NPTEL
NPTEL ONLINE CERTIFICATION COURSE

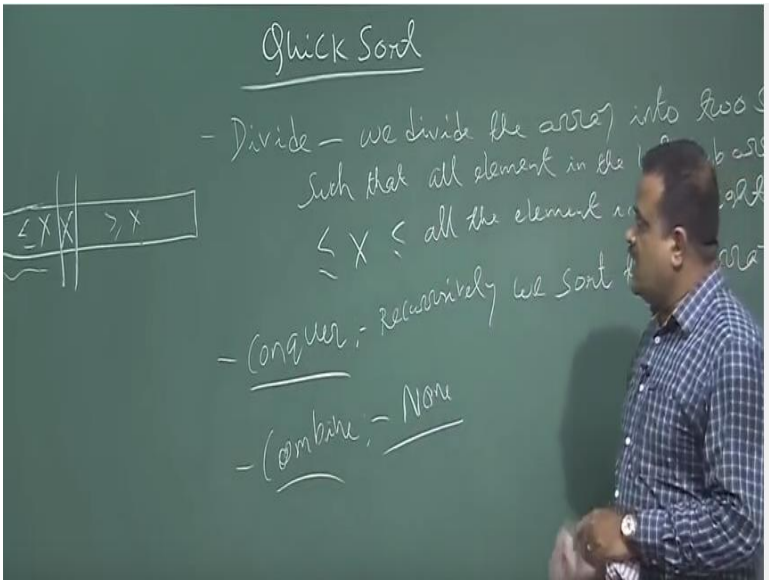
Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathemetics
IIT Kharagpur

Lecture 04: Quick Sort

Okay, so we talk about quick sort, it is another sorting algorithm, so it is a divide and conquer algorithm.

(Refer Slide Time: 00:30)



So we, in the divide step, so we have given the array of size f and we need to sort it. So what we do we divide the array into two sub arrays based on the pivot element. So we divide the array into two sub arrays such that the left sub array, all the elements in the left sub array are less than X and all the elements are in the right sub array are greater than X where X is a pivot element.

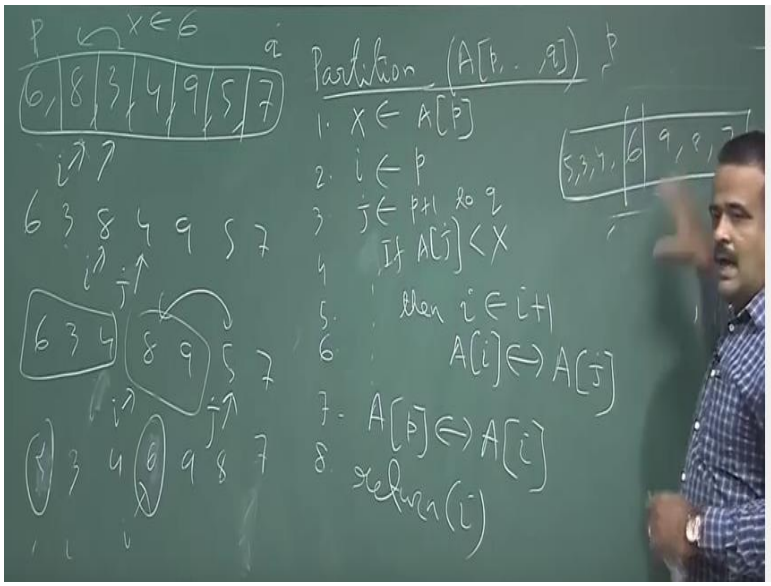
This is all the elements in the right sub array. So basically we are partitioning the array into two parts. So this is the pivot element, we can take any element of the array as a pivot element, but for our partition algorithm we will take first element as a pivot element. So we choose a pivot element and that is X, so now in the, in this divide step we will be using a partition algorithm, partition sub routine where it will partition the array into two parts.

In the left part all the element much be less than X and in the right part right sub array all the elements must be greater than X. So that is the divide step. So what is in the conquer step, so now we have two sub arrays. Now we sort this sub array, this sub array recursively. So recursively we sort two sub arrays, we sort the sub arrays.

So that is the conquer step, and then what is the combine step? So nothing to combine, I mean X is in correct position, so this is we are going to sort by recursively, this is we are going to sort by recursively. So no job is left, so none nothing, nothing to be done here. So done, so we want to have way, so this is the divide and conquer algorithm.

So in the divide step we need to have a partition algorithm which will partition the array into two parts. So let us just talk about the partition algorithm of quick sort. So let us just write the pseudo code for partition, okay.

(Refer Slide Time: 03:41)



So this is, suppose this is our array from P to Q okay, initially it is from n to n, but we will have to recursively call this again and again.

So we are taking the general term P to Q and we choose the first element of the array as a pivot element A[P] so we have array, this is A[P] to A[Q]. So we choose this as a pivot element so that is our X. Now we want to do what, we want to partition this array into two parts such that X will be sitting somewhere here and then this is all the elements will be here is less than X, all the elements here is greater than X.

So let us just try to write the code, so we choose i index as P, and we are having a loop j P+1 to q, and the loop invariant we want is, so we want, so this is our loop invariant. So i will be pointing somewhere here at some point of time, and j will be pointing here. So we want all the elements over here must be less than X, all the elements over here must be greater than X and this is here to decide.

So let us just write the code, so we choose I index as p, here we are having a loop j p+1 2q and the looping variant we want is, so we want, so this our looping variant so i, i will be pointing

somewhere here at some point of time and j will be pointing here, so we want all the elements over here must be less than x, all the elements over here must be greater than x and this is yet to decide. So initially everything is here to decide. Now we have looping j, so if this guy this is our A[j] if A[j] is greater than X, then we can simply increase j by 1, that will be taken care by this module, but the problem is, if this guy is less than X so that we have to fix, so if under this loop if A[j] is less than X then we have a problem. Then we cannot just increase j by 1, because that is not we are looking for.

So what we have to do, so what is the solution we cannot increase j by this 1, because this is some the guy over sitting here is less than X. So what is the solution, solution is we know the guy who is sitting here, this is the, this is i, this is i+1, I mean A[i+1]. So the guy over sitting here is > x so what we can do we can sort this and this and we can increase iy i+1 and j/1 that we are going to do here. So if this then we increase i by this and we exchange A[i] with A[j] and i is increasing here, now this was < than so this is coming here and j is peacefully can increase by 1.

Okay so this is the code so this is for loop, so after this loop we will just execute this, so now finally what will do we exchange Ø with A[i] so that i then we return the i, okay, so this is the code, this is the suitable code for partition algorithm for quick sort. We will take an example, suppose we have this 6, 8, 3, 4, 9, 5, 7, suppose this is our given array and this is p to q, okay, so now we want to execute this code on this given input so what we do we just, so what is our a, x, x is basically 6 which is basically first element of the array we are taking.

So we are starting from i=this j=this now we execute this now A[j] is not A[j] > than 6 we can peacefully increase j by 1, now here A[j] is < than 6 now we got a problem but we know that here we are sitting here is > than so we increase i by 1 and we exchange 3 and 8, so 3, 8, 4, 9, 5, 7 so now i is pointing here and j is pointing here, so now this is again < so what we do we just increase i by1 and we exchange this and this so 8, 3, 4, 8, 9, 5, 7, so now i is pointing here and j is pointing here.

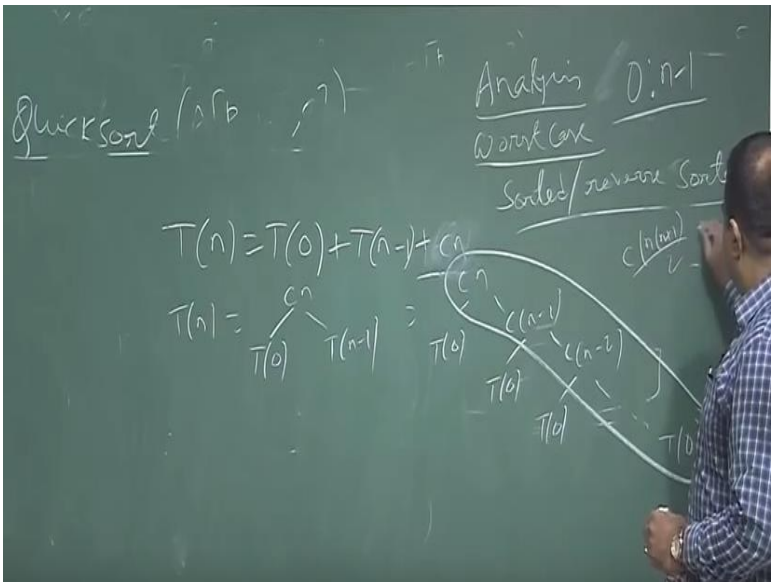
So this is > than 6 so no problem we can just increase j by 1, okay, so now, now we see everything over here are < than x and everything over here are > than x so that is the loop variant

void we want. Now we reach here, now we got a problem here so what we have to do we have to increase i/1 and you have to exchange this and this so 6, 3, 4, 5, 9, 8 7, now j is pointing here and i is pointing now this 9, i is pointing this 5, okay .

Now 7, 7 is > than so peacefully done so this our i so now we exchange this and this in this step and we put this pivot element here 6 and this is 5, and this is 1, 2, 3, 4 so we return 4, 4 is the position of the pivot element after this partition algorithm, so you see here after execution of this so we have 6 is our pivot everything is, here is < than 6 so all of elements are 5, 3, 4 and everything over here is 9, 8, 7 so that this is the divided step and in the conquer step we have to again recursively sort this array, this array, add, then done, nothing to combine.

So this is the partition of √10 so we have to write the pseudo code for quick sort then, so what is the time complexity for this code? This is basically a for loop of size n if the our array is n, if p, if p and q such a way that length is n then this is a linear time algorithm so this is the θ of n time algorithm. So now we have to write the pseudo code for the quick sort.

(Refer Slide Time: 11:23)



So this is A so we have to call this quick sort on this array. So what we do if p is < than q then only otherwise we will return if p=q there is only one element it is sorted we just return, so if p equal to, if p is < than q then what we do we call the partition algorithm, partition subroutine on this, okay, and then we call quick sort is the recursive call, quick sort so this will be so after partition X is sitting here and this is P to Q and this is the r, now we have to sort this, so quick sort A, P to R and again this is to sort the right sovereign A[r+1…..q]. So this is the c2 code for quick sort and the initial call is, initially our array is for 1 to n, so initial call is quick sort A[1…n], okay.

So this is the c2 code for quick sort, now we have to analyze this algorithm, analysis of quick sort. So okay then so what is the vast case for quick sort? Because in this version we are taking the first element as a period each time, so what is the washed case? If the array is sorted or resorted, if the array is sorted or reverse sorted in that case every time our pivot element is the maximum or minimum.

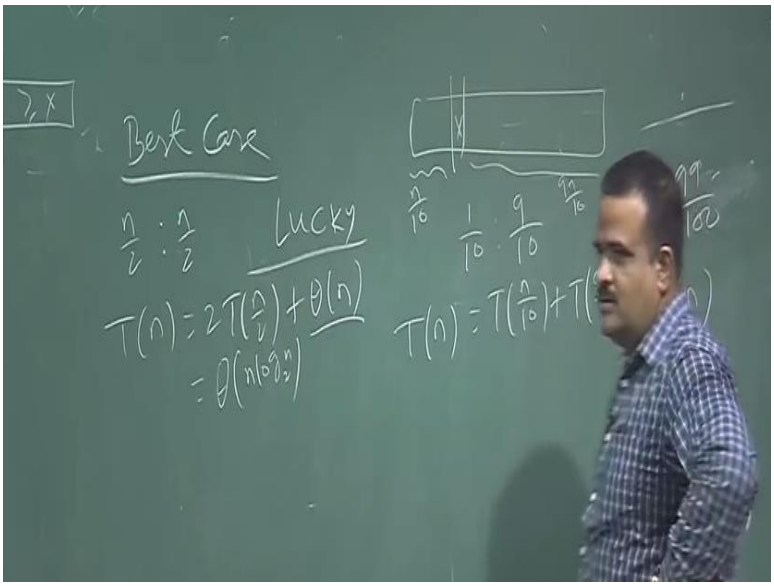
So that means it will after the partition it will divide the array into, if it is minimum, this all the elements over here or if it is maximum X will be sitting here all the elements over here. So it is dividing the array into 0:n-1, so which is very bad partition. So if that is the case then what is the time complexity, what is the recurrence? So the recurrence will be so 1, we have nothing in the one part of the array.

So $T(n) = T(0) + T(n-1) + \theta(n)$ this $\theta(n)$ is for the, this partition sub routine the first part of partition sub routine and the divide and the combine, nothing to combine but this is the cost for divide and combine, this is the basically the time complexity for the partition, okay. So this is the recurrence for this washed case when we have chosen the minimum or maximum element each time.

So what is the solution of this recurrence, how to solve it? We can use the recursive tree to solve it, so T(n) is basically $\theta(n)$ we can just a c(n). So c(n) T(0) then this is again T(n-1), so c(n) →T(0), c(n-1)→ T(0) c(n-2) like this T(0) like this so this will stop at T(g) be again, so this sum

itself it so the total time complexity is the sum of all terms. So this term itself is C(n(n+1)/2 this is basically order of n^2. This is the vast case time for the quick sort, okay.

(Refer Slide Time: 16:11)



Now when the, when is the best case? When is the best case for quick sort? So best case means, it is partitioning or partition is a good partition that means our period is good, it is partitioning the array into half, half. So n/2, n/2 in that case we are very much lucky, okay. So our pivot is very good pivot it is just the, it is choosing it is putting the middle like this, okay.

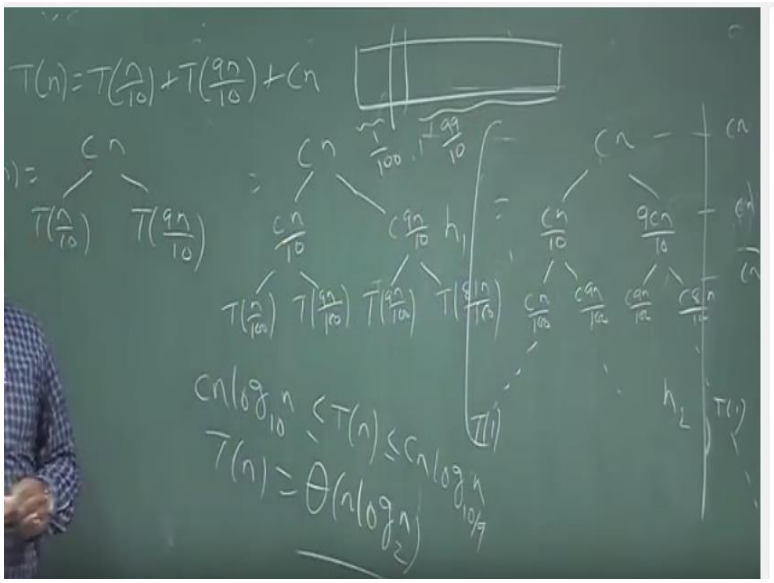
So in that case what is the time complexity, what is the recurrence? Recurrence is so we are dividing into two part half, half so again you need to sort this part and this part so two times $T(n/2) + \theta(n)$ this is the first part of partition, now what is the solution? This is again by master method this is the same recurrence as mark shot by master method we can, we know this is order of n log n.

So this is good I mean you are lucky if our partition is so good but it may happen that our partition is not that much good but it is partitioning like this, some fraction over here some fraction over here say (1/10) : (9/10) or may be (1/100) : (99/100) like this, so then we have to

see whether this is a good partition or bad partition, either you are lucky or unlucky. So suppose our partition, our partition is like this.

Then what is our recurrence? The recurrence is basically so it is dividing into two part this size is n/10 and this size is 9n/10, so T(n/10)+T(9n/10) + θ(n). So this is the recurrence for this, okay. So now how to solve this recurrence, so we will take help of recurrence tree to solve this.

(Refer Slide Time: 18:24)



So this is our recurrence Tn/10 + T9n/10 + Cn so Tn is basically Cn so Tn/10 T9n/10 so this again we will Cn so this again we will, so now this is our problem so recursively solving so this is now the size of the problem n/10 and again we are assuming the same partition will occur.

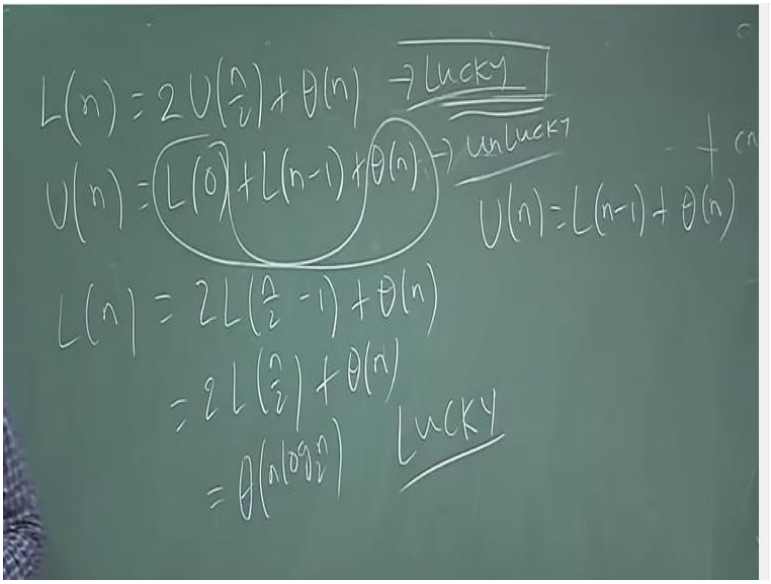
So again for this sub problem it will be 1/10: 9/10 so if you assume that then again it will be, so basically then we put n = n/10 over here so Cn/10 and here it will be, so we put n = n/10 so n/100 yeah T 9n /100 and here we put just n= so C9n/10 and here T 9n/100 and here it is T 81 n /100 like this, so maybe one more step Cn so Cn/10 9Cn/10 and here it is Cn/100 C9n /100 dot it will end at T1 because and so this T1 then it is sorted.

Like this, so this is basically C9n/100, c81n/100 it is also in do it T1 but which branch will end first this branch will this which branch will end first, so this branch will end first, this branch will go little long like this, so now what is what is time complexity for this how to yeah so this branch will end first because this is going little faster and this branch is go little more, okay, so if we just the total time is the sum of all with just some level wise, all level are in Cn like this so the time complexity so what is the.

If the height of this tree is left side is h1 and if this is h2 so h2 is little more than h1 so we can have this Cn, this is the height of that h1 Cn log of n, okay. Now we can replace this by 2 and then law 10_2 that will be taken care by this constant so this is basically n log n, so good, so we are lucky, so even the partition is not half, half, even the partition is some fraction over here some fraction over here this is 1/10, 9/10 even we can try with 1/100 99/100 some little fraction if we can guarantee then also the our partition is giving us.

Good result so it is a lucky case, okay, now we will see the case where we are alternatively lucky or unlucky and we want to see whether eventually we will be lucky or not, so that means we are this is divide and conquer approach, we are dividing the problem into sub problems so suppose first time we are lucky so an in the next step we will be unlucky.

(Refer Slide Time: 22:54)



So this is the lucky recurrence, so L(n) so now we are lucky so that means it will divide into but next step will be unlucky so this is the lucky step and for unlucky step. What we have suppose now we are unlucky so that means it will divide into 0:n minus 1 but next step will be lucky so this is the unlucky recurrence, so this is the situation, I mean we are lucky unlucky lucky or lucky this so we want to see eventually we will be lucky or unlucky, so we will with this is our equation one equation two so we will put this here so L(n) = 2, now this will be taken care by this part so this is basically unlucky recurrence is basically.

So L(n – 1) + Θ (n) so this is basically L(n – 2 – 1) + Θ (n) we are combining these two, now this is just in low or this is n/2 + Θn because this – 1 is just a silly, I mean if we comparing to integer part so this is basically give us the solution order of n log n so lucky day lucky, okay, so even if there are many unlucky stay in the middle but eventually we are lucky so if we can nC or that there is good number of lucky step then eventually we are lucky, so these idea will help us to build a part one of the quick sort which is called randomized quick sort.

(Refer Slide Time: 25:15)

Okay, so, so in the randomized so in our fraction of the quick sort we are choosing the pivot element as the first element of our partition algorithm, partition subroutine. Now we can choose any other element as a pivot like if we chose middle element as a pivot, okay so we always follow the middle element as a pivot.

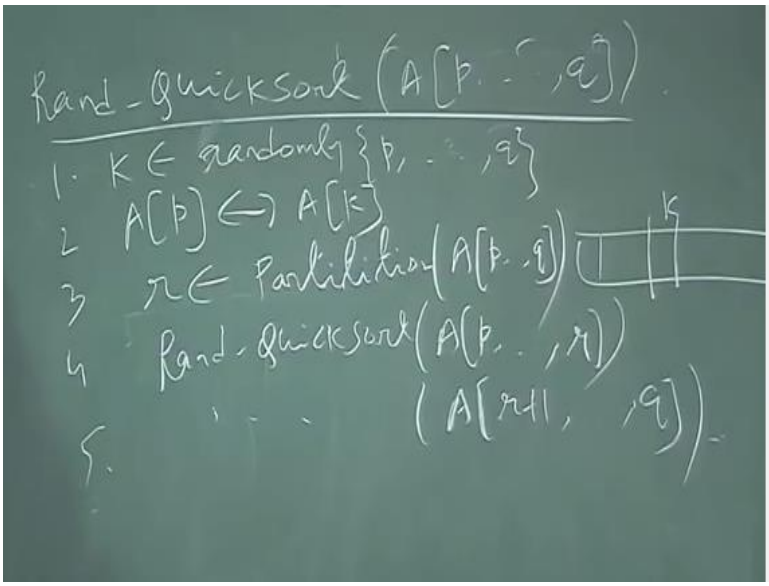
So in that case, so in our case first element as pivot problem if you take the, this, if you take the sorted or reverse sorted array, then it is always choose the minimum or maximum. But in this case also if you choose the middle element as a pivot alloys, then we can construct input where we can put the middle element as the minimum or maximum, so one can come with some input here every time we are putting the minimum or maximum here, so for here like this.

So in that case our algorithm will perform badly, because you are choosing every time minimum or maximum as a pivot element. If you choose pivot is the middle even though if you choose pivot as a second element then also we can construct input, here we put every time second element as minimum or maximum, so that way we can always construct, we can always get an generated input where our algorithm is performing badly. So to overcome these what we do we

chose the pivot element randomly, we will not declare before running the code which one will be our pivot element.

So that no one can come with some input where our algorithm will perform badly, so that is the idea. So we idea is to chose the pivot element randomly, select the pivot randomly from the array, so pivot will be, so we have array at here, P to Q, so pivot element will be any one of these array we can choose randomly, so this will be, this maybe the pivot, these maybe the pivot, so they are equal likely. So each element can be a pivot with probability if they are in element 1/n. So that is the fraction of the quick sort, so let me just quickly write the code.

(Refer Slide Time: 27:55)



So this is the randomized quick sort, okay. So now, so what we do we choose a random number from this, so we choose randomly an index from this and that is going to be our pivot. So this is suppose k, k which is randomly, okay now we, we know a fraction of the quick sort how our, we know the fraction of the partition where our first element has a pivot, so we want to, if you want to use that we can extend this A[P] and A[K], now, now we call our partition algorithm and then partition A,[p to k].

And then we call this quick sort, randomized quick sort again, randomized quick sort A, p to r and then randomized quick sort A,[r+1 to q] that is it, so only thing is we are choosing a pivot randomly since we want to use our own partition algorithm we are converting, we are extending this and this and then we are running our partition algorithm and then it is returning our, X will be sitting there, the X and then again recursively we are doing it.

So what is the time complexity of this code, how we can analyse this code, because this will depend on the partition, okay. Now if we denote T(n) is the runtime for this, of this randomized fraction of quick sort where array element is same, then T(n) will be written as in the functional form, this is the partition cost, if the, if the partition is 0:n-1, then if the partition is 1:n-2 it is like this, so in general if the partition is k:n-k-1, if the partition is k:n-k-1 so it is n-1+Θ(n), Θ(n) is for cost for the partition subroutine.

Now this is in functional form, now this so how to get these value, so we can do some, we can take help of some indicator algorithm, so this is what if the partition is k: n-k-1, 0 otherwise. And so the, and we are assuming the probability of each partition is equally likely, so each partition probability is 1/n, so probability, so this indicator algorithm, and so expectation of this is 1/n. Now we can write T(n) in this sum $\sum$ of because we cannot take the expectation just like this, X_K xT(K+Θ(n), okay.

So this is the expression, so we do not know which partition will occur, but we know only one partition will occur. So that is why we are taking help of this, only one X_K value is 1 other is 0. So now we, we are bringing this because of we want to take the expectation, now if you take the expectation, expectation will be this side so after some calculation we will, we can see this will be nlogn, so the expected runtime of this fraction of the quick sort is nlogn, okay.

So but we need some probability to prove this which is out of, which is not in the scope of our this course, so but in here so we must know that we can show this expected value of n is nlogn, thank you.