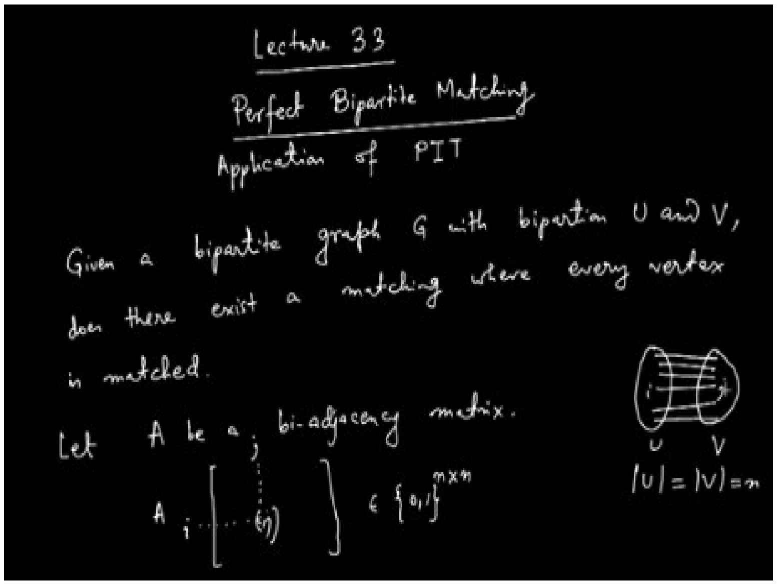
# Selected Topics in Algorithm

**Prof. Palash Dey**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Kharagpur**

**Module No # 03**
**Lecture No # 13**
**Perfect Bipartite Matching, Randomized Quicksort**

In the last class we have seen the Schwartz Zipple Lemma and using this Schwartz Zipple Lemma we have seen how we can design a simple yet efficient algorithm for polynomial identity testing problem. So in the next class in this class we will see another very simple applications and elegant application of polynomial identity testing problem into perfect bipartite matching.

**(Refer Slide Time: 00:55)**



Lecture 33

Perfect Bipartite Matching

Application of PIT

Given a bipartite graph G with bipartition U and V, does there exist a matching where every vertex is matched.

Let A be a bi-adjacency matrix.

$A = \begin{bmatrix} & \vdots & \\ i \cdots & (i,j) & \\ & & \end{bmatrix} \in \{0,1\}^{n\times n}$

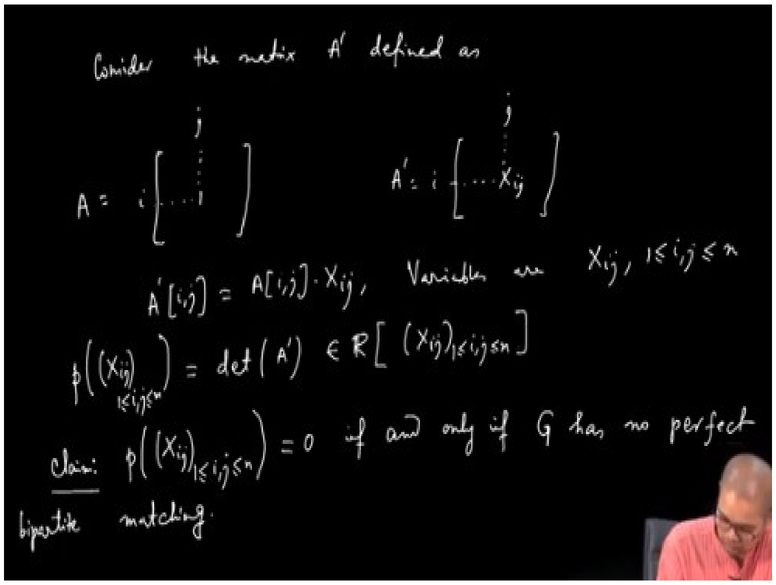
$|U| = |V| = n$

This lecture 3.3 perfect bipartite matching it turns out that the many interesting problems can be reduced to polynomial identity testing problem. And one of such example is perfect bipartite matching is as an application of P I T polynomial identity testing. So what is the perfect bipartite matching problem? Given a bipartite graph G with bipartition say U and V does there exist a matching where every vertex is matched.

So here is the bipartite graph one part is U other part is V and the question is does there exist a matching where does there exist a matching where all vertices are matched there is no exposed vertex of course this could be solved using an application of max flow. But we will see another and an elegant application of polynomial identity testing problem. So let A be a bi-adjacency matrix suppose the cardinality of U is same as cardinality of V = n if cardinality of U is not equal to cardinality of V.

Then of course we cannot have a perfect matching there would be at least one vertex remain unmatched in whichever is a larger set. What is by adjacency matrix? So here is a how does how it look like it is an n cross n matrix entries are in between either 0 or 1 it is n cross n and the i, jth entry is one if there is an edge from i to j. Now what we do is that we for each one I introduce a variable.

**(Refer Slide Time: 05:19)**



Consider the matrix $A'$ defined as

$$A = \begin{array}{c|c} & j \\ i & \begin{bmatrix} \cdots 1 \end{bmatrix} \end{array} \qquad A' = \begin{array}{c|c} & j \\ i & \begin{bmatrix} \cdots X_{ij} \end{bmatrix} \end{array}$$

$A'[i,j] = A[i,j] \cdot X_{ij}$, Variables are $X_{ij}$, $1 \le i,j \le n$

$p\left((X_{ij})_{1 \le i,j \le n}\right) = \det(A') \in \mathbb{R}[(X_{ij})_{1 \le i,j \le n}]$

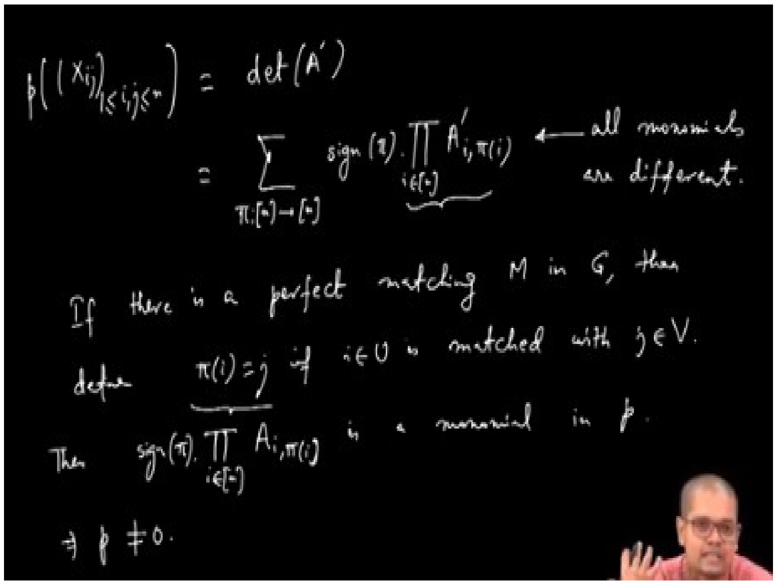
<u>Claim:</u> $p\left((X_{ij})_{1 \le i,j \le n}\right) = 0$ if and only if G has no perfect bipartite matching.

So consider the matrix A prime defined as so from e suppose is i, j-th entry this entry is 1. So let me write this way I have a 1 in i-th row and j-th column and in a prime i in i-th row and jth column I keep a variable $X_{ij}$ so I have n square variables. So basically $A'[i,j]=A[i,j]X_{ij}$. So variables are $X_{ij}$ i and j both vary from 1 to n. Now what I do is that I look at the determinant of it determinant of a prime.

This is a polynomial in these variables X so $X_{ij}$'s this is polynomial of $X_{ij}$'s and here is the claim you know this polynomial this determinant is a 0 polynomial. If and only if and only if G has no perfect matching or the other way this is a p is a non-zero polynomial if and only if P has a perfect bipartite matching. So see that once we prove this claim the problem of finding if G is a perfect bipartite matching or not reduces to checking whether a polynomial namely $p(X_{ij})$.

This polynomial has is it is it a 0 polynomial or non-zero polynomial? If it is a 0 polynomial then it does not have a perfect bipartisan matching with the non-zero polynomial then it has a perfect bipartite matching as simple as this. So and this polynomial can be thought of as a polynomial over reels.

**(Refer Slide Time: 09:39)**



$$p\left((X_{ij})_{1 \le i,j \le n}\right) = \det(A')$$

$$= \sum_{\pi:[n]\to[n]} \text{sign}(\pi) . \underbrace{\prod_{i\in[n]} A'_{i,\pi(i)}} \longleftarrow \text{all monomials are different.}$$

If there is a perfect matching M in G, then

define $\underbrace{\pi(i) = j}$ if $i \in U$ is matched with $j \in V$.

Then $\text{sign}(\pi). \prod_{i\in[n]} A_{i,\pi(i)}$ is a monomial in $p$.

$\Rightarrow p \not\equiv 0$.

So let us prove it so what is $p(X_{ij})$ in this is determinant of a prime now this is from the definition of determinant what is it? It is set of all bijective functions from you know 1 to n to 1 to n these are the sum over all permutations $sign(\pi)$ times product $A'_{i,\pi(i)}$ in. And now if there is a matching there is a perfect matching M in G then define $\pi(i)$ to be j if $i \in U$ is matched with $j \in V$.

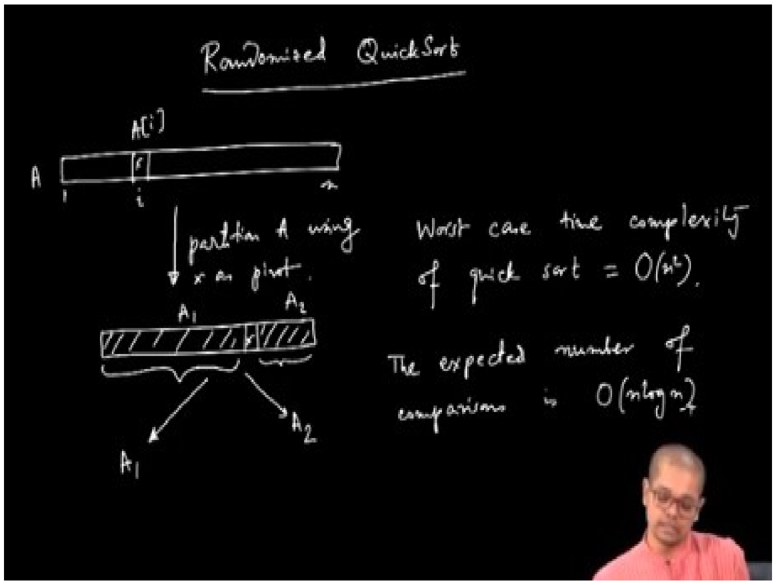
Now you see that this particular term the term corresponding to $\pi$ is not can or remains no term cancels first observation is that all monomials are different. So the only way this could be a 0 polynomial is that all in every of this product at least one $A'_{i,\pi(i)}$ is 0. So because monomials do not cancel and if there is a perfect bipartite matching we exhibit a monomial. Let us call $\pi$ this is how this monomial is defined then $sign(\pi)$ times product i in [n] $A'_{i,\pi(i)}$ is a monomial in P.

And hence this implies that P is not equal to 0. So if there exists a perfect matching in G then P is a non-zero polynomial and all statements are reversible that means if P is non-zero. It is a non-zero polynomial that means at least one monomial survives and hence the at least we have at least one perfect bipartite matching. So now you see that you know this randomized algorithm is so simple that to check whether this polynomial is a 0 polynomial or not.

All I do is to pick n square many points n square minus a real numbers uniformly at random and evaluate this polynomial evaluate this determinant. And so and that is it and that take n cube time which is an algorithm always also this much simpler than running the flow method. So these are Hallmark of randomized algorithm that often the algorithm design the algorithm coding the algorithm these are very simple.

And this because of the simplicity the hidden constants in the running time are also very small. So next we do for warm up before we move on so our next example of randomized algorithm is randomized quick sort.

**(Refer Slide Time: 14:53)**


Randomized QuickSort

Worst case time complexity of quick sort = $O(n^2)$.

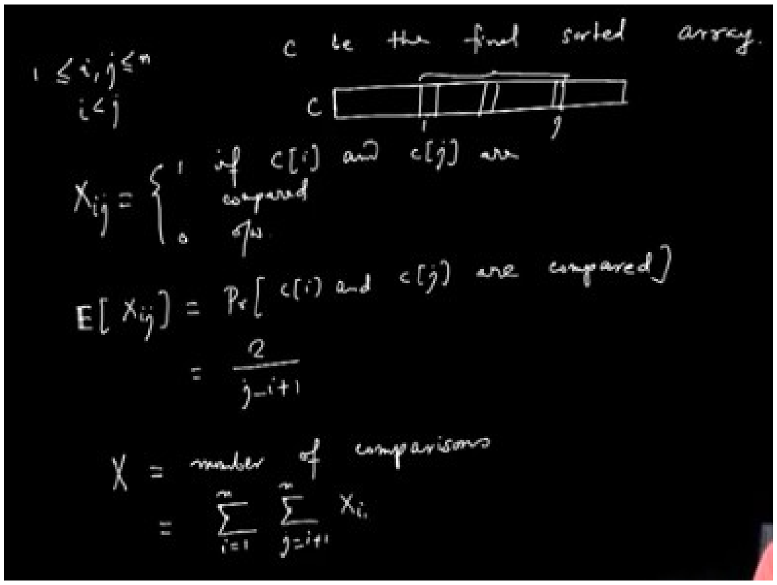
The expected number of comparisons is $O(n \log n)$

So what is a randomized quick sort? You know we all know quick sort let us again briefly go over it I have an array A of n integers $A[1,\ldots,n]$. I pick an index I uniform layer random this is $A[i]$ and suppose this element is X then I do what then I partition A using X as pivot suppose this is the partitioned ready X suppose stays here. That means the left elements the elements at the in the left of X they are smaller than or equal to X the elements which are right of X they are larger greater than or equal to X.

So this is one part let us call this $A[1]$ and the right part let us call it $A[2]$. Then the algorithm recursively source $A[1]$ and $A[2]$. Now the worst case time complexity worst case time complexity of quick sort is $O(n^2)$. And this happens if you know always this partitioned array this is unbalanced that means most of the elements are staying in one part. So for consistent consistently if we pick our pivot to be the smallest element or the largest element then we get the running time because of n square.

But the expected time complexity if we pick X uniformly random from the set of element all elements in the array then the expected number of comparisons is $O(n \log n)$. And thus the expected time complexity is also $O(n \log n)$ so how do you prove it?

**(Refer Slide Time: 18:16)**



$1 \le i,j \le n$
$i<j$

$C$ be the final sorted array.

$C$ [ ] $i$ $j$

$$X_{ij} = \begin{cases} 1 & \text{if } C[i] \text{ and } C[j] \text{ are compared} \\ 0 & \text{o/w} \end{cases}$$

$$E[X_{ij}] = \Pr[C[i] \text{ and } C[j] \text{ are compared}]$$
$$= \frac{2}{j-i+1}$$

$$X = \text{number of comparisons}$$
$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

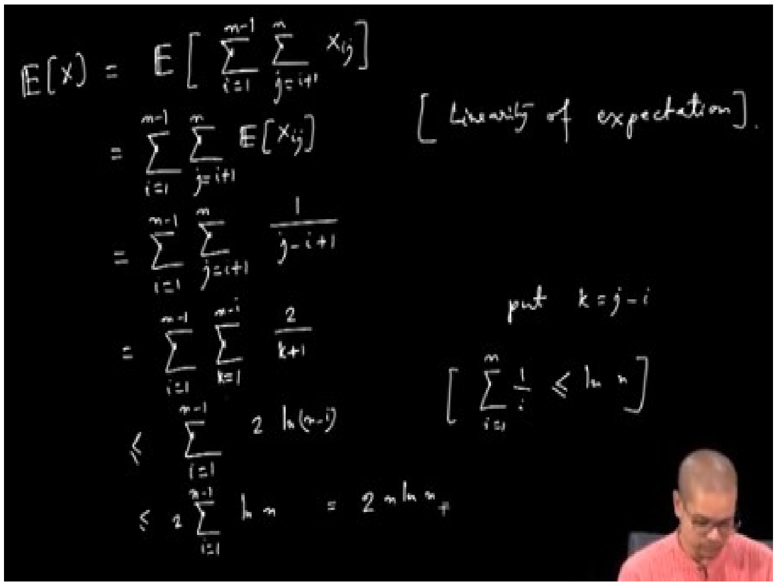
So for that we define for every i, j 1 to n. So let see be the final sorted array in C of course also look like this you see now we define a variable a random variable $X_{ij}$ to be is an indicator random variable it takes one if certain event happens. If $C[i]$ and $C[j]$ are compared in the ins in A in the run of the in the execution of the randomized quick sort algorithm if $C[i]$ and $C[j]$ are compared and 0 otherwise. So what is the probability what the X expectation of $X_{ij}$ is because it is a random indicator random variable this takes value one this is one times probability of this event.

That $C[i]$ and $C[j]$ are compared plus 0 times something 0 times the complement even but 0 time so that term disappears so this is just probability that $C[i]$ and $C[j]$ are compared. Now when does it they could be compared so you know comparison always happens in this partition step all the computation is and all the comparisons are happening in the partition step and these partition algorithm compares elements with the pivot. So $C[i]$ and $C[j]$ will be compared if either $C[i]$ or $C[j]$ one of them has been chosen as at the as a pivot.

And at that particular call of that quick sort routine $C[i]$ and $C[j]$ belongs to one set it is not that they have been separated by some other people for example if an element in between $C[i]$ and $C[j]$ is picked as a pivot then I know I will be in the left part $C[i]$ and $C[j]$ will be in the right part. And they will be separated and they will be never compared so these probabilities among these elements from $C[i]$ and $C[j]$ the first time a pivot is picked is one of $C[i]$ and $C[j]$.

This is 2 by j - i +1 there are i - i +1 elements and if either $C[i]$ or $C[j]$ are picked as pivot among this j - i +1 elements then only $C[i]$ and $C[j]$ are compared otherwise they are not compared. So now let me define X to be the number of comparisons so this is $\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$. And you can assume i is less than j because $X_{ij}$. And $X_{ij}$ they carry the same in same meaning so to avoid repetition we assume that j is strictly more than i it is sum of $X_{ji}$.

**(Refer Slide Time: 22:45)**



$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

[Linearity of expectation].

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{j-i+1}$$

put $k = j - i$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$\left[\sum_{i=1}^{n} \frac{1}{i} \leqslant \ln n\right]$$

$$\leqslant \sum_{i=1}^{n-1} 2 \ln(n-i)$$

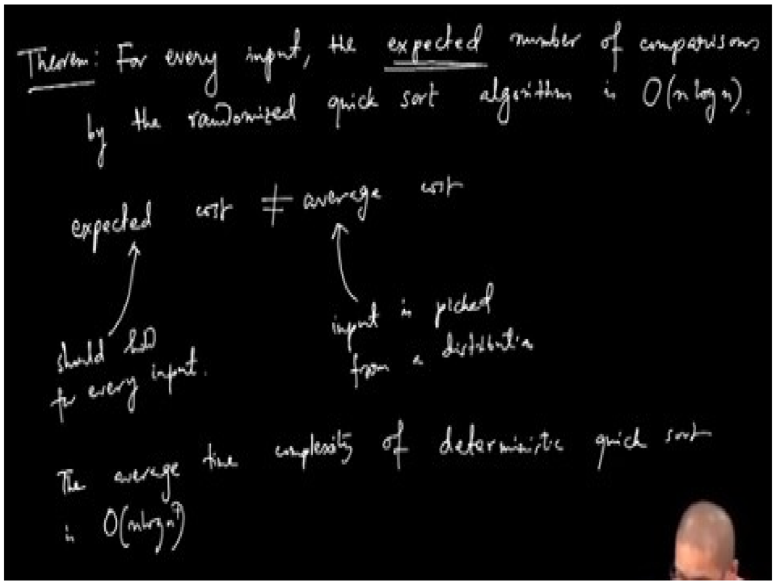
$$\leqslant 2 \sum_{i=1}^{n-1} \ln n = 2n \ln n.$$

Now then expectation of X from linearity of expectation or first let me write this way expectation of this $\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}$ now apply linearity of expectation. So linear is from linearity of expectation it basically says that you can exchange expectation and finite or countable sum. Now this is summation i =1 to n. this expectation is probability that $C[i]$ and $C[j]$ is compared and that is 1 by j - i +1.

This is actually this index can go I can go till i -1. Because j has to be strictly more than i for i = n that there is no inner sum foreign to n - 1 now you see that g - if we if we put k = j - i then j - i starts from one and it goes till n - i $\frac{2}{k+1}$. And then this is less than equal to summation i =1 to n -1 summation k =1 to ∞. No not Infinity this inner sum is then bounded by these $2\ln n$ is because you know summation on n – i.

Summation i over i = 1 to n is less than equal to login is a standard inequality you can prove this using calculus or area method there are there are various ways to prove it so each term is less than login.

**(Refer Slide Time: 26:56)**



Theorem: For every input, the expected number of comparisons by the randomized quick sort algorithm is $O(n \log n)$.

expected cost $\neq$ average cost

should hold for every input.

input is picked from a distribution

The average time complexity of deterministic quick sort is $O(n \log n)$?

So this is equal to twice in login so what you proved is theorem here is an important thing that for every input. The expected number of comparisons by the randomized quick sort algorithm is $O(n\log n)$. So you put special attention on this term expected so expected cost you know or in this case in this case expected number of comparisons this is not equal to average cost. Average is used when you know input is picked input is picked from a distribution. Whereas the expected you know it should hold for every input.

So here the expectation is over the randomness of the algorithm that is the important thing. For example the average time complexity for quick sort is $O(n\log n)$ so for example the average time complexity of deterministic quick shot which is still $O(n\log n)$. Because you are picking input randomly but the expected time complexity of deterministic of quick sort is $O(n^2)$ because there are inputs where this algorithm will make $O(n\log n)$ time. So let us stop here.