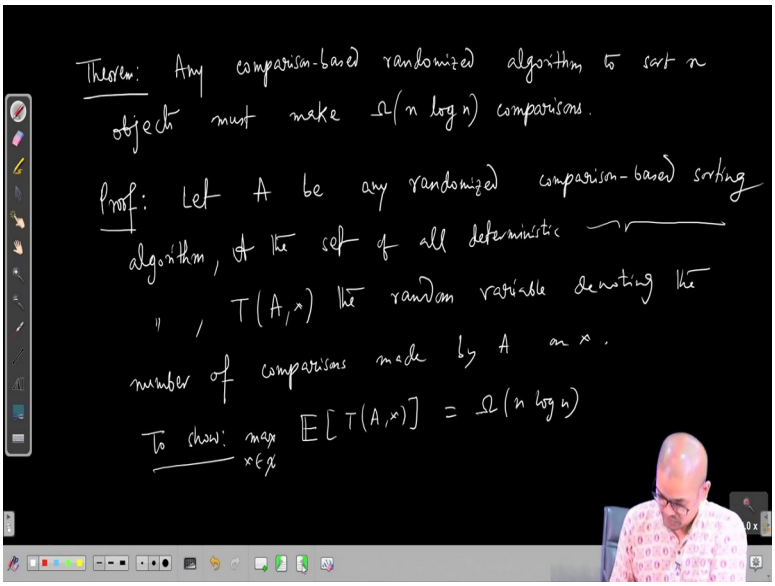
Sorry, this is e_y is a random; y is a random y is a distribution $yB_e a$. Now instead of taking min if the minimum of this quantities is greater than something then for any if I replace min with any particular any particular. So, what is z? z is a probability distribution over algorithms for this A pi probability distribution over the deterministic algorithms. So, z is a randomized algorithm.

So, minimum over all randomized algorithms. So, if I pick a particular randomized algorithm namely the randomized algorithm given to us then still this inequality holds. Because if minimum is greater than something then any if you replace instead of taking minimum, if you take something then that is also the greater. $e_x B_z$ is like σ_A , σ_A is the corresponding probability distribution for our randomized algorithm A. Same goes here if maximum is less than equal to something then if I pick some particular value then it is also remains less.

$a \in A(\Pi)$ and. So, what y I should take? I should take this y is a probability distribution over the input I will take x the given probability distribution $xB_e a$. Now what is this? This is nothing but this costs. So, that is max x in this expected cost of T A x this is greater than equal to minimum $a \in A(\Pi)$ expected cost of T expected cost of a on X this concludes the proof.

Nothing great, but what is hidden is its power its usefulness. So, let us see its usefulness while proving this theorem by proving this theorem.

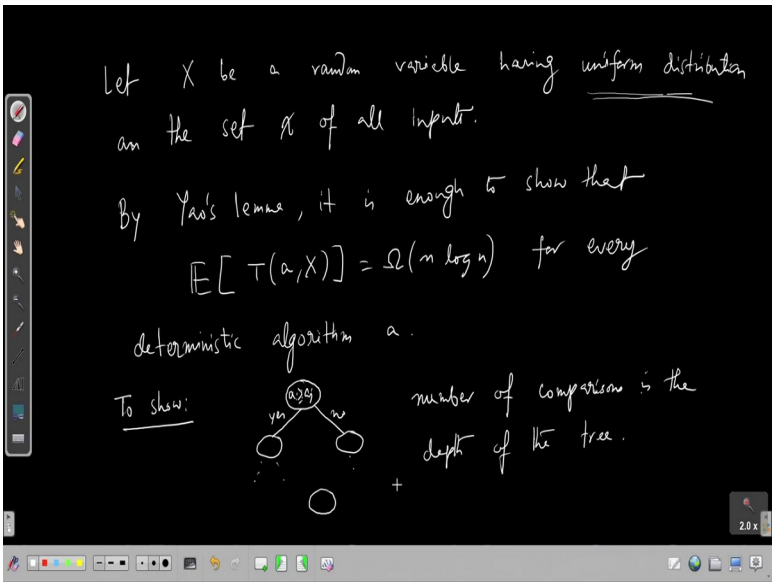
(Refer Slide Time: 22:51)



So, what is the res theorem is any comparison based, any comparison based randomized algorithm to sort in objects makes must make omega n log n comparisons proof. So, let A be a randomized algorithm any randomized algorithm randomized comparison based sorting algorithm ok. And as usual borrowing notations this calligraphic A the set of all deterministic algorithms the set of all deterministic comparison based sorting algorithms ok.

And as usual just same notation a $T(A,x)$ with a random variable denoting the number of comparisons made by A on x ok. So, to show what we need to show expectation of $T(A,x)$ and I should take max x in this is $\Omega(n \log n)$. Now we need to show a lower bound.

(Refer Slide Time: 26:38)



So, now using Yao's lemma let capital X be a random variable with having uniform distribution on the set X of all inputs ok. So, by Yao's lemma; by Yao's lemma it is enough to show it is enough to show that show that expectation of $T(a,X)$ is $\Omega(n\log n)$ for every deterministic algorithm a deterministic comparison based sorting algorithm a. Now you see that by using Yao's principle we have shifted the randomization from algorithm to input.

And the on input also the randomness is in our hand we are defining our suitable distribution which we are in this case uniform distribution. And now before at the beginning the randomization was inside the inside the algorithm and we do not have any clue about how it is sampling and so on.

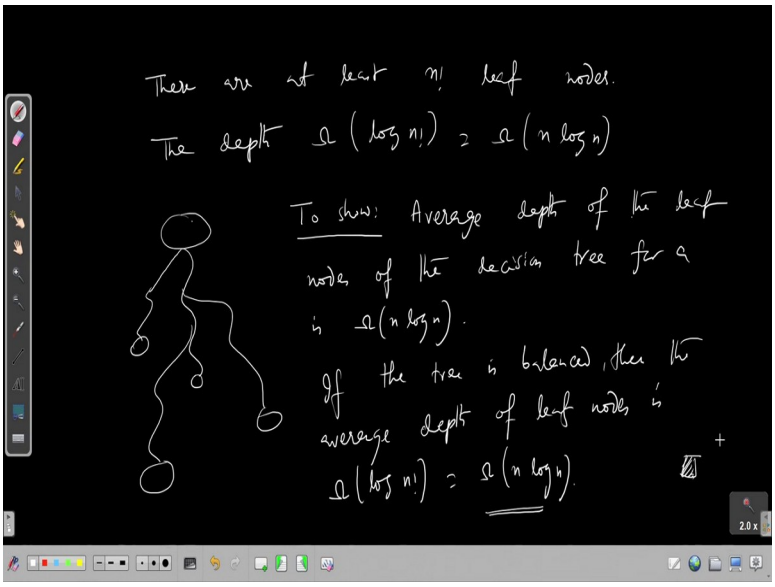
Now we have made the algorithms simpler in the sense that it is now a deterministic algorithm and randomization is also simpler because we know the distribution uniform distribution in this case, distribution of our choice and on the input. So, now what we need to show is that because it is a uniform distribution.

So to show, again from here it is easier it will be easier if you know the proof for the deterministic algorithm. That for deterministic algorithm the number of comparisons must be $\Omega(n\log n)$. Now here also you view the computation of that sorting as a decision tree and you the height of the decision tree is must be at least $\Omega(n\log n)$.

So, what is that decision tree it is like let me briefly tell. So, each node is a decision node at every node we are making a comparison we are comparing two elements say a_i and a_j and depending on we are checking if a_i is greater than equal to a_j or not. If it is yes then we go to some path and if it is no then we go to some other path and so on.

So, we make some comparisons and at the end we output. Now what is the number of comparisons? In the number of comparisons right number of comparisons is the depth of the tree.

(Refer Slide Time: 31:10)



And we know that, go there are at least n factorial leaf nodes. So, the depth is omega n $\Omega(\log n!)$ which is $\Omega(n \log n)$. Because we are interested in the worst case bound then it is fine, but now if you look at the look at the suppose look at the decision tree of for a, so what is the average depth of any node.

So, suppose our leaf node is here what is the average depth of the leaf nodes, suppose the leaf node is here not these are binary trees leaf node is here. So, it may now it is fine we have one or few nodes at long depth if the average depth is small then it is fine, but is the average depth is also $\Omega(n \log n)$.

So, to show average depth of the leaf nodes of the decision tree for a is omega n log n. Simply because that is the average number of comparisons that is it. Now, why it should be the case? Ok. So, the easy case is if the tree is balanced if the. That means, all the leaf

nodes are at the same level as much as possible; that means, if I take any two leaf nodes either they are have the same depth or their depth can differ by at most one.

That is what you mean by say balanced tree. If the tree is balanced then the average depth average depth of leaf nodes is omega log of n factorial which is omega n log n. Now, it takes a little bit some argument to sort of convince that you know the balance tree sort of best to minimize the average depth. If you have a un unbalanced tree; that means, there are two leaf nodes whose depth differ by more than 1 then we can sort of change it change the tree and it will only increase the balance of the tree and also it will reduce the average depth of the; depth of the leaf nodes.

So, this way if we make this operations we come to a come to the conclusion that among all trees the tree which has among all trees, having the same number of leaf nodes the balanced trees have the least average depth of leaf nodes and that is the depth is also omega n log n. So, if the tree is unbalanced then also the average depth can only be more than the balanced one and it is also $\Omega(n \log n)$. So, that concludes the proof ok.

Thank you.