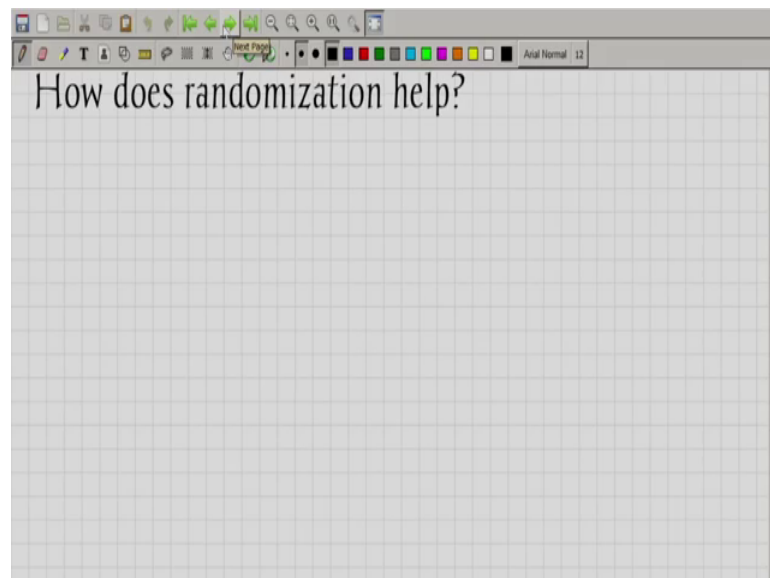


Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 01
Introduction to Randomized Algorithms

Welcome to the first lecture on Randomized Algorithms. The textbook for this course that we will be following is the textbook by Michael Mitzenmacher and Eli Upfal that name of the book is Probability and Computing. And the second textbook for the course will be Randomized Algorithms by Motwani and Raghavan.

(Refer Slide Time: 00:59)



So, before we study randomized algorithms, let us first try and understand in what way does randomization help. If we have a deterministic algorithm, one possibility is that for every deterministic algorithm, the adversary could choose a particular input which could make our algorithms perform extremely poorly. So, for such adversaries, randomization can probably help that is after taking the input we do not mind you, we are not choosing a random input, but let us say we take the input given by the adversary and modify it in a certain sense in a random way over which the adversary has no control.

So, this is possibly one way in which randomizations could help in a wide variety of computations. We will see a more examples of how randomization helps, but let us first

(Refer Slide Time: 02:07)

Let me describe the randomized quicksort algorithm and then we will again revisit these issues. So, the first step is choose a number y uniformly at random from S . So, each a 1 to a n is equally likely to be chosen in this step. The second step: construct the subsets S_1 containing all elements of S which are less than y . And construct the subset S_2 containing all elements of S which are greater than y . And then the third step would be

the recursive step, recursively sort S_1 and S_2 , and then output S_1 that is a sorted version of S_1 , followed by y followed by S_2 , so that will give us the sorted version of the set.

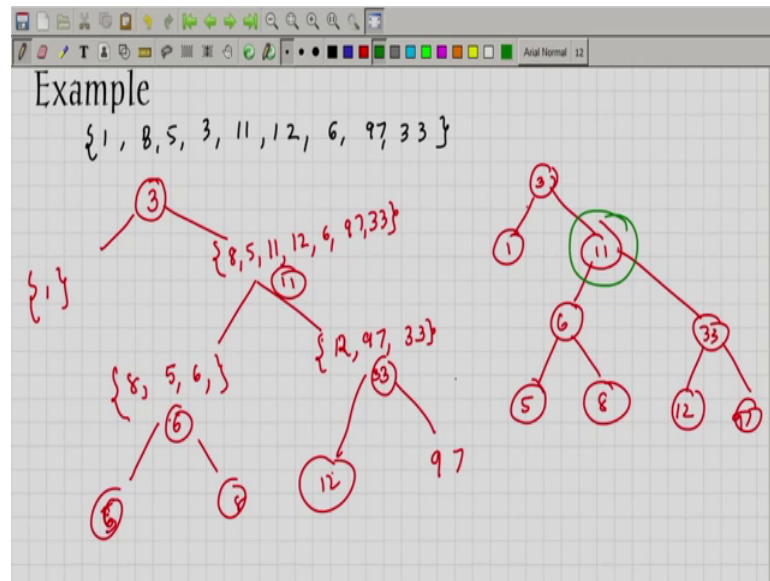
Now, what is the randomness, how does it affect the input-output characteristics of this algorithm? The only place where we are using randomness here in this algorithm is in the choice of the random element from it. We know that for this algorithm, the answer returned will always be a sorted input, but the running time for this particular algorithm the running time is a random variable, the running time is a random variable.

Now, when we say that the running time is a random variable, does it depend on the initial set of integers or does it depend upon the choice of y ? You can see that it is just a function of the choices that we make inside the algorithm; it is not really dependent on the input integers. So, this is going to this algorithm is going to give a random running time. What we are interested in is the expected running time of this algorithm. We could also look at the probability that the algorithm fails to complete its task in a given amount of time ok. So, those are the kind of questions that will be that we will be looking at when we study randomized algorithms.

So, here we mentioned that the running time is a random variable, and it depends only on the choices made inside the algorithm. In particular, if you look at let us say bubble sort, there are certain inputs on which the algorithms could do particularly bad, but here there is no input on which we can guarantee that the algorithm will do poorly. In fact, for every input, no matter what the input is the running time is only a function of the choices that we make inside the algorithm depends only on the random choices that we make inside the algorithm.

So, when we get the expected running time, it means on every input the expected running time is such and such as supposed to we are not saying that for the average input the running time is such and such. We are saying for a every input the running time must going to be a certain or the expected running time is going to be a certain value. So, let us do an example of this randomized quicksort.

(Refer Slide Time: 09:59)



So, let us say our set of numbers is following 1, 8, 5, 3, 11, 12, 6, 97, 33 ok. Suppose, the first element that we pick is 3 ok, so let us say 3 is the first element that we picked. When we split this, we have one set of numbers namely one and everything else is greater than 1 namely 8, 5, 11, 12, 6, 97, and 33 again this is already sorted, so nothing to be done there single element set.

From here we again choose a random element. Let us say if we had chosen 6 then this again gets split into two parts. If 5 will be on one side, let us say this time we will choose 11 instead of 6. So, if we chose 11 on the left side would be 5 sorry 8, 5, 6 and on the right side will be 11 sorry 12, 97, and 33 and let us say here we had picked 6, this gets split into 6 and 8. And here if we had picked 33 and this get split into 12 and 97 ok. So, this is how the algorithm works.

And basically I have drawn the answer the computation in a tree format, so that consists of these nodes. So, if we do a in order traversal of this tree, I will get the sorted list 1, 3, 5, 6, 8, 11, 12, 33 and 97. So, this is how the algorithm works. If we had instead of 3, chosen a different number at the start we would have got a different tree. And depending on that tree the running time the algorithm could vary. What we are interested in is on and on the average how much time does this algorithm take, what is a expected running time of this algorithm. We do that analysis ok.

(Refer Slide Time: 13:09)

Analysis

$S_i \triangleq$ i th in S with rank i .

$X_{ij} = \begin{cases} 1 & \text{if } S_i \text{ \& } S_j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$

$X = \sum_{i < j} X_{ij} = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij}$ (# of comparisons)

$E[X_{ij}] = 1 \times \Pr\{X_{ij}=1\} + 0 \times \Pr\{X_{ij}=0\} = P_{ij}$

$E[X] = \sum_{i < j} E[X_{ij}] = \sum_{i < j} P_{ij} = \sum_{i=1}^n \sum_{j=i+1}^n P_{ij}$

Fact: S_i \& S_j are compared iff S_i is a parent of S_j (or vice versa)

Aim: Compute $E[X]$

Fact: S_i \& S_j are comp. iff S_i or S_j is chosen ahead of the els in b/w S_i \& S_j .

Diagram: A horizontal line with points labeled $i-1$, S_i , S_j , and $i-j+1$ elts. S_i is circled and labeled (10), S_j is circled and labeled (20).

So, let us introduce some random variables, but before that let us say S_i denote the element in S with rank i . So, the i th smallest element in S , we will denote it by S_i and let us. So, this is not a random element this is a fixed S_i is a fixed element if you are given a particular input S_i is the i th smallest element in S . Now, let us introduce a random variable X_{ij} which we will define to be one if S_i and S_j are compared during the course of the algorithm and equal 0 otherwise ok.

So, if you look at any particular element S_i and S_j if they are compared during the execution of the algorithm then we will have X_{ij} equals 1, otherwise we will have X_{ij} equals 0. You can note that X_{ij} equals 1 would imply that these are compared and that is the only I mean any element i and j are compared at most ones; certain elements may not be compared at all. So, the total number of comparisons gives us a bound on the running time of the algorithm. So, if you look at summation X_{ij} or all i less than j ok, this can be written as summation i going from 1 to n summation j going from $i+1$ to n X_{ij} ok. So, this is the number of comparisons.

We want to estimate how much is this? So, let us see when will X_i and X_j be compared. So, here you can see that anything in the left sub tree will not be compared with anything on this right sub tree. For example, if you look at 11, based on 11 so all the elements below 11 were compared with 11 that is how we split it into two parts. But 6 is not going to be compared with 33 ok, because they fall in different sub trees. Similarly, one is not

going to be compared with 11 everything got compared with 3, but they are not going to be 1 is not going to be compared with 12

So, S_i we can write this fact as S_i and S_j are compared if and only if S_i is a parent of S_j or vice versa. If they fall in distinct sub trees, they are not going to be compared one element is a parent of the other that is the only case in which they will be compared. So, now, this is when X_{ij} is equal to 1. So, let us call this number of comparisons as X . What we are interested in is so our aim is to compute the expectation of X by linearity of expectation we know that this summation this expectation is just the sum of the expectation on the individual random variable. We can write that expectation of X is equal to sum over say i less than j expectation of X_{ij} , X_{ij} being an indicator random variable its expectation is going to be just its the probability of the event that indicates.

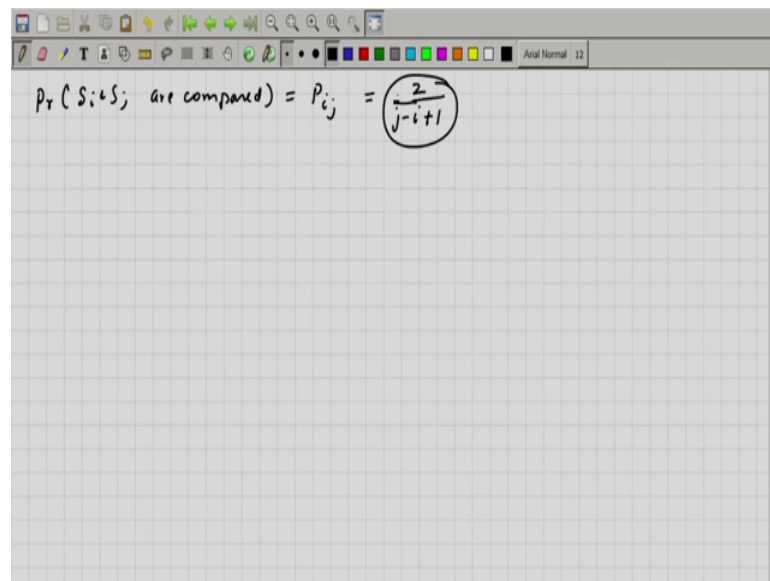
So, expectation of X_{ij} is equal to 1 into probability that X_{ij} is equal to 1 plus 0 into probability that X_{ij} is equal to 0. So, this we will call this as P_{ij} . The probability that X_{ij} is equal to 1, we will indicate it by P_{ij} . So, therefore, expectation of x is going to be summation over i less than j P_{ij} ok. Let us compute that probability how much is P_{ij} going to be. So, we can also write it as summation over i going from 1 to n , j going from i plus 1 to n P_{ij} .

So, let us look at the element S_i and the element S_j . When will they be compared? So, there are i minus 1 elements here, in between there are certain elements and after this there are certain elements. Now, in our algorithm, we are randomly choosing various pivots. In this interval, from i to j there are precisely i minus j plus 1 elements ok. So, if this is the 10th element and this is the 20th element then in between up from 10 to 20, including 10 and 20, there are 11 elements in our algorithm at each stage we are randomly choosing a an element y which we may call as a pivot element.

So, if an element in between S_i and S_j is being chosen as the pivot element before S_i then S_i and S_j will not be compared. For example, let us say if this element it is called this is alpha, if alpha was chosen before S_i and S_j , then what we can say is alpha will be compared to this side it will go into one subtree and as S_j will go into another subtree, and there is no possibility of comparing S_i and S_j . The only case in which they will be compared is when out of these elements, these i minus j plus 1 elements either S_i or S_j is picked before every other elements inside this ok.

So, I will write this as a fact S_i and S_j are compared if and only if S_i or S_j is chosen ahead of the elements in between S_i and S_j ok. So, we can imagine the entire algorithm in the following way. We choose a random permutation of 1 to n that tells us the elements that we are going to pick as pivots in each stage of the algorithm ok. So, in amongst these permutations, we need to find out the number of permutations in which S_i or S_j appears before an element in between S_i and S_j .

(Refer Slide Time: 21:51)



$$Pr(S_i < S_j \text{ are compared}) = P_{ij} = \left(\frac{2}{j-i+1} \right)$$

So that happens with the so probability that S_i and S_j are compared is equal to be denoted this by P_{ij} , this is equal to 2 divided by j minus if you assume i to be the larger element, assume i to be the smaller element, then this becomes j minus i plus 1. While considering a random permutation π , let x be the first element chosen from S_i to S_j by symmetry each element in the set S_i to S_j is equally likely to be x .

But if x is anything other than S_i or S_j , S_i and S_j will not be compared during the sorting process. So, the probability that S_i and S_j are being compared is equal to the total number of favorable choices for x which is 2 divided by the total number of choices that we can make which is j minus i plus 1. So, P_{ij} becomes 2 divided by j minus i plus 1.

(Refer Slide Time: 23:11)

$$P_{ij} = \frac{2}{j-i+1} \quad \therefore \sum_{i < j} P_{ij} = \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} = \sum_{i=1}^n \left(\sum_{k=2}^n \frac{2}{k} \right) \leq H_n$$

$H_n \triangleq$ n^{th} Harmonic Number

$$= \frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n} \approx \log n$$

$$\leq \sum_{i=1}^n H_n \approx n \log n$$

So, therefore, summation i less than j P_{ij} will be equal to summation i going from 1 to n summation j going from $i+1$ to n by $j-i+1$ ok. So, when j equals $i+1$ this quantity is summation i equals 1 to n , this will vary from k equals 2 to n by k ok. So, this summation is at most. So, this is less than the inner summation is less than H_n . So, this part is H_n . So, we can say that the whole summation is less than i going from one to n H_n , where H_n is the n^{th} harmonic number its equal to $1 + \frac{1}{2} + \dots + \frac{1}{n}$, this is approximately $\log n$ ok.

So, the overall summation is going to be less than $n \log n$. What you can say its approximately $n \log n$ ok, so that tells us that randomization can help us come with very simple algorithms with very good running time. We will now look at yet another problem in where randomization helps us which is the problem of matrix verifying matrix multiplication.

(Refer Slide Time: 25:21)

Verifying Matrix Multiplication

Input: A, B, C A, B, C are $n \times n$ matrices over F .

Output: YES if $AB = C$
 NO otherwise.

Algorithm

1. Choose a vector ($n \times 1$ matrix) randomly. (x)
2. Compute ABx & Compute Cx .
3. Return YES if $ABx = Cx$. (O(n^3))
 Return NO otherwise.

Complexity Analysis:

- $A \times B$: $O(n^3)$
- C : n^2
- x : n
- Total: $O(n^3)$

Notes:

- Could give incorrect answers.
- If $AB \neq C$
- $AB \neq C$
- $x = [x_1, x_2, \dots, x_n]^T$

So, let us take the problem. The input is three matrices. So, we can assume that A, B, C are n cross n matrices or some suitable field. The elements come from a field if you can think of them as real numbers ok. Now, what we are interested in is the output should be yes if A times B is equal to C ; and this should be no otherwise ok. So, if the matrices A and B multiply and give us the matrix C , then the answer is yes; otherwise the answer is no.

Let us think of an algorithm to do this we could simply multiply the matrices A and B and then compare it with the matrix C that is going to take us this multiplication if we do a Naive multiplication we will require n cube steps, and then we can compare it with C that is going to be taking n square steps. So, the running time of this algorithm a deterministic algorithm is bounded by n cube, you need you require no more than $O(n^3)$ of course, we could do faster multiplications of the matrices in which case we will get something like n raise to ω , where ω is the exponent of matrix multiplication ok. Let us say something like 2.5 whatever is the value of ω it is still greater than the input size.

The input size is n square. There are three matrices of size n cross n which means the input is of size three n square. So, can we get an algorithm which will do this task better than let us say these algorithms by I mean instead of can we do it without multiplying out matrices A and B , is there some other way?

That was a simple technique ok. So, write down the algorithm choose a vector. So, we will think of this as a $n \times 1$ matrix randomly. So, all possible $n \times 1$ matrices or vectors are equally likely. Compute so this vector let us call it as x compute $A B x$ and compute $C x$, return yes if $A B x$ is equal to $C x$, return no otherwise. This is a better algorithm than the previous algorithm where we just computed A , B and C and then compared them. But this algorithm could give incorrect answers, there are certain inputs on which the algorithm could give incorrect answers, but there are certain inputs the algorithm will never give incorrect answers.

For example, if $A B$ is equal to C , no matter which random vector you take, you are always going to get $A B$ you are going to get the answer yes ok. But there are certain inputs on which the algorithm could give incorrect answers. For example, if A is not equal to C , then there are certain x s that you could choose which could result in the answer, yes, when the correct answer would have been no. For example, when if you have two matrices says that $A B$ is not equal to C , and the random vector that you chose was unfortunately let us say the all zero vector. If you multiply that with $A B$, you will get 0. And if you multiply it with C , then also you will get 0. And zero being equal to the zero vector being equal to the zero vector, the answer would be yes whereas, the correct answer would have been no ok.

So, clearly this algorithm does not have some properties that the earlier algorithm did not have. For example, it gives incorrect answers ok, some certain undesirable properties. But what have we gained out of this I mean by introducing I mean we now have a chance of committing mistakes, what did we gain by doing that. Well, the running time of this algorithm is much better than the previous algorithm, because here the matrix multiplication we will compute it in the following way A times B times x .

Now, B being an $n \times n$ matrix and x being an $n \times 1$ matrix, the multiplication only takes $O(n^2)$ because we could just take the matrix B and multiply it with x so these elements are multiplied with these elements ok, so that is that takes $O(n^2)$ to compute one element we have let us say n elements to compute each of them can be computed in linear I mean $O(n)$ order of n time. So, total running time is $O(n^3)$. So, this is this computation can be done in $O(n^3)$, and you will get $n \times 1$ matrix which again can be multiplied with A which is an $n \times n$ matrix and that also takes $O(n^3)$.

So, this entire computation can be done in $O(n^2)$ time and then you have to compare it with you to compute Cx and compare with it. So, the total running time is $O(n^2)$ ok. So, this computation this computation takes only $O(n^2)$ time. So, we have saved on computation computational effort, but we have introduced errors. So, we will try and analyze how bad is the error ok.

(Refer Slide Time: 32:43)

The image shows a handwritten analysis on a grid background. At the top, it says "Analysis". Below this, there are two cases for the comparison $AB \neq C$ and $AB = C$. For $AB \neq C$, a decision tree shows "YES" and "NO" branches, with "NO" circled. For $AB = C$, the result is "YES". The main analysis is for the case $AB \neq C$, where $D = AB - C$ and $D \neq 0$. It states that the algorithm returns YES only when $Dx = 0$. The probability $P_r(Dx = 0 \mid D \neq 0)$ is analyzed. A matrix D is shown with elements $d_{11}, d_{12}, \dots, d_{1n}$ in the first row. The equation $d_{11}x_1 + d_{12}x_2 + \dots + d_{1n}x_n = 0$ is written. A vector $x = [x_1, x_2, \dots, x_n]^T$ is shown. The analysis concludes that the probability is at most $\frac{1}{|F|}$, which is less than or equal to $\frac{1}{2}$. The final result is $x_1 = -\frac{\alpha}{d_{11}}$.

So, let us look at the matrix AB times x being equal to Cx . This is the only case sorry. So, if AB is equal to is not equal to C , the algorithm could return two answers. If AB is equal to C the algorithm will always return the answer, yes, and that is the correct answer. When AB is not equal to C , the correct answer should have been no, but the answer the algorithm returned could be yes. So, we want to compute the following probability, probability that AB is not equal to C and the algorithm returns yes as the answer.

So, when does this happen? So, since AB is not equal to C we can look at the matrix AB minus C let us call this is the matrix D ok. So, the algorithm returns yes only when is D times x is not equal to this is equal to 0. We know that D is not equal to 0, but D times x is equal to 0. So, we will look at this probability that Dx is equal to 0 under the condition that D is not equal to 0 ok. So, D is this matrix which contains lot of elements let us say $d_{11} d_{12} \dots d_{1n}$ the d_{n1} to d_{nn} ok. So, this times x_1, x_2, x_n is equal to 0.

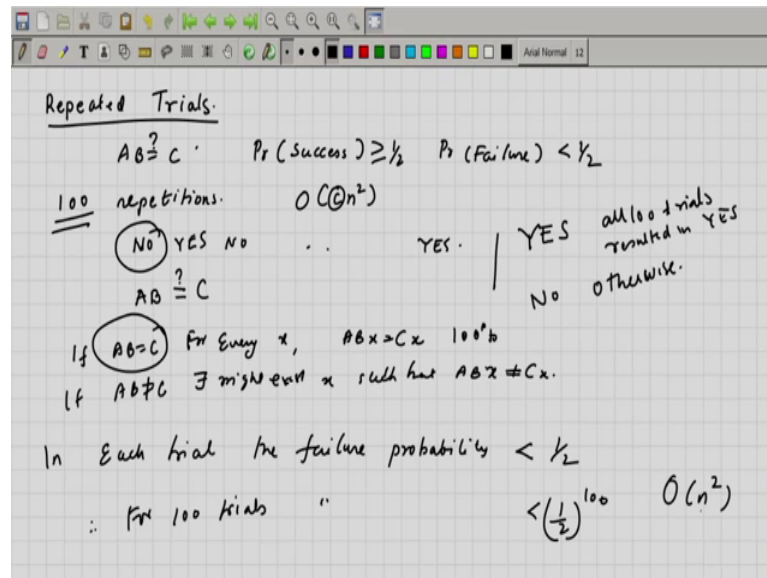
We want to look at what is the probability of. So, this is our matrix D which is known to be a nonzero element at least one of the elements inside this is nonzero element. And we want to look at the amongst the choices of x_1 to x_n , how many of those choices would result in making this product 0 that will basically give us a bound on the error probability. So, let us we can we can assume without loss of generality that d_{11} is the nonzero element. If d_{11} was not the nonzero element, we could carry out this entire analysis by shifting rows of D and columns of d to make d_{11} first element and then x_1, x_2, \dots, x_n has to be chosen appropriately.

So, let us say that d_{11} is nonzero. Now, if we get this entire matrix to be 0, clearly one requirement is that the first element in the column vector $d \cdot x$ the if you consider the column vector $d \cdot x$, the first element inside that should be 0 that would give us this equation $d_{11}x_1 + d_{12}x_2 + \dots + d_{1n}x_n$ should be equal to 0. So, let us imagine that these choices the choices for the random variables x_1, x_2, \dots, x_i mean x_2 up to x_n . So, these random variables let us say that their choices were already made ok, that gives us some particular vector with x_1 not yet being chosen.

Now, once you fix these x_2 to x_n , this expression $d_{12}x_2 + \dots + d_{1n}x_n$ they get fixed and let us say that value is α ok. Now, we can look at this as a linear equation $d_{11}x_1 + \alpha = 0$, therefore, x_1 is equal to $-\alpha/d_{11}$. In other words, there is precisely one value of x_1 that you can choose which will result in $d \cdot x$ being 0. So, if the random vector was chosen in such a way that each elements x_i was chosen from a field of size let us say f , then only one choice out of f choices would result in the algorithm returning an incorrect answer ok.

Even if the field was of size two the probability of error is less than half because the probability of error is going to be less than one by size of the field which is certainly. So, this quantity is certainly less than half ok. So, we know that the probability that the algorithm makes a mistake is no more than half, but is that good enough. So, we will do something that we will be using again and again which is the notion of repeated trials.

(Refer Slide Time: 38:41)



So, we have an algorithm which checks if A B equals C, and it returns the correct answer with probability greater than half ok. So, probability of success is greater than half or in other words probability of failure is less than half. By success what we mean here is that the algorithm returns the correct answer, this is an algorithm which returns the correct answer every time when A B is equal to C. And it returns the correct answer with probability greater than half then A B is not equal to C.

Now, let us just repeat the algorithm 100 times. And we will note down the answers returned by the algorithm. If the first time the answer is no, and the next is yes and no and so on ok, what can we say. If A B where equal to, so we can we just repeated the algorithm 100 times, so the running time is just C n square where C is an absolute constant here it is let us say we repeated it 100 times. So, 100 n square as the running time. If now if this was the output we know that if A B we are equal to C, then the algorithm would never return the answer no ok, because on all inputs A and B where which are equal. So, if A B is equal to C, every, for every x, A B x is equal to C x. If A B is not equal to C, there might exist x such that A B x is not equal to C x ok.

So, here if it says if the algorithm says no, that means, the algorithm has found out some x such that A B x is not equal to C right. So, even if there was one such the answer is going to be we can conclude that A B is not equal to C. So, even in this 100 repetitions after processing this we will say the answer yes only if all 100 trials resulted in yes; and

we will say no when all trials meaning so we will say no otherwise ok. So, you know we could also just stop at the first time a no appears that is equivalent, instead of doing it 100 times we can continue doing 100 times if all the previous runs resulted in yes.

Now, what is the probability of success of this algorithm well each of these repetitions being independent of each other if $A = B$ were equal to C , we will we will our algorithm will be correct with 100 percent probability. If $A \neq B$ is not equal to C , each trial the failure probability is less than half each style be argued that the success probability is greater than half. Therefore, in each trial the failure property sorry the failure probability is less than half. Therefore, for 100 trials the failure probability is less than half to the power 100, only if each of the trials fail, we will be giving the answer no ok.

So therefore, this is this very small number. So, we still have a $O(n^2)$ algorithm with very small error probability or in other words we have very high error probability high success probability. We will stop here for today; continue our study of randomized algorithms in the next lecture.