Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Module No. # 09

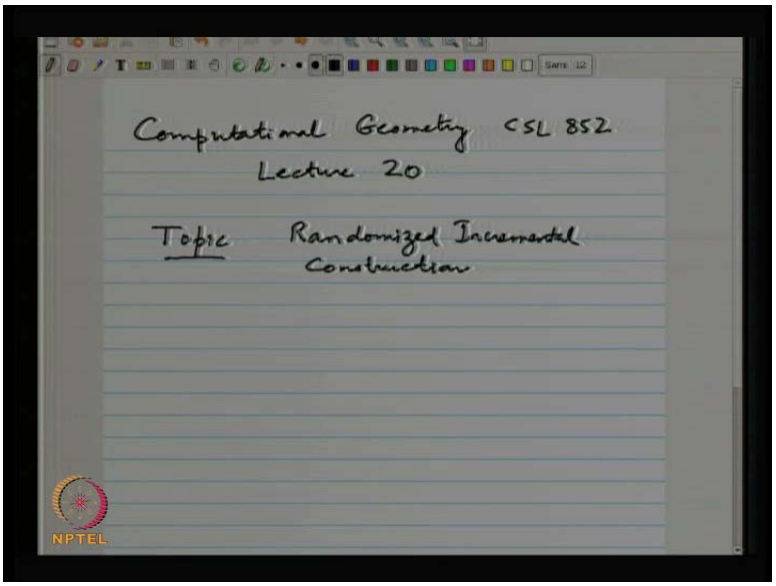
Randomized Incremental Construction and Random Sampling

Lecture No. # 03

Randomized Incremental Construction

Welcome to lecture 20. So far today we will lay the framework for a very important useful paradigm for the algorithm design, namely randomized incremental construction.

(Refer Slide Time: 00:41)

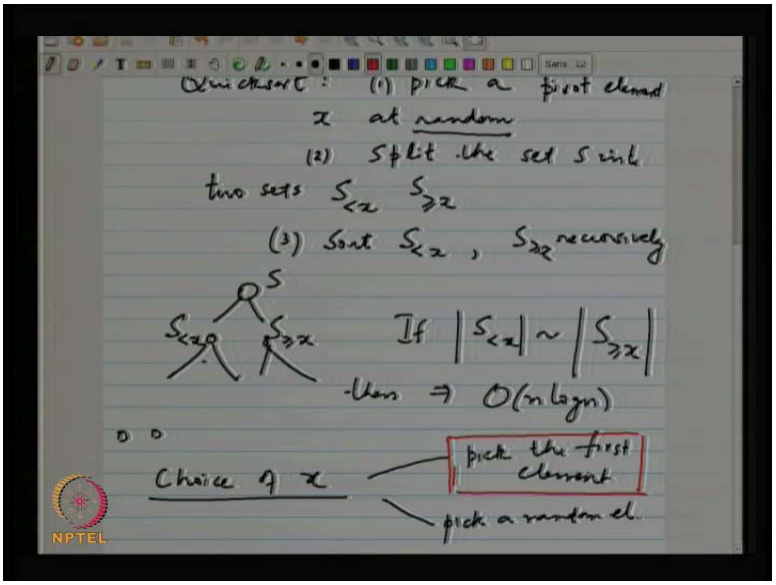


This method actually goes back you know to 30-40 years, even un perhaps can be thought as one of the first formal algorithm designed techniques. It is not known to be under this name that time, a subsequently over the years as you people discover more and more algorithms, that kind of can you fix this framework. It was a formally questioned as a randomized incremental construction.

So, the basic framework asset goes back whose way back to quick sort algorithm. Actually, if you think about the quick sort algorithm, one way to describe it would be

that we have given an element to sort. Thus, usually describe it is that pick a splitter of a pick pivot element that partitions the set into two sets, two subsets, and then you recursively sort the two subsets right.

(Refer Slide Time: 02:09)



So, quick sort can be thought of as pick a pivot element at random, and spilt a set s into two sets. So, let us call this pivot element x 2 sets s less than x and s greater than equal to x. Then sort and recursively is in the same, this is the same method that is again with each of the set pick the pivot, a pivot element and then continues.

So, this raise to a kind of a recursion tree uses a set s that depending on x. So, the subsets s less than x less than or to equal to x depend really on the choice of this pivot element. If you recall few courses in early course in algorithms, the efficiency of this algorithm really depends on how evenly this satisfies. So, this recursion tree and then subsequently, again you sort s strictly as an x tends to s greater than equal to x. In the same method that choosing pivots in this two sets and continue till your sets becomes the individual success, become very small like you know may be if you want you can go all the way till become symmetrically means, but you know in most implementations of quick sort the terminal once you have let us say some less than ten elements chosen.

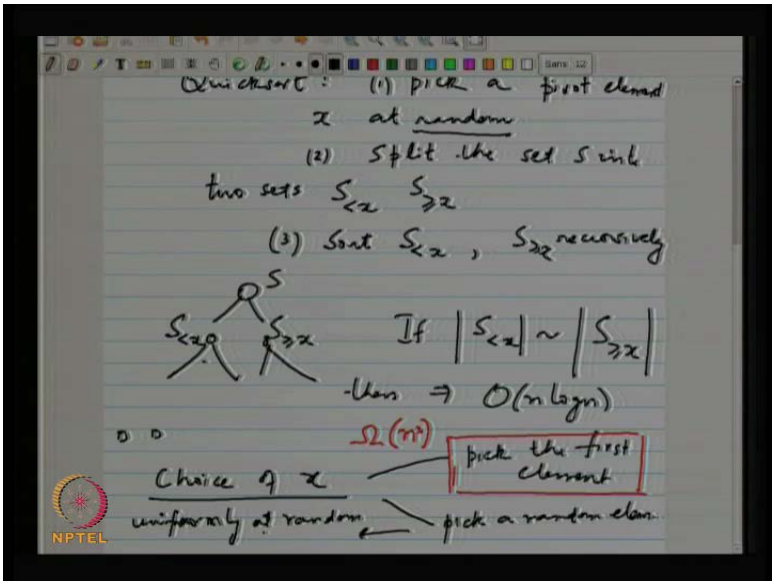
The success or the efficiency of the algorithm really depends on how evenly the pivot partitions are set. So, if s is these two sets the size of this and size of this are closely matched, then it implies something like an order n log n algorithm. Since, we are not

discussing quick sort for say, let me not go into you know why this is, so but once they are evenly split, you can consider that you are the depth of the recursion tree is not going to be more than log n. In each level of the tree you are doing about linear work.

So, this is the classical definition version of quick sort. In fact, the classical definition for it does not even required to pick this element at random but there is a fairly big difference in picking this x set random we saw we picking. So, choice of x, if possibilities one is pick the first element of s in whatever order it is given or like in this version. We have set pick a random element.

The difference between these two versions is that in text books, you will find that this is the version that is being analyzed. Then you can come up with a pretty bad permutation namely, that if you have your initial sequence of elements sorted in a reverse order. Then when you pick a pivot in using this method that is pick the first element then you are going to incur a huge cost. That is your first element is going to produce two very unequal subsets namely, one of just unit size. Other one is that is to the elements n minus 1 and you have to keep continuing this, because n minus 1 element. Then again when you pick a pivot, again you have picked the first elements that again create very unbalanced partitions. This way you can get something like an omega n square kind of performance.

(Refer Slide Time: 06:51)

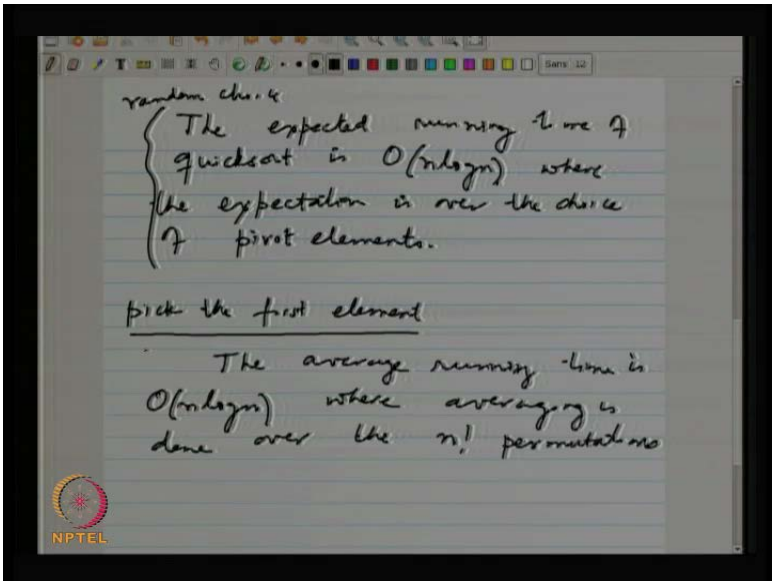


So, in many descriptions of quick sort, you hear this statement that quick sort has n square worst case complexity. What they are really saying is, if you pick the first element as the pivot, then for certain permutations, for certain worst case permutations, you will run into this, run into this pathological situation where your sets are very unevenly split. Now, as a post to this, if I pick random element, although it seems like draw the certain change very small change, you can make all the difference the following way.

Now, if you pick random element which is, let us say use random number generator and choose any of these elements with equal probability. So, all elements are picked, uniformly picked. So, any of this n elements can get picked and so even the first elements can get picked, but then the first element can get picked with probability 1 over n because all indices or all elements are likely to be picked as a pivot element.

They are all equiprobable and that is what makes the situation difficult to construct a bad permutation. Now, what is a bad permutation? The bad permutation completely depends on the choice of the pivots which is not known in advance. You are not; you will not be able to come up with this pathological worst case situation. That is you know take this permutation and then, you can force algorithm to partition it back. It really depend on the choice of the random element, and that is why the other characterization of quick sort performance which says that the expected running time, the expected running time of quick sort is order n log n where the expectation is over the choice of pivot elements.

(Refer Slide Time: 08:53)



So, this is a more accurate statement and this version of quick sort, it is very different. Let us say you know trying to create a bad permutation. In fact even if you keep the same permutation and run quick sort again, it may not behave the same way because every time you run quick sort, fewer elements are going to be chosen independently. Therefore, you cannot even force quick sort to have, let us say two consecutive worst case runs. So, that is really impossible.

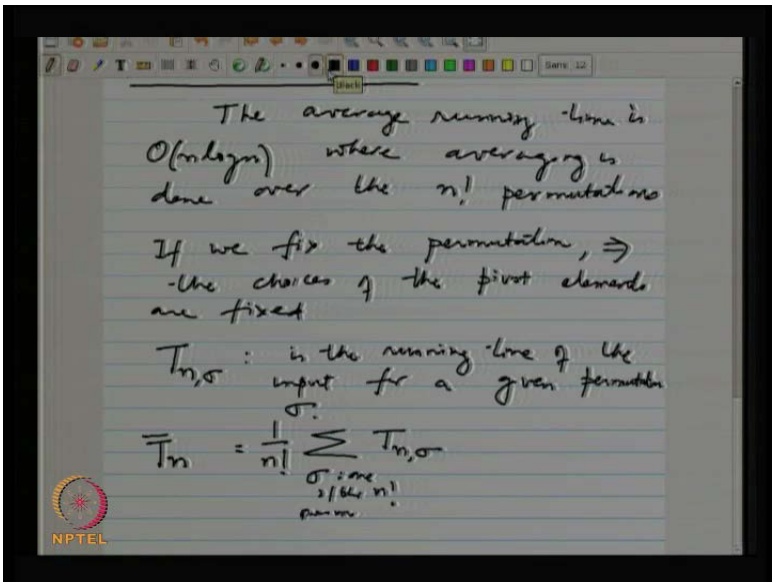
So, you can claim that this statement holds with probability almost one. That is very different from the case where you pick the first pivot element. First element is the pivot in which case the expectation of the average case is over the permutations. So, this is the over random choice. So, this random choice, where as if you first pick the first element, so in this version of the quick sort which this is actually the classical version that whole described.

Then, the average running time is order n log n where averaging is done over the n factorial permutations. n factorial permutations cause initially the n elements let us say all of them distinct the initial input can be permuted any other ways. If all this permutations are equally likely that is a very strong that is what the averaging is being done over. So, the averaging is done assuming that all these permutations are equally likely and then, that gives raise to this average running time. This is n log n.

However, as they set that there is no guarantee that this permutations are equally likely, you can actually get a permutation on which the quick sort will perform n square comparisons right. This is the version or this is basically the crux of the gel frame work of randomized incremental construction. So, what we are basically saying here is that if you fix the permutation whether the n factorial permutation of n elements, so if I fix the permutation, if we fix the permutation a, it basically means the choices of it implies there is choice of the pivot elements are fixed.

If we fix the choice pivot elements, then there is no variation running time because I know that this is the first element is the pivot. So, when I get the input, I know exactly how it is going to fix next time. I know the second element will be the pivot. So, it just behaves in a completely deterministic predictable manner, and once you fix the permutation, then the running time there is a variation right.

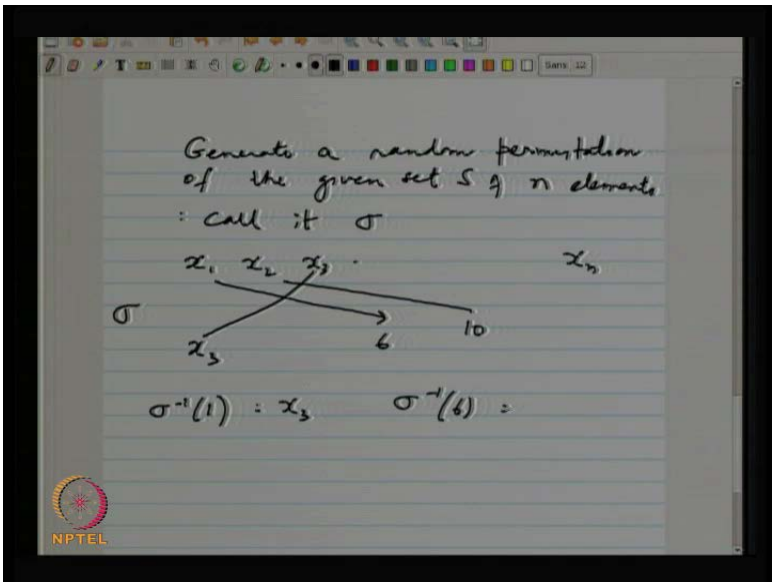
(Refer Slide Time: 11:57)



So, we can actually claim that the running time of quick sort given a permutation a. Let say pie. Let us call that let us denote that by t n. Let me call it sigma n to pie t n sigma is the running time of the input for a given permutation sigma. So, this is fixed. As I said once you fix the choice of the pivot element, this sequence as the pivot elements t n sigma is fixed. There is no variation. What this average case running does is simply says claims that this t bar which is an average case, an average running time of quick sort for sorting n elements. This is nothing but simply the averaging of that assuming all permutations equally likely.

So, I will sum up the running time over all permutations. So, sigma is one of the n factorial permutations. Of course, the averaging is done over all permutations. So, I should divide by one over n factorial. So, this is what is made by the average case analysis of the classical quick sort algorithm. So, given a permutation sigma t n, sigma is fixed. I look at all possible; I assume that all permutations are equally likely. So, I sum up the running times which are related average and the weights are actually equal because all permutations are equal. Lightly stated in other way, I can generalize this approach to other problems in the following way.

(Refer Slide Time: 15:23)

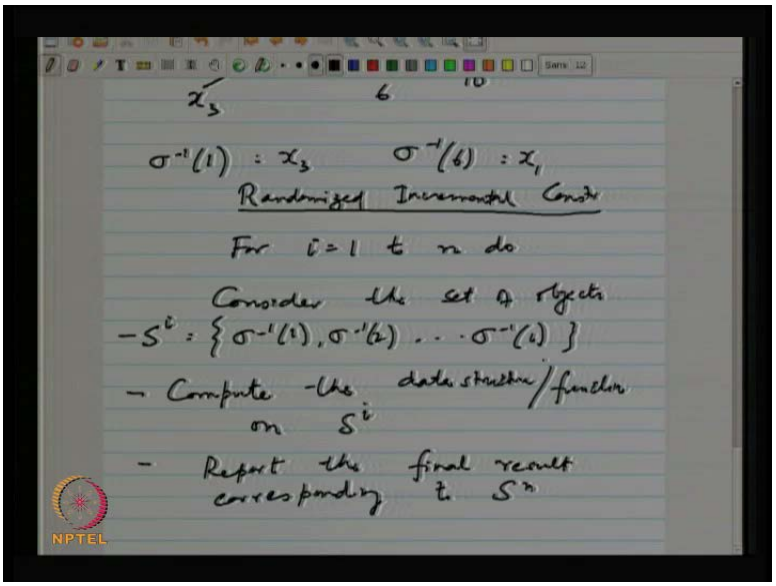


So, what we do is, initially generate a permutation. So, generate random permutation, the given set s of n elements call it sigma. So, initially my objects are given as a, let us say x 1, x 2, x n and arbitrary ordering. You know once you apply sigma to that. So, once you apply sigma to that x 1 is mapped to something. Maybe, it is mapped to say a x, may be x 2 is mapped. Sorry, it is the sixth position that is what I am s as we have seen.

So, x is mapped to the sixth position of x 2 is map to the tenth position and may be x 3 mapped to the first position and so on so forth. This is what is happening. So, this is my random permutation. So, I can refer to this element, the first element of the random permutation. Random permutation is nothing but sigma inverse of one. So, sigma inverse of one is basically x 3 sigma inverse of 6. The sixth position is equal to x 1 and so on so forth, so that when I want first object of random permutation, I say sigma inverse of one etcetera.

So, that is the random permutation we have generated. Now, this is very simple frame work where what you do is, you just add these or consider these elements in the ordered generated by the random permutation.

(Refer Slide Time: 17:17)



So, for i equal to 1 to n do add or let us say, consider the set of objects sigma inverse 1 sigma to the first i objects are random permutation. Let us call it s of i. Now, whatever function or whatever computation you are trying to do on this partial input, so compute it could be data structure, it could be a function. So, I am just saying that you know data structure oblique function on s of i. So, finally report the final result corresponding to s of n. You consider all the objects.

So, this is the general frame work of RIC. RIC is Randomized Incremental Construction because it is happening or it is in the context of a specific function or specific data structure. So, if it is something sorting like a quick sort, so in this type, compute the data structure, the function. Basically, we maintain a sorted subset of these elements. It may not be immediately clear. Why we are saying that? Even in quick sort, we maintain the sorted subset of i elements. Actually, if you think about not sorted the first i elements, but the pivot elements themselves are sorted.

So, i pivot elements, they are actually sorted among themselves and the remaining elements are yet to be sorted right. So, you have some initial elements x 1, x 2, x n at a point that where we have chosen the ith first ith pivots. So, I can mark those elements. Suppose, these are the elements that have been chosen as pivots, so you are maintaining a sorted set among these pivots. So, may be the sorted elements set this is the smallest

element. You know this is a second smallest, this is a third smallest and this is the fourth smallest.

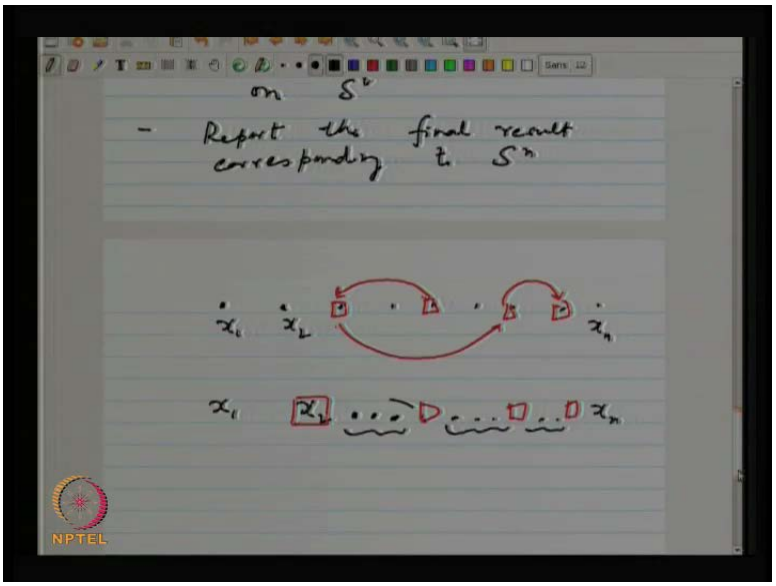
So, you are actually maintaining a sorted subset corresponding to the pivots. All the elements that appear between the pivots, they are not sorted. So, if I unwind and I pretend that my initial set I will consider the elements in sorted fashion. So, suppose these are elements, actually the sorted fashion after you picked the ith pivots. I can represent the same diagram in terms of you know so may be these are the four elements that have been chosen. Therefore, what remains to be done complete the sorting is, so these are not yet ordered.

You know elements appearing the pivots are not yet ordered and that is basically what we recursive calls two. So, you call the sorting recursively and this interval and this interval and so on so forth. Therefore, finally, the sorted set is just you know pasted together. So, that is precisely what we mean by when you say that the partial computation corresponding to si depends on the problem at hand. So, in quick sort, the partial ordering corresponds to the sorted ordering of the pivots and the elements between the pivots are not yet totally ordered.

The reason I am trying to explain everything in the context quick sort is the geometric problems are priory more difficult to visualize in terms of what happens when you insert the object random. This is simply enough problems. So, that I can take up and explain the frame work and do an abstract analysis without getting into the details of the geometric properties. Finally, when you analyze this thing just I said before, so tn sigma let us say corresponds to the running time of a specific permutation. These permutations are being generated at random. So, all permutations are equally rightly.

So, generate the random permutations. So, sigma is a random permutation which means that each of these n factorial equally likely. We are taking the expectation essentially over the o. Now, I can see expectation a over the random permutation right. So, we are actually a permuting the given set of objects into random permutation. In fact, when you do that we do not keep the permutation as such and I actually randomly permute and then, I pick the first element of the pivot. This behaves exactly identical to the situation where you give me a permutation and I pick any element at random to be the pivot.

(Refer Slide Time: 23:05)

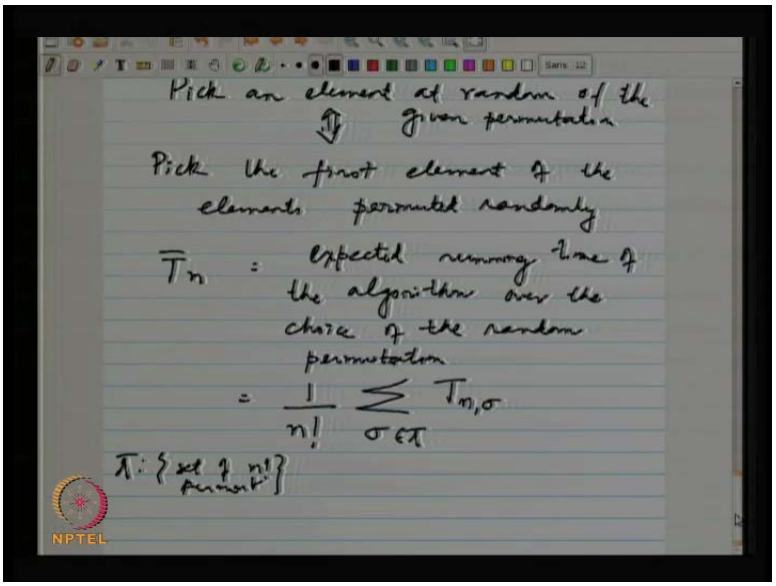


So, these two are any element at random to be the pivot. So, these two now become equivalent right. So, these two situations where pick a pivot at random. This now becomes the same or the behavior becomes the same as pick, the pick at element at random of a of the given permutation and pick the first element of the elements permuted randomly.

So, they become equivalent. So, what one case we are not changing anything to the given permutation, but then again I am chose the pivot at random where as the second situation, we pick the first element, but then before we pick the first element, we actually permute the elements of random. So, they become equivalent.

Therefore, the distinction that I drew between the classical version of quick sort and the random picking of the pivot, they now becomes the equivalent. Now, everything basically becomes I can capture everything by this analysis that the expected running time. Now, the expectation is over the random permutation.

(Refer Slide Time: 25:16)



So, t bar n is the expected running time of the algorithm for over the choice of the permutation. So, if the permutation is fixed, the running time is fixed. Is the same as one over n factorial sigma belongs to pi is the set of permutations set of n factorial permutations t n sigma right. So, this is what is going to capture running time of the algorithm based on RIC, I will see.

RIC is short for randomized incremental construction. So, here is basically where all the power is, where all the action happens. That you know some permutations like we discussed for quick sort are bad, but then again some permutations are good because you know if in those permutations where you know we end up picking more or less the middle element is a pivot, those permutations are good.

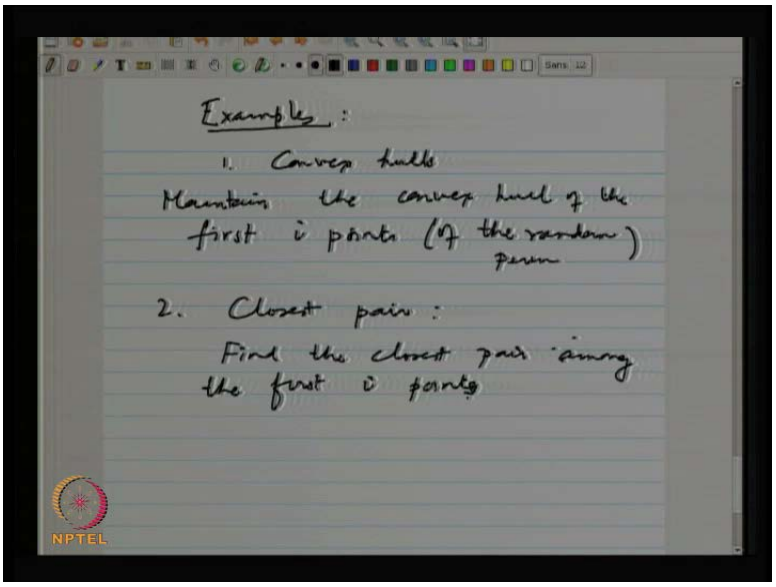
You know the question is over all choices of permutations. Are we really doing well or are we actually not doing well? Which way are we going? It turn out that you know because given the average case analysis quick sort most permutations actually happen to be good. Therefore, you can get n log n kind of expected behavior. Now, for a generic problem, it would certainly depend on the specific problem properly of the problem, but what is called amazing is that this frame works in the context geometric problems for many problems.

It actually leads to very good performance which are basically you know provably optimum. So, let me still expose this analysis with respect to quick sort, because I can

stay away from proving certain geometric properties which may just be a little distracting from the actual crooks of the analysis. So, let me try to analyze a quick sort in a way that can be generalized to other problems is under this frame work randomized incremental construction. So, this can be reviewed as other ways of analyzing the quick sort contain very general way that will extend to geometric problems.

So, for instance I am just to mention a few examples. Geometric problems could be, let us say for examples. So, examples and these are actually done in practice, a convex hulls.

(Refer Slide Time: 28:28)



What do you do basically maintain a data structure. So, maintain the convex hull. So, what is required a data structure required or some kind of computation is required to maintain the convex hull of the first i points of the first i points; of course, this i points are being picked from the random permutation.

Then, another classical example which is been very successfully is, compute the closest pair of points. Basically, you know you find the closest pair from among the first i points and one of the things that we always try to do is when we go to the next step namely when we consider to the next point i i plus first point, we update the data structure that is where the efficiency lies in the data. I did not say much about the data structure, but when we are computing the data structure function s of i, we are what is implicitly here is that you bring some updates from s i minus 1 using information of s i minus 1 that is

namely when we added the i minus 1 object we maintained computed something some data structure or some function.

When we add the ith object, we do not again want to compute the scratch. We want to somehow update the structure. So, that object caused to try to keep it as low as possible and that is what leads to the efficiency of this approach. So, certainly we are not computing from scratch that is why we are incrementally modifying or updating with the data structure. So, in case of quick sort again, when we add, when we consider the next element.

So, you have this situation where they are, let us say 4 pivots. When you pick the first pivot, let me call it different elements. Suppose I pick this as the fifth pivot. So, the only thing that I have to update is see these intervals are not getting modified ok. These are not modified, but only this interval, the elements corresponding to the elements, this interval they have to be now refined into two partitions right; one to the left of this element and one to the right of this element. So, this is the extent of the update. The others are basically not affected and that is what we want to exploit or take an advantage of…

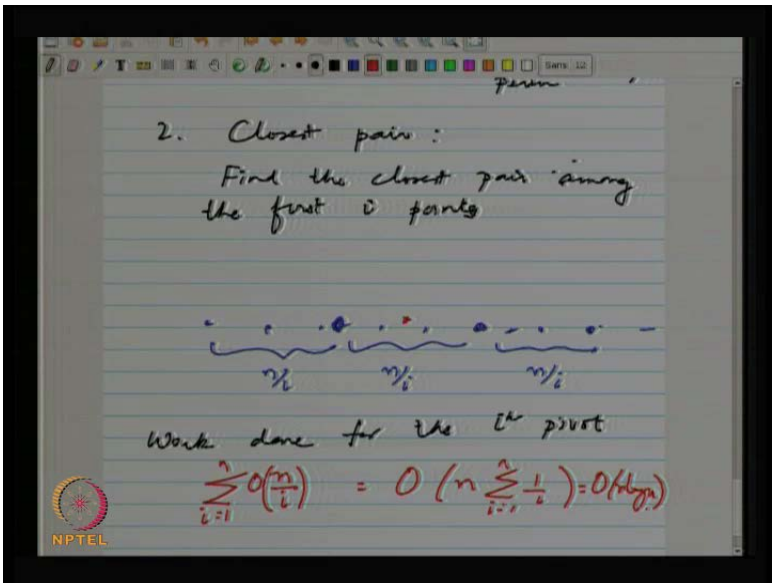
Then when we are sure to analyze this, it could become quite laborious. When I pick the next pivot which I am saying is the blue element, it could, of course it is coming from a random permutation. Any of the un-added objects could be the next pivot element. Now, how do we know where this object falls? Depending on where that objects fall, you will have to work with the elements in that partitions. So, if the partition is large, then the move elements need to be refined into this the two partitioned. All these elements are equally lightly to be chosen as a next pivot. It appears that this kind of analysis if we try to do from first principle, it is going to be very difficult because that probability, that element, the next element will lie in a larger pivot is much more because that contains more elements.

If you are choosing the next element at random from the analytic element which is what basically the random permutation, thus there is a higher chance that will pick elements from a larger partition. Therefore, one is inclined to believe that the expected cost which is basically the cost of partitioning, all the elements in all the interval is going to be quite large because this small probability there we pick the pivot from there. So, this is

actually somewhat misleading. This intuition is kind of misleading because why did this larger permutation occur. The larger permutation, then sorry the larger interval occurred because in the past elements that we added, they are not picked very informally.

Therefore, that leads tools of some variations of course, because you are picking the elements at random. There will be variation of the intervals, but then if they are more or less equal, then the work done when we pick up the next pivot element that is predictable. In fact, the best way that it can happen is, if that all elements some at the stage and picking the ith pivot. So, if all the sets are of equal size, namely everything has size about n over i point.

(Refer Slide Time: 33:58)



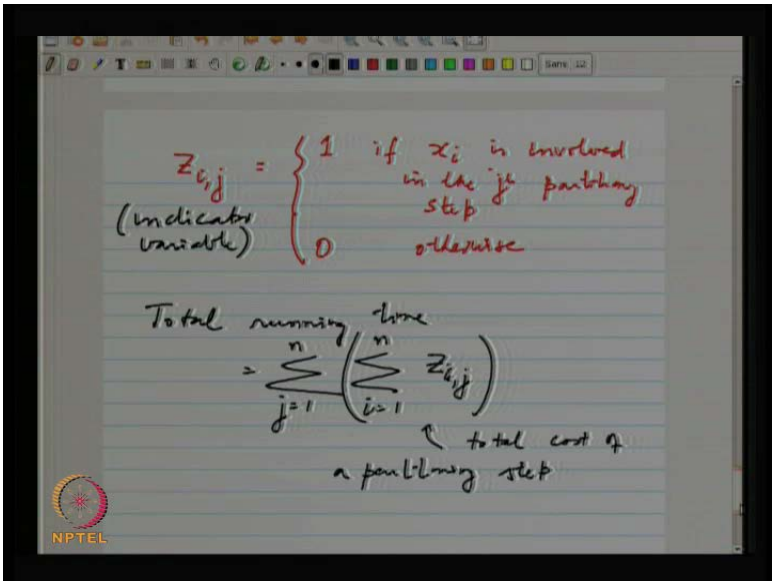
So, these are the pivot elements. Suppose, all these elements, all these intervals have equal size, then actually you can do a calculation like you know, so the amount of the work done. So, work done for the ith pivot is basically the ith pivot is with respect of where this falls into may be it falls here. We are dealing with exactly n over i elements, and only this n over i elements have to be repartitioned. Then, the cost is linear in the number of elements.

So, that cost is simply n over i and as you keep adding more and more elements, you basically have to sum this. Let me put some big 0 just make sure, it take care of small variations. This from i equal to 1 to n is nothing but big o of n times one over i, i is equal to 1 to n and that is a magical figure of n log n right. So, one would be tempted to believe

that this analysis is too simplistic, but it is little of nice right. I mean, it gives us what we are looking for. So, somehow we need to find the way to rationalize this calculation and that is precisely what is going to happen.

So, of course, I cannot assume. So, when we added the ith element, all the intervals were more or less equal, so that we cannot assume because it is a random process. So, we will have to somehow analyze this random process recursively. For doing that you know we will introduce some random variables. So, let us say a. So, we use something called actually an indicator variable. So, z i j is equal to 1.

(Refer Slide Time: 36:13)



If x I, now x 1, x 2, x n are the labels of the elements in the inter permutation. So, just to be a specific, we are using that labeling. It does not matter which labeling is used right, some unique way to identify the elements. So, all we are saying is that if xi is involved in the ith partitioning step, this variable is equal to 1 is involved in the ith. Sorry, not the ith which is a j. I am sorry, if xi involved in the j th partitioning step and to keep them to step and zero otherwise. Let us go back to this example, suppose I have all these x 1, x 2, x 3. You know some number x 10 may be x 11, may be this is x 50, x 51 etcetera ok.

Now, what I am saying that in the ith pivot step, this red element is the pivot of the ith step. This is the ith pivot and therefore, all the elements in the interval where these pivot elements belong to, namely the second interval here, all those elements are going to be repartitioned right. So, x 10, x 11, these are the elements that have to be actually

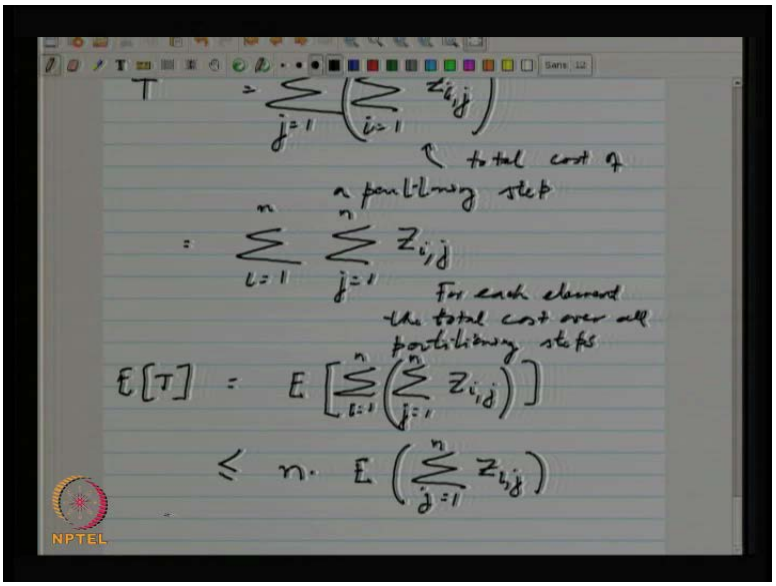
reallocated. If you are using an array, or if you maintain other kind of space, so all the work is basically happening there. x 1, x 2, x 3 or x 50, x 51, they are completely unaffected.

Therefore, we are not going to be paying to move those elements. So, whatever cost we are paying for algorithm in the ith pivot, state can be charged to the intervals. They are actually getting affected by this pivot. That is precisely what this variable is called in general called indicator variables indicator 0 1 variable. So, these elements add for the j th pivot. If x i is actually involved in the partition, then we are saying it is 1, otherwise it is 0.

Now, if you charge this way, then the total running time of the entire process, this random process that leads to the sorting. So, we can say the cost is basically put on the elements itself. So, if this element x 10 is getting repartitioned, we are putting some cost from that element itself. So, I can actually analyze or look at the running time by only looking at the cost charged to each element, and each element is likely to get involved in further partitions on down the recursive calls. So, the total cost, total running time is just another way of adding of things.

So, over all elements, over all partition in steps right. So, let us first look at the partitioning. Step j equal to 1 to n when i equal to 1 to 10 and z i j. So, inner some this corresponds to total cost of a partitioning step. Therefore, all the elements, some of the elements this z i j will be 0. Exactly, those elements that falls in the intervals where the partition elements falls in right and the remaining zero and then, I submit over all the partitioning steps.

(Refer Slide Time: 41:00)



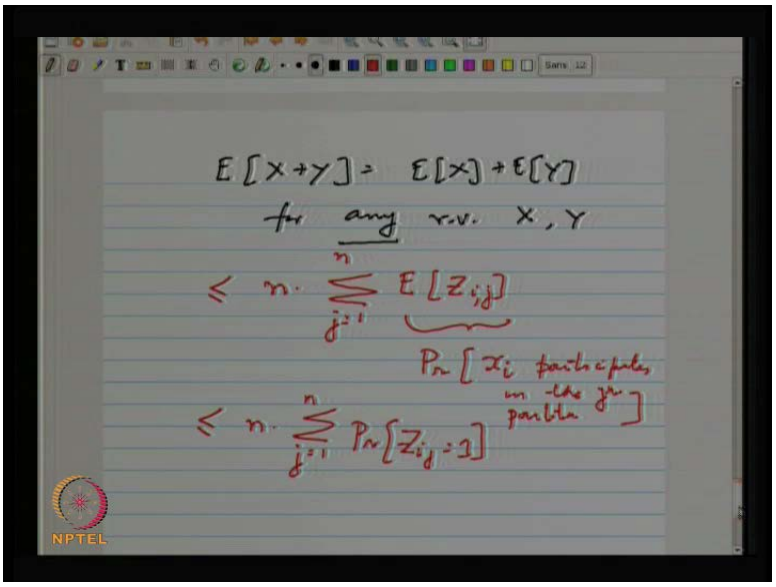
So, the outer some takes care of all the partitions. Now, I can rewrite this. Let us call this total running time as t. So, t can be also I can just interchange the sum. Actually, I can write it as j equal to i equal to 1 to n. This interchanges sum i j. So, now, I am summing for each element, the total cost over all permutations. Now these are all, sorry all partitioning steps partitions. Now, whether or not a certain z i j should be 0, one really depends on the random permutation that we are talking about.

So, once it is a random permutation, everything is fixed, but I have to do this analysis in a way that I am able to look at the expected cost over all random permutations. So, I will take the expectation on both sides. So, then t is nothing but e of this whole thing with summation. Now, we will claim that this is less than or equal to n times the expected cost of a single element. So, the expected cost is the worst case expectation for a single element. If I can analyze that and multiply that by n and that will be total expected cost right, so this I can rewrite as overall the pivot steps. What is the cost of a single element approved by the single element in the worst case? Since, there n elements if i multiply that by n subsequently upper bound the whole thing and now, I use on the top of this, the fact that expectations sum up.

So, linearity expectation and that is really the key of the analysis of the randomized construction. So, expectation, I mean this is a classical result e of x plus e of y. The

beauty of this does not depend on whether x and y are independent. Even for dependent variables this goes through right.

(Refer Slide Time: 43:33)



So far any random variables x y. Now, I am certainly talking about random variables starting from indicator variables. So, I am just renaming them and in the same variables, I am just calling them indicator random variables because these variables can be 0 or 1 depending on the specific random permutation that we have chosen. So, I am now using the specific subscript actually for all these things. I should have clearly written a sub states, because before I use the expectations for a given permutation sigma z i j will be 0 or 1.

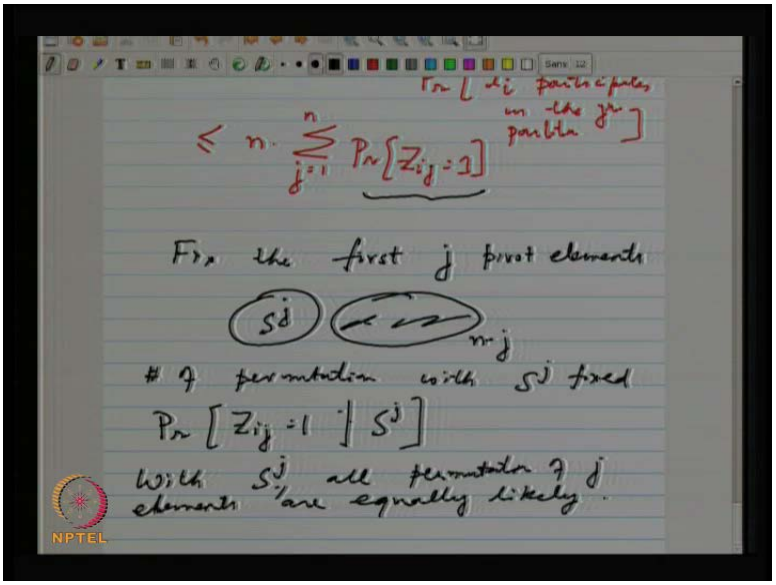
Now, I am beginning the expectation. So, I do not have to look at a fixed permutation, but it is an expectation for all permutation and that point essentially the z i j is become your random variables. So, this is the indicator random variables. So, now I can expand this, and write it as less than or equal to n times. It is the linearity of expectation. It is j equal to 1 through n expectation of z i j.

Now, what is expectation of z i j? z i j is 0 or 1 depending on whether or not the ith element is affected in the j th step or not. That is nothing. This quantity is nothing but the probability that x i participates in the j th partition. So, it is a 0 1 variable. So, the probability that something is 0 times 0 and 0s it is only to the other term probability that z i j is equal to one times the probability equal to 1 is exactly this one.

So, the whole thing comes down to analyze this quantity n times j equal to 1 through n. So, what is the probability that x i j again to that z i j equal to one. So, what is the probability? That is certain random variable will be affected with this, and this will give us a bound. How do we randomize because we really do not know how the previous it is completely depend on the choices of the previous pivots, so far this we use a beautiful technique for backward analysis again. We understood why it is called backward analysis, so far that we will view this way.

So, we want to eventually analyze this quantity. What is the probability that certain element will participate in the j th step? Now, we will use just nothing. So, we can only use the fact that these permutations are random. Therefore, all permutations are not only just random; they are picked uniformly of random. There is a certain symmetric property of these permutations. So, namely if you fix the first, so have taken the j th pivot. So, fix the first j pivot elements.

(Refer Slide Time: 47:12)

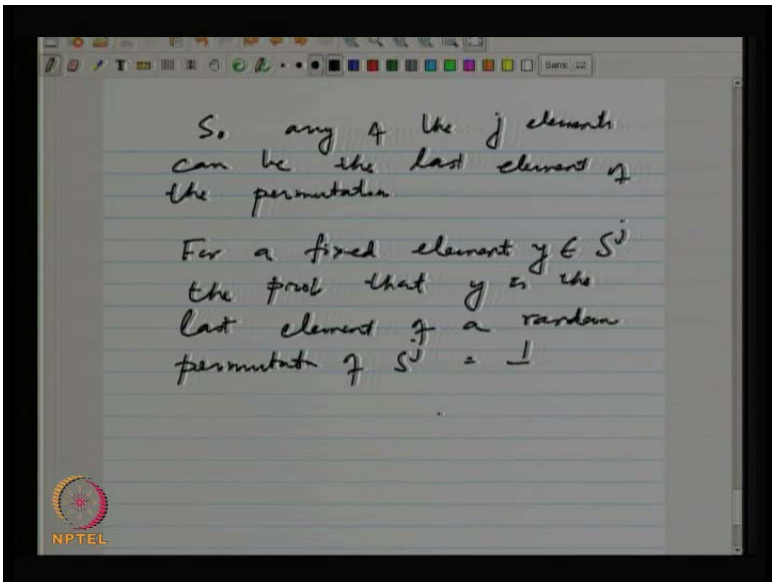


So, if you fix the j pivot elements, then I can only. So, there is of course, the first j elements let us call it s sub j. Then, the remaining a elements. So, we have fixed the first j elements of a random permutation. It is no longer that random and you have only choices of permuting the remaining n minus j elements that random. So, by fixing the first j elements, I can still permit the remaining n minus j elements but this is for a fixed choice of j elements.

Now, all elements all permutations are equally likely. So, all the number of permutations with s j fixed is basically what the way that it actually permits the j elements here times the number of whether you can perform n minus j elements here. So, we are not lost anything, but we are only this fixing the choice of the first j elements. We will do some kind of a condition calculation. So, by fixing the first j elements, we will find out this probability.

So, we will try to compute the probability that z i j equal to 1 given that we have fixed a set of the first j elements, because other elements are not really going to effect the probability, but I fix this. Then once I can do this conditional calculation, then we should try to uncondition that o. Now, again within s j elements, within the sj, all permutations are equally likely of j. Elements are equally likely which means that any of the j elements can be the last element of the permutation.

(Refer Slide Time: 49:50)

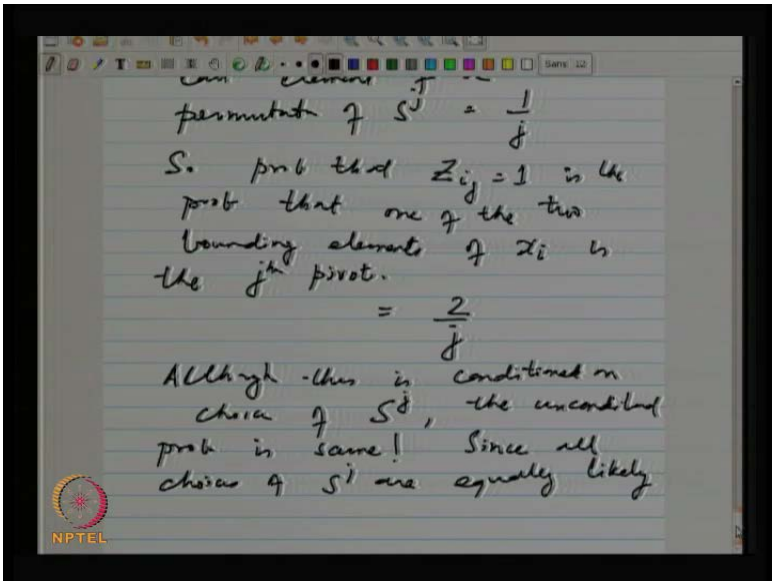


So, any of the j elements can be the last element between that if you fix an element. So, for a fixed element, let say y in sj the probability that y is the last element of random permutation of sj is one over j. Any of these elements can be last element. Now, we make a very critical observation. So, when is the ith element going to be affected? The ith element is going to be affected only when one of these red elements is the ith pivot chosen all the elements in the interval, that this one falls into getting affected for a fixed element.

So, for a fixed element like this when it is getting affected? It is getting affected in the ith step. If one of its two bounding elements changes the interval that contains its bounding element changes, then this element is going to be effected. You look at x 2. You know the bounding elements are not getting affected to be formal. We can introduce an artificial bounding element is minus infinity and plus infinity here.

So, unless the bounding element of the interval is getting affected by the ith insertion, that particular element will not participate in the partitioning process. So, how many elements? So far a fixed element, let us say it is 11 of any other fixed element. What is the probability that it gets affected? The probability it gets affected is the two bounding elements. One of the two bounding elements is the ith pivot right. So, probability that i j equal to 1 is the probability that one of the two bounding elements of x i is the j th pivot.

(Refer Slide Time: 52:16)

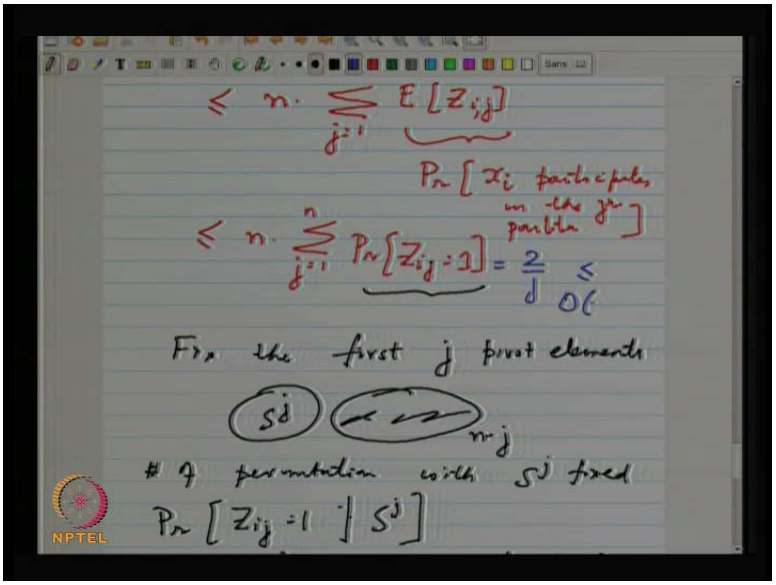


What is the probability? So, for specific element to be the j th pivot, it is 1 over j for two elements to be the j th pivot. So, that is basically equal to 2 over j. So, this is the probability that the ith element will participates in the j th pivot. Now, notice that all those things. This is conditional. The probability was conditioned on choice of sj. Some fixed sj. We did this calculation, but it only depends on j.

So, the conditional and the unconditional cost is the same. The unconditional cost is same and that follows the symmetry because all choices of sj are equally likely. Therefore, we just got what we wanted. It is 2 over j. So, for the jth step, it is 2 over j.

So, you go back to this calculation that we claim was too simplistic. Now, it is actually exactly the same thing. This is exactly the same thing, now to the probability. So, probability z i j equal to 1 less than or equal to, let us say equal to 2 over i.

(Refer Slide Time: 54:24)



Therefore, summation over all i simplex j th step. So, that is basically bounded by the harmonic which is 2 log n. So, for a single element, it is 2 log n. Therefore, for all the n elements, when you multiplied by n, so here is what we did basically n times this probability and this probability we computed to be equal to 2 over j. So, its whole thing is less than o. So, this is the randomized incremental construction analysis.

This kind of analysis goes through many other problems like convex hulls and to this sphere, except that we have to apply it in the context of a specific problem when you do this. So, we have to come up with this figure in the context of the problem. This is for quick sort, but then again all these calculations are quite close to each other. So, we just thought how you are going to actually do this analysis, but eventually this is what comes out of. So, we will continue with more applications in the next lecture.