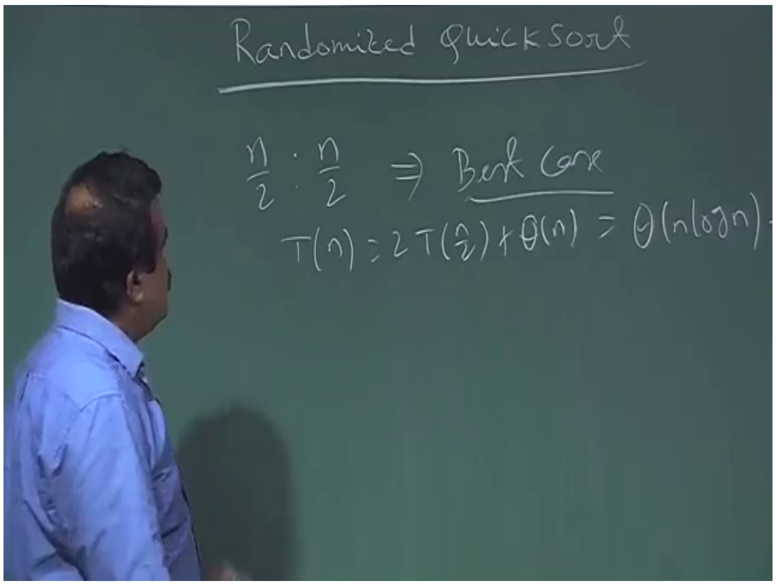
An Introduction to Algorithms

Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 12
Randomized Quick Sort

So we are talking about we are doing the analysis of quick sort and we have seen if the pivot is a good pivot then it will partition the array into n by 2 is to n by 2.

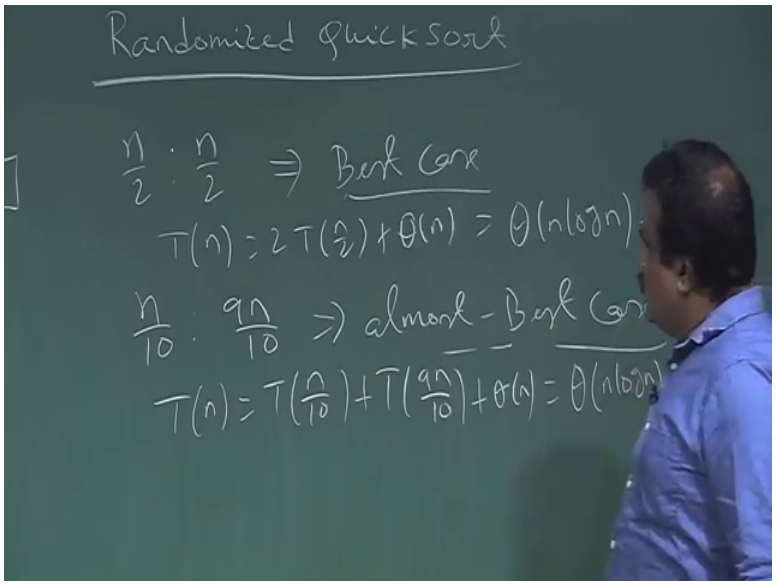
(Refer Slide Time: 00:31)


Randomized Quick Sort
$\frac{n}{2} : \frac{n}{2} \Rightarrow$ Best case
$T(n) = 2T(\frac{n}{2}) + \theta(n) = \theta(n\log n)$

So, this is the best case. So, if our partition algorithm will partition the array into this way then it is the best case in that case we have the recurrence is 2 T n by 2 plus theta of n and that will give us order of n log n.

So, this is the lucky case order of n log n and even if we see here even we have seen if this is not n by 2 is to n by 2 if it is just if we can ensure the partition will put little fraction over here say 1 by 10 is to 9 by 10.

(Refer Slide Time: 01:25)


Randomized QuickSort

$\frac{n}{2} : \frac{n}{2} \Rightarrow$ Best Case

$T(n) = 2T(\frac{n}{2}) + \theta(n) = \theta(n\log n)$

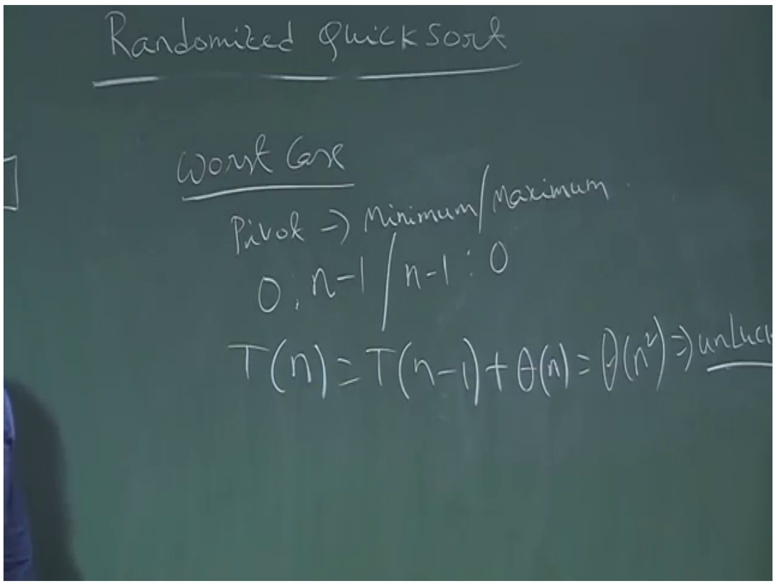
$\frac{n}{10} : \frac{9n}{10} \Rightarrow$ almost - Best Case

$T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \theta(n) = \theta(n\log n)$

So that means, if the partition is n by 10 is to 9 n by 10 then we have seen this is the almost this is referred as almost best case quite this is also best case because for this we have we have seen the recurrence is T n by 10 plus T 9 n by 10 plus theta of n this is the recurrence corresponding to this partition and this will give us also order of n log n.

So, this is also lucky case. So, even if one can ensure if the partition can ensure that it will partition some little portion guaranteed some little fraction in one side and other side the remaining element then also you see it is the lucky case and which whichever is the unlucky case the worst case worst case is basically 0 is to n minus 1; that means, if we choose the pivot element to be minimum or maximum.

(Refer Slide Time: 02:31)


Randomized Quick Sort

Worst Case

Pivot $\to$ Minimum / Maximum

$0 : n-1 \,/\, n-1 : 0$

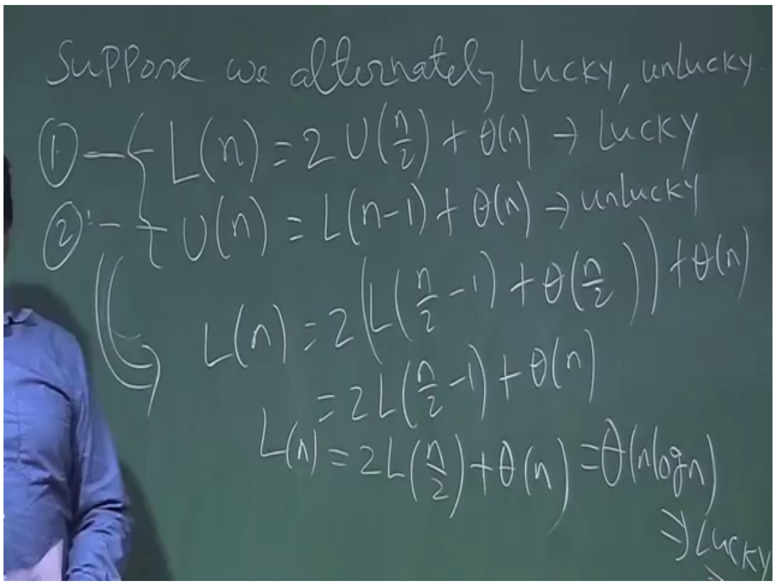
$T(n) = T(n-1) + \theta(n) = \theta(n^2) \Rightarrow$ unluck

So, the worst case is worst case for quick sort is if the pivot is if the pivot is minimum element minimum or maximum element from the array in that case the partition will be 0 is to n minus 1 if it is minimum or n minus 1 is to 0 so; that means, one sided nobody is there another sub array is containing all the elements. So, in that case recurrence will be T n is equal to T n minus 1 plus theta n and this will give us the solution order of n square.

So, this is the worst case and this is we refer as unlucky case we are unlucky if the time complexity is coming to be order of n square so. So, now, will do a alternative will do a another type of analysis suppose we are in our. So, we have given a array. So, we call the partition algorithm. So, it is partitioning into 2 part if it is good if it is not minimum or maximum then it will partition into nor 0 into n minus 1. So, it is a lucky case now suppose in the next time will be unlucky. So, if we have suppose we are alternatively lucky unlucky.

So, then we want to see whether eventually will be lucky or not. So, that is the analysis we want to do. So, this is the more intuition.

(Refer Slide Time: 04:15)



Suppose we alternately lucky, unlucky.

$$① :- \left\{ L(n) = 2U\left(\frac{n}{2}\right) + \theta(n) \rightarrow \text{lucky} \right.$$

$$② :- \left\{ U(n) = L(n-1) + \theta(n) \rightarrow \text{unlucky} \right.$$

$$L(n) = 2\left(L\left(\frac{n}{2}-1\right) + \theta\left(\frac{n}{2}\right)\right) + \theta(n)$$

$$= 2L\left(\frac{n}{2}-1\right) + \theta(n)$$

$$L(n) = 2L\left(\frac{n}{2}\right) + \theta(n) = \theta(n\log n)$$

$$\Rightarrow \text{Lucky}$$

Suppose we alternately lucky unlucky. So, we have given a array of size n slowly we call the partition it has reducing the array into 2 sub array then again we call the partition. So, we have subsequent call. So, so suppose in the first sort we are lucky in the next sort will be unlucky. So, alternatively will be lucky unlucky like this sequence and then we want to see whether eventually we are lucky or not.

So, that analysis will do. So, so; that means, now suppose we are luck. So, this is the lucky recurrence. So, if we are lucky now then it will be partition the array into 2 equal part half half and then the recurrence will be two, but if we are lucky now next time we know we are going to be unlucky. So, that is the unlucky recurrence plus theta of n. So, this is corresponding to lucky recurrence and what is the unlucky recurrence now suppose we are unlucky now if we are unlucky so; that means, it is the worst case.

So; that means, it will divide the array into 0 is to n minus 1 so; that means, out pivot is minimum or maximum from the given input. So, in that case the recurrence will be, but now if we are unlucky then next time we know will be lucky. So, that is why this unlucky means it will be n minus 1 is to 0 or 0 is to n minus 1. So, the recurrence will be l of n minus 1 plus theta of n. So, this is corresponding to the unlucky recurrence. So, now, we want to solve this to get the; so, if this is 1 this is 2 now we put these 2 in 1.

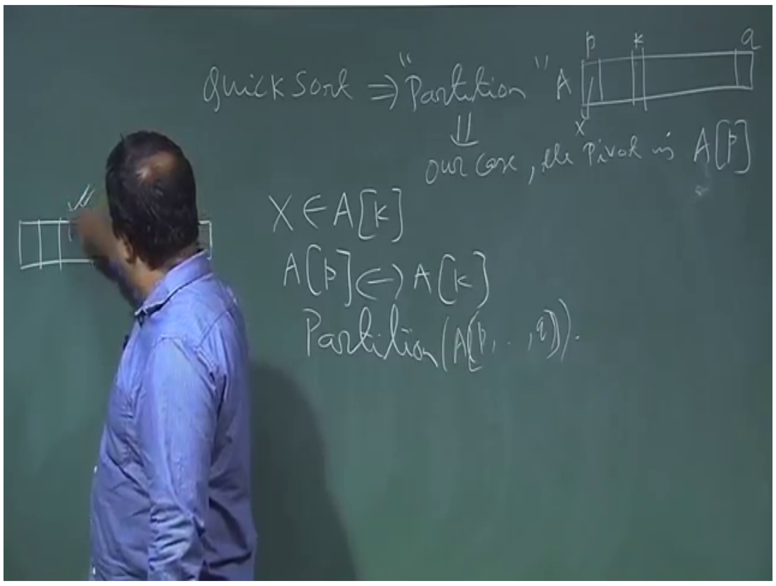
So; that means, this will give us quite l of x 2 of so, now, we can put l of n by 2 minus 1 plus theta of n by 2 we are just putting this u u of n u of u of n by 2 u of n is this recurrence. So, we have n we are putting just n is equal to n by 2 this plus theta of n. So, this will give us basically 2 of l of n by 2 minus 1 plus theta of n because this 2 of theta of n by 2 will be combined with the theta of n and it together will give us theta of n. So, now, this minus 1 is will play role of like round offing I mean lower ceiling or upper ceiling.

So, this we can easily write as 2 of l of n by 2 plus theta of n. So, this is basically our l of n. So, recurrence for time complexity, these will give us the solution order of n log n. So, lucky case, this is lucky, lucky day. So, we are lucky. So, this is the observation this is the observation what. So, suppose in the whole sequence of execution. So, we have a big array and we are calling the quick sort. So, it is partitioning and then it is dividing into 2 sub array and again we call the quick sort in the sub sequence say it is further partitioning like this.

So, in this whole execution suppose we are fifty percent lucky fifty percent unlucky so; that means, if we are lucky unlucky lucky unlucky in this way then finally, our time complexity will be l log n. So, finally, we are lucky. So, this is this is more intuition that. So, if we can ensure that throughout our execution we are good amount of time of iteration I mean recurrence we are lucky then it will give us a lucky; lucky case. So, that will give us the idea of random as quick sort. So, if we can. So, if we can choose the pivot randomly then I mean then it is a there is a chance that the pivot will not be minimum or maximum.

So, it may be minimum or maximum, but there is a chance that it will not be minimum or maximum, that way, it may give us the lucky sequence; lucky recurrence. So, that way it will ensure sort of that eventually will lucky. So, that will do in the random way quick sort. So, in the whole sequence of execution if we can ensure at some number of times of this recurrence is lucky recurrence then eventually we will be lucky. So, this is one idea behind this randomized version of the quick sort and another idea is the adversary point of view. So, suppose we have a quick sort version and quick sort.

(Refer Slide Time: 09:55)

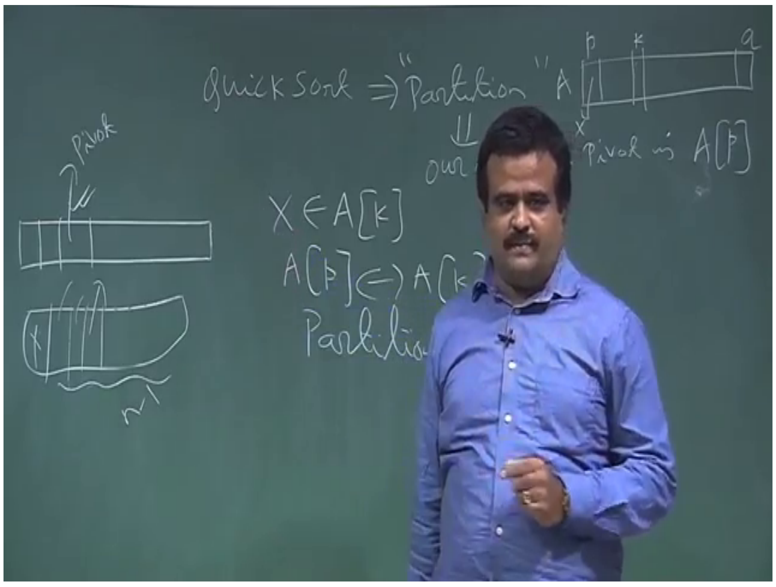


So, we have a; we have corresponding the partition algorithm partition sub routine now in our partition algorithm we choose the first element as a pivot. So, our partition our case we choose. So, our case the pivot is p basically the first element. So, if we have array starting from p to q. So, our case, the pivot is this one; x our partition algorithm. So, we know the position of the pivot suppose we want to take any other position as a pivot say third element from this array.

So, this is the k th element suppose we want the k th element it is not the k th (Refer Time: 10:54) or something it is the index k th index suppose want to. So, instead of first element if we want to take third element as a pivot in generally if we want to take k th element as a pivot. So, k th element means k th index. So, A k as a pivot say then we can change our partition algorithm or we can do what in the initially we will just change this A p and A k exchange A p and A k and then we call our original partition algorithm.

Because now we have exchange this pivot element in the first element then we call our original partition algorithm anyway that is that we can do we can modified the partition algorithm accordingly. So, now, we can call the partition algorithm with the index p to q now this as a if we fix the position of the pivot this as a drawback instead of first element.

(Refer Slide Time: 12:10)



If you fix this third element if we this is our say rule we always choose this is we always choose this element as a pivot in here we are choosing the k th element.

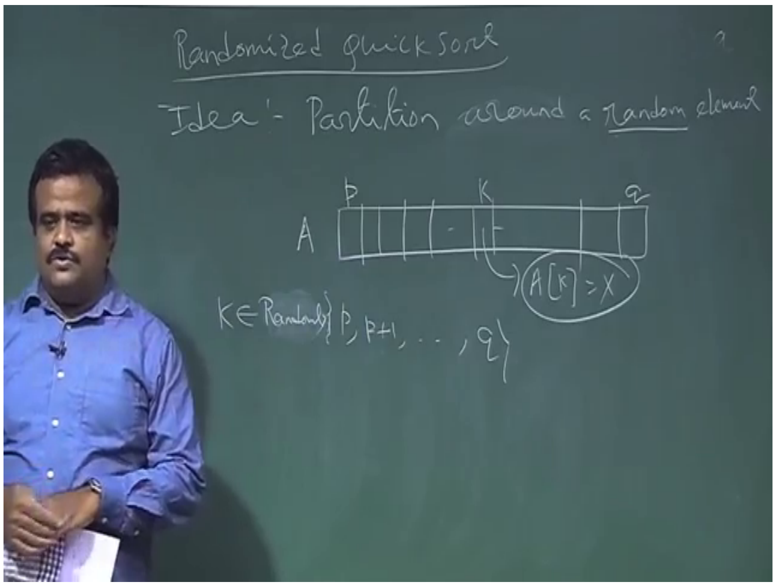
So, if k is three this is the pivot now what is the drawback of this suppose we design a algorithm the sorting algorithm and we give it to the somebody to buy this. So, what that company will do company will take this algorithm and company knows that I knows my pivot is the third element. So, the adversary always can construct a input where putting third element as a minimum or maximum of the given array. So, we can always do that. So, we put the; we have given array we put the minimum or maximum in the third position then we call the partition. So, again it will divide into.

So, if we put minimum again it will divide into just x will be here remaining all the n minus 1 element will be here again from this. So, from this sub array this is first element this is second this is third again we this will be going to be pivot again we put the minimum among this array remained here. So, this way one can always construct a input where always we are going to choose the pivot as a minimum or we can choose the maximum. So, that is the drawback. So, if we know before had if we know what will be the position of our pivot whether first element or third element then one can always come with an input.

Where by putting the minimum or maximum element in that position always throughout the execution sub sub sequence sub sub array then that will perform the worst case. So, that will always give us the 0 is to n minus 1 or n minus 1 is to 0 partition because that is happen to be a minimum or maximum of that corresponding array or sub array. So, is this clear? So, if we know the; knowing the position of the pivot is dangerous in that sense. So, we can always come with same input where our code will perform worst case I mean perform badly.

So, to avoid that the idea of randomized quick sort came, so, randomized quick sort idea is we will not let the others know which one which position we are going to choose as a pivot. So, that will determine at the run time. So, that is the randomly we choose the pivot element among this element. So, that will save us this type of constructing of this type of input where our code will perform badly. So, that is the idea of randomized quick sort. So, we talk about randomized quick sort. So, idea is to choose the pivot element randomly.

(Refer Slide Time: 15:09)


Randomized quick sort
Idea:- Partition around a random element
p
k
q
A
A[k] = x
k ∈ Randomly {p, p+1, . . . , q}

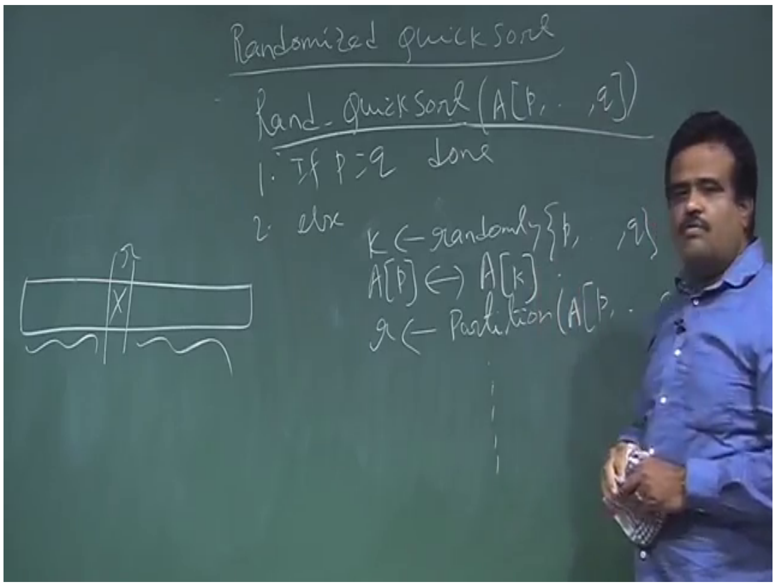
So, idea is to partition the array partition around partition around a random element random element. So, we have choosing a pivot randomly. So, we have given a array this is a array starting from p to q initially it is want to a. So, any element could be a pivot element any position. So, we just choose a random number k. So, this is say set index set p p plus 1 up to q from this index set we choose A k and k randomly we generate random number randomly. So, we choose a. So, if there are n element n element.

So, they are equally likely to come as a pivot element. So, this is 1 by n. So, we choose a k. So, from this index we choose A k randomly. So, this is k and this is going to be our A k is going to our pivot this is our x and this is a random choice. So, before running the code we do not know which k will be coming it depends on the random number generator may be you this p A p can be pivot may be A p plus 1 we do not know any one of this. So, any one of this element will can come with a probability 1 by n if there are n element. So, that is the idea.

So, we choose the pivot randomly from this set at the run time. So, before running the code nobody knows which is going to be the; which position which is going to be the pivot element. So, that is why adversary or nobody can come with some input where our code is performing bad because we do not know which position is going to be pivot. So, that is why we cannot put in that position minimum or maximum of that array. So, that is the idea. So, we choose the pivot randomly among this set. So, we have n element among this anyone of this can participate as a pivot. So, that will be decided at the rum time. So, run time we generated random number based on that we choose a index and that A k will be the pivot element.

So, this is the idea of randomized quick sort. So, we can just write a code for that. So, we can just quickly write a code rand quick sort.

(Refer Slide Time: 18:12)



```
Rand_QuickSort(A[p, .. ,q])
1. If p = q done
2. else  k ← randomly {p, .. ,q}
         A[p] ⟷ A[k]
         r ← Partition(A[p, . .
```

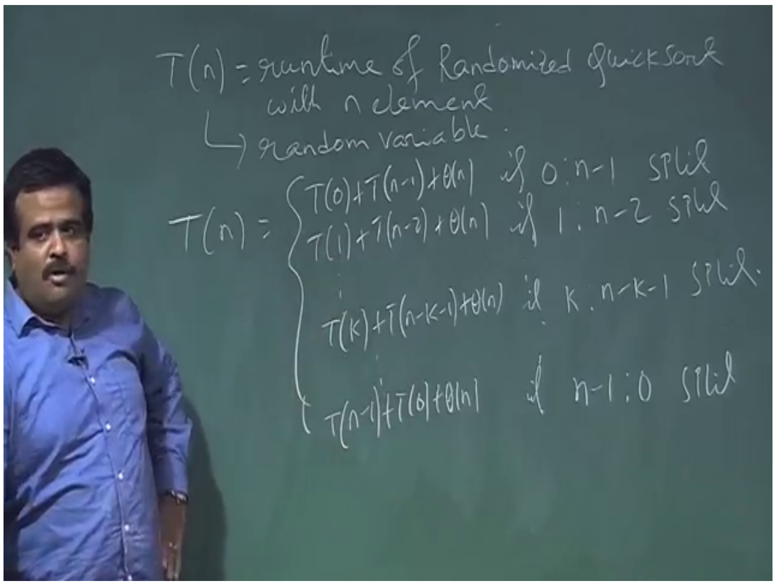
So, it is basically A p to k. So, now, we earlier version of the quick sort if A p is our pivot element, but here, again we have the initial condition if p is equal to q then done else what we do we have to call the partition algorithm and that partition should be random partition.

So, we call the else say r is equal to will have to explain what is the random partition from A p to q and even if we want to use our partition then what we do we know the our original partition is taking first element as a pivot then what we do we just call this we choose A k randomly from this set. So, generated random number we choose A k from this set and then A k is our pivot now we exchange A k with A p because we know the our partition is.

Basically taking first element as a pivot then we call the then we call this partition our original partition algorithm our first element as a pivot and then it will partition into 2 part and then we have that just the remaining call quick sort. So, it will partition into 2 part this is the r x will be sitting here this is the left sub array right sub array again we have to call the randomized quick sort on both the sub array and again on this sub array also we have to call a pivot randomly. So, that is very important.

So, that is very important. So, in a sub sequence step also we choose the pivot randomly. So, again for this sub array we have to call A k we have to generate a random index k and that will going to the pivot. So, similarly here and the initial call is rand quick sort from a 1 to n. So, this is the code for randomized quick sort now we want to analyze this code what is the run time of this. So, to analyze this we need to take help of random variable some probabilities stuff will come.

(Refer Slide Time: 20:51)



T(n) = runtime of Randomized QuickSort with n element

↳ random variable.

$$T(n) = \begin{cases} T(0)+T(n-1)+\theta(n) & \text{if } 0:n-1 \text{ split} \\ T(1)+T(n-2)+\theta(n) & \text{if } 1:n-2 \text{ split} \\ \vdots & \\ T(k)+T(n-k-1)+\theta(n) & \text{if } k:n-k-1 \text{ split.} \\ \vdots & \\ T(n-1)+T(0)+\theta(n) & \text{if } n-1:0 \text{ split} \end{cases}$$

So, what is the time complexity? So, we denote T n is the ran time of randomized quick sort with n element.

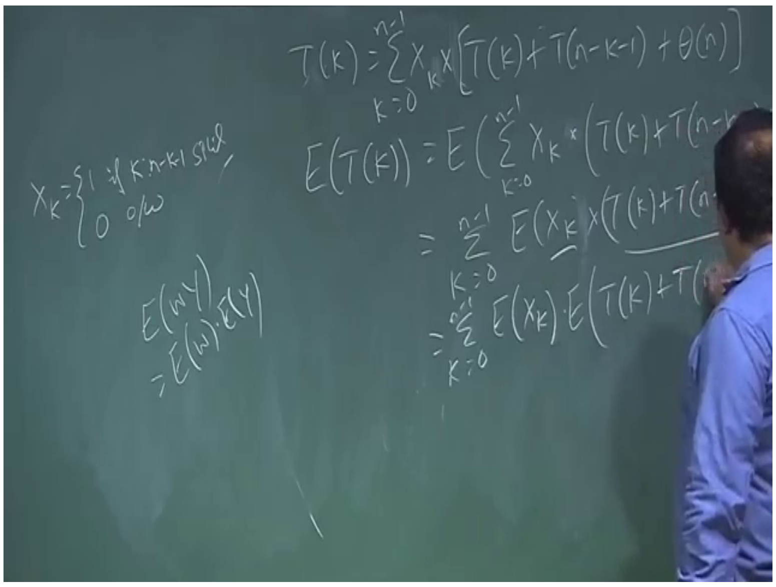
So, then T n is a random variable basically T n is a random variable because T n is will depend on the input pattern. So, that is why it is random variable. So, T n is basically what T n is basically can be written as the form which depends on the partition. So, if the partition is 0 is to n minus 1 if this is the split then T n is T 0 plus T n minus 1 plus theta of n if this is the split otherwise if T 1 plus y n minus 2 plus theta of n if the split is n minus 2 dot, dot, dot and dot, dot, dot.

If the split is change to n minus k minus 1 split then it is basically T k plus T n minus k minus 1 plus theta of n dot, dot, dot and this will be up to this n minus 1 plus theta of 0 plus theta of n if the split is n minus 1 is to 0 split. So, T n is in this functional form. So, we do not know which partition will occur we do not know which split will occur, but we know one of the split will occur any one of this. So, if it if the random number is bad then the split will be this if the random number is very good.

Then the split will be half half and the, depending upon the random number we are choosing depending on the pivot element we are choosing the partition will be one of this, but partition, but before running the code we do not know which one will going to occur. So, this is in this is the time complexity that is why it is a random variable this is in time complexity in the functional forms now we want to make it in algebraic form. So, that we can take the expectation, to make it in the algebraic form we need to take help of what is called indicator random variable.

So, what is that we will just defined x k is 1 if the split is k is to n minus k minus 1 if this is the split otherwise 0 0 otherwise. So, x k x k is the indicator random variable which is 1 if the split is this. So, 1 of the split will occur so; that means, 1 of the x k is 1 and then remaining x k is 0 and corresponding that expression will come so; that means, we can write this functional form in algebraic form like this in terms of this x k, T k.

(Refer Slide Time: 24:28)



$$T(k) = \sum_{k=0}^{n-1} X_k \times [T(k) + T(n-k-1) + \theta(n)]$$

$$X_k = \begin{cases} 1 & \text{if } k : n-k-1 \text{ split} \\ 0 & \text{o/w} \end{cases}$$

$$E(T(k)) = E\Big(\sum_{k=0}^{n-1} X_k * \big(T(k) + T(n-$$

$$= \sum_{k=0}^{n-1} E\big(X_k \times (T(k) + T(n$$

$$= \sum_{k=0}^{n-1} E(X_k) \cdot E\big(T(k) + T($$

$$E(WY) = E(W) \cdot E(Y)$$

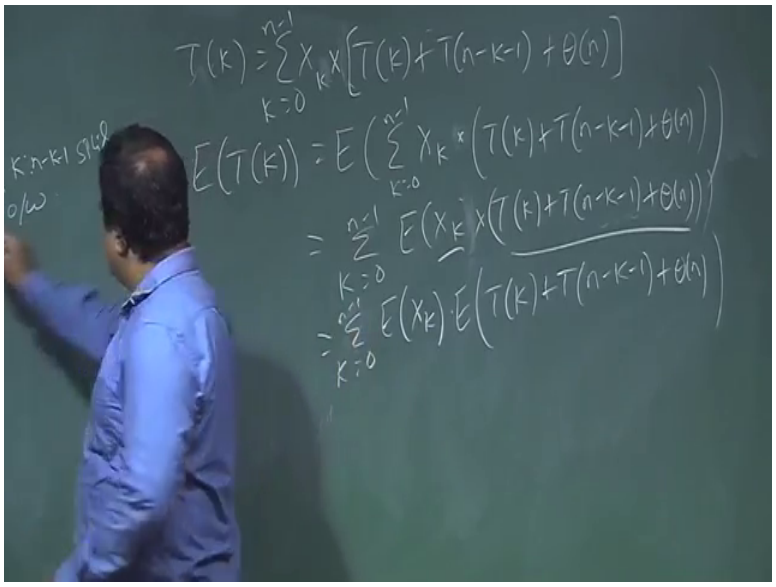
So, T k can be written as summation of x k into this is into T k plus T n minus k minus 1 plus theta of n this is x k and k is varying from 0 to n minus 1. So, this is the expression that functional form we write in algebraic form because we want to take the expectation we want to find the expectation now here we can say the.

So, now we take the expectation on both sides. So, this is basically expectation of this summation of x k into this is into T k plus T of n minus k minus 1 plus theta of n. So, this is basically k is 0 to n minus 1 now expectation is a linear function we can take the expectation inside. So, this is basically summation of x 0 to n minus 1 expectation of x k into this form T k plus T of n minus k minus 1 plus theta of n ,so, now, if we take this to be independent random variable this and this.

So, why from this independent rays are coming. So, we are choosing the random number for choosing the pivot element now every sub sequence step we have to choose. So, that choice of random number if they are independent then we can say this is independent. So, if we assume that independentness then this product can be written as. So, if 2 random number is independent expectation of w and say y if they are independent then we can write as this. So, this property is there in the probability theory.

So, that as a by that assumption we can write this is k is equal to 0 to n minus 1 expectation of x k into expectation of this theta of n.

(Refer Slide Time: 26:54)



$$T(k) = \sum_{k=0}^{n-1} X_k \times [T(k) + T(n-k-1) + \theta(n)]$$

$k : n-k-1$ split

o/w

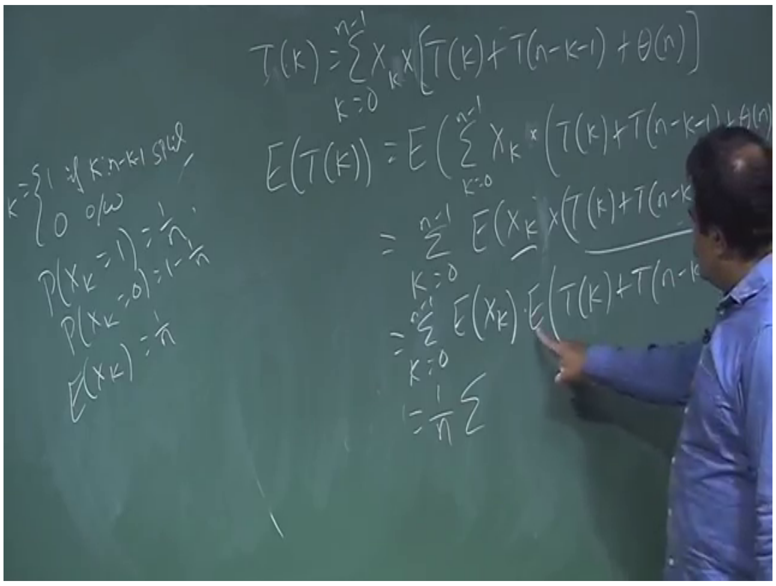
$$E(T(k)) = E\left(\sum_{k=0}^{n-1} X_k * (T(k) + T(n-k-1) + \theta(n))\right)$$

$$= \sum_{k=0}^{n-1} E(\underline{X_k} \times \underline{(T(k) + T(n-k-1) + \theta(n))})$$

$$= \sum_{k=0}^{n-1} E(X_k) \cdot E(T(k) + T(n-k-1) + \theta(n))$$

So, now this is basically expectation of this is a discrete random variable sop expectation of this. So, probability of x k equal to 1 is basically 1 by n because there are n possible split among this split one of they are equally likely.

(Refer Slide Time: 27:18)



$X_k = \begin{cases} 1 & \text{if } k : n-k-1 \text{ split} \\ 0 & \text{o/w} \end{cases}$

$P(X_k = 1) = \frac{1}{n}$

$P(X_k = 0) = 1 - \frac{1}{n}$

$E(X_k) = \frac{1}{n}$

$T(k) = \sum_{k=0}^{n-1} X_k \times [T(k) + T(n-k-1) + \theta(n)]$

$E(T(k)) = E\left(\sum_{k=0}^{n-1} X_k * (T(k) + T(n-k-1) + \theta(n)\right.$

$= \sum_{k=0}^{n-1} E(X_k \times (T(k) + T(n-k$

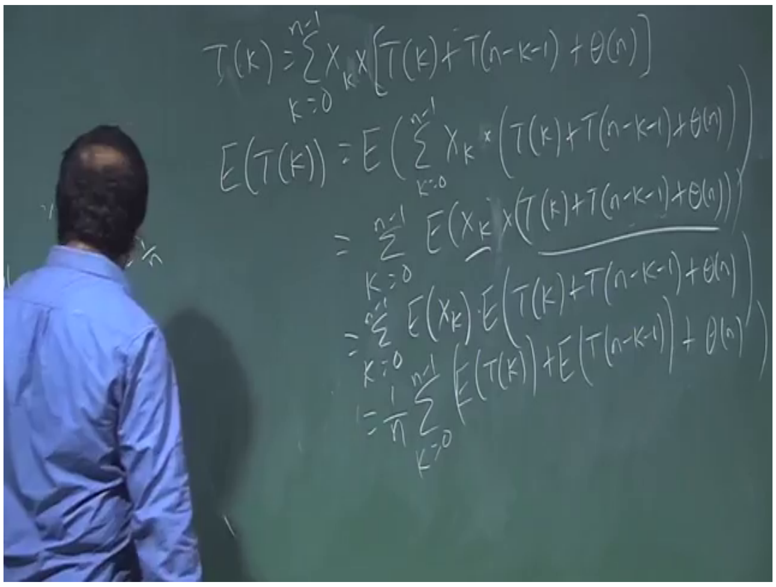
$= \sum_{k=0}^{n-1} E(X_k) \cdot E(T(k) + T(n-k$

$= \frac{1}{n} \sum$

So, this probability is 1 by n. So, the expectation and probability of x k equal to 0 is 1 by this. So, expectation of x k is basically 1 into 1 by n and 0 into this. So, it is basically 1 by n.

So, this will give us basically 1 by n will come our summation of expected value of this now this again we take the expectation inside

(Refer Slide Time: 27:33)



$T(k) = \sum_{k=0}^{n-1} X_k \times [T(k) + T(n-k-1) + \theta(n)]$

$E(T(k)) = E\left(\sum_{k=0}^{n-1} X_k * (T(k) + T(n-k-1) + \theta(n))\right)$

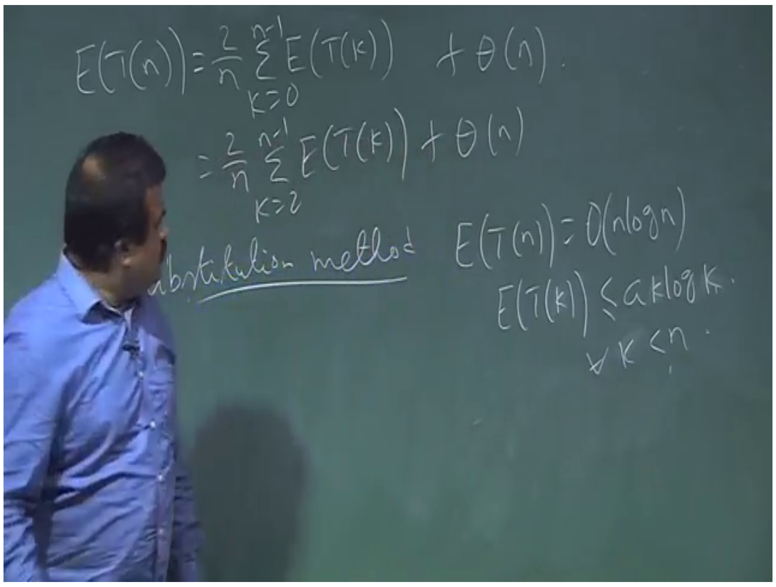
$= \sum_{k=0}^{n-1} E(\underline{X_k} \times \underline{(T(k) + T(n-k-1) + \theta(n))})$

$= \sum_{k=0}^{n-1} E(X_k) \cdot E(T(k) + T(n-k-1) + \theta(n))$

$= \frac{1}{n} \sum_{k=0}^{n-1} \left(E(T(k)) + E(T(n-k-1)) + \theta(n)\right)$

So, expectation of T k plus expectation of T of n minus k minus 1 plus expectation of theta n so, that is nothing to do with random n that is basically theta n, this is term k is equal to 0 to n minus 1. So, if we simplify this; this will give us basically what. So, let me rub this.

(Refer Slide Time: 28:17)

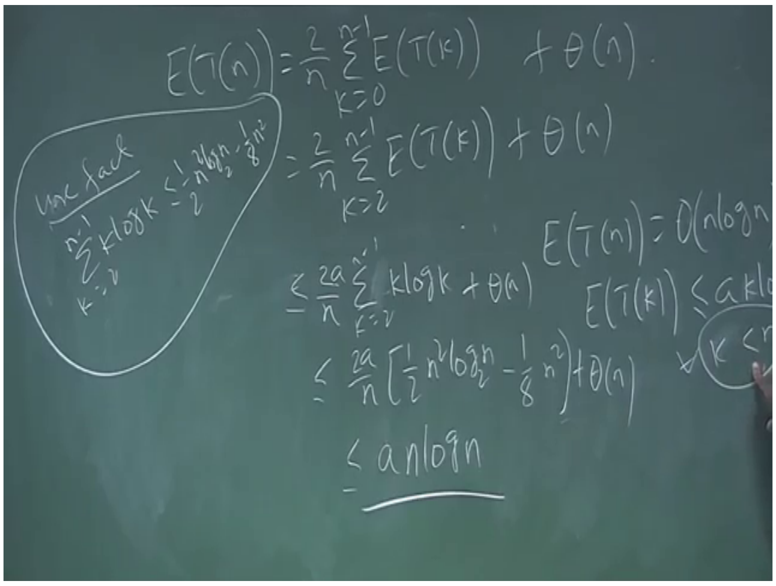

$E(T(n)) = \frac{2}{n}\sum_{k=0}^{n-1} E(T(k)) + \theta(n)$.
$= \frac{2}{n}\sum_{k=2}^{n-1} E(T(k)) + \theta(n)$
ubstitution method
$E(T(n)) = O(n \log n)$
$E(T(k)) \leq a k \log k$.
$\forall k < n$.

So, this will be give us expect we want to show the expectation of T n is basically l log n. So, that is our goal we want to show this expected value of T n is l log n theta of l log n. So, this is basically now this 2 term these 2 sum we can take the sum inside. So, this is from 0 to n minus 1 and this is for if we put n is equal to 0 it is n minus 1 to 0. So, 2 terms is coming out and this will combined with the theta of 1. So, this is basically 2 by n summation of expectation of T of k; k is equal to 0 to n minus 1 plus theta of n.

So, you got this now this first 2 term we can combine with this. So, this can be written as 2 by n summation of k is equal to 2 two n minus 1 expectation of T k theta of n why we take 2 to n minus 1 will use a result which is starting from 2 to n minus 1 and for 0 and 1 this term will be combined with theta of n. So, we got this now we want to show this is to be theta of n log n. So, for that we have to use some inequality.

So, that is we got this now we assume this is the substitution method we prove this using substitution method. So, what we are trying to prove we are trying to prove expectation of T n is big o of big theta of n log n. So, now, to prove this we are assuming. So, this is a big theta we are proving. So, we are assume this expectation of T n is less than equal to sum a l log n where a is a constant. So, this is our assumption. So, this is. So, this is true for k this we have to prove. So, k and k, this is true for all k less than equal to n.

(Refer Slide Time: 30:36)



$$E(T(n)) = \frac{2}{n}\sum_{k=0}^{n-1} E(T(k)) + \theta(n).$$

$$= \frac{2}{n}\sum_{k=2}^{n-1} E(T(k)) + \theta(n)$$

$$\leq \frac{2a}{n}\sum_{k=2}^{n-1} k\log k + \theta(n)$$

$$\leq \frac{2a}{n}\left[\frac{1}{2}n^2\log_2 n - \frac{1}{8}n^2\right] + \theta(n)$$

$$\leq a n\log n$$

use fact

$$\sum_{k=2}^{n-1} k\log k \leq \frac{1}{2}n^2\log_2 n - \frac{1}{8}n^2$$

$E(T(n)) = O(n\log n$

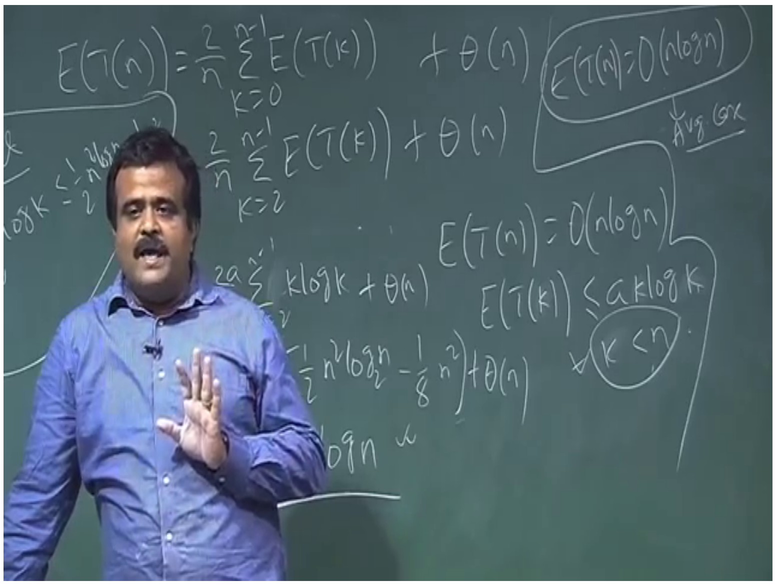
$E(T(k)) \leq a k l$

$\forall\, k <$

So, this we just put it here and this will be basically give us. So, this is all less than 2 by n and summation over A k log k plus theta of n. So, now, we will use a result use this fact what is this fact and the fact is summation of k log k k is from 2 to n minus 1 that us why you take 2 to n minus 1 is less than equal to 1 by 2 n square log n base 2 minus 1 by eight n square. So, this result we are going to use.

So, this result again we can prove by mathematical induction summation of k log k. So, if you take this a out a will be here. So, this is summation of k log k and this k is starting from 2 to n minus 1 this is basically less than equal to this is also we can prove by induction 2 by a n and this is basically half n square log n base 2 minus 1 by eight n square plus theta of n. So, now, we can choose this such that this will be basically less than equal to a. So, this is A n log n. So, this is true for n. So, we assume this is true for up to up to n minus 1 now we prove that this is true for n.

(Refer Slide Time: 32:16)


$E(T(n)) = \frac{2}{n}\sum_{k=0}^{n-1} E(T(k)) + \theta(n)$
$E(T(n)) = O(n\log n)$
Avg case
$\frac{2}{n}\sum_{k=2}^{n-1} E(T(k)) + \theta(n)$
$E(T(n)) = O(n\log n)$
$E(T(k)) \le a k\log k$
$\forall k < n$

So, this is true for all n. So, basically we prove that the randomized quick sort expected run time is n log n, but this is expected run time this is what is called average case analysis this is expected run time, but worst case runtime is always order of n square because in the worst case we always choose the pivot in such a way randomly we are choosing pivot. So, what case pivot will be always minimum or maximum? So, worst case run time is always order of n square whether it is randomized or normal, but if it is randomized version then expected run time is order of n log n.

So, this is the randomized version of the quick sort and quick sort is in place sorting algorithm for quick sort we are doing everything in the array we are for lagging merge sort we are taking extra array for doing the merge sub routine for quick sort we do not need the extra array. So, it is a in place sorting algorithm.

Thank you.