Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

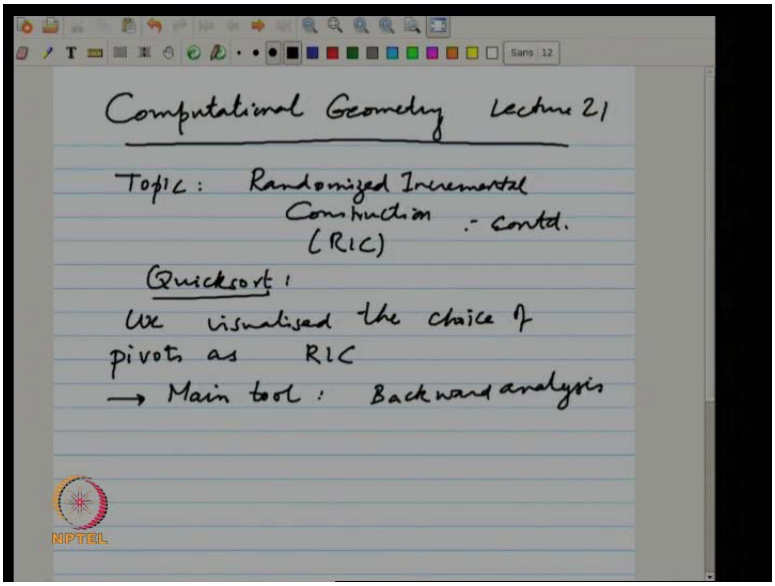
Module No. # 09

Randomized Incremental Construction and Random Sampling

Lecture No. # 04

Randomized Incremental Construction (Contd.)

(Refer Slide Time: 00:38)



As we last time, we just sort of illustrated the use of a illustrated with an example, the use of Randomized Incremental Construction and the example was quick sort. There are; of course, other known analysis of quick sort, but this is one of the more elegant one, perhaps. So, what we did was just to summarize, we showed; we visualized the choice of pivots as randomized incremental construction, RIC in short

(No audio from 01:16 to 01:31)

And we obtained, what is already known about quick sort; I mean we established the bound using this process and the main tool for analysis that we used was this notion of backward analysis.

Yes. How do you obtain the permutation? That was the question, I posted to the class. So, are you asking me, how do you actually obtain a Random permutation?

Yeah.

So, perhaps I should actually assign this as the problem, but I will give you some hint. So, of course, you know any method for random permutation has to randomized. There cannot be deterministic method, right. So, what you do is you need access to some kind of random number generator, which is available in all programming languages.

And, the idea is; I will give you the idea. So, the idea is that; well, I mean one is that you could; the obvious way to do it is that you start with some; you have this, let us say, you have an array of size n and then you choose any one of these and insert in another array. Of course, when you complete this process, I find a random assignment for the first element andt find a random second element and so on and so forth. Now, this rule produce a random permutation.

So, whenever you have to develop an algorithm for random permutation, you have to first prove that all permutations will be adequately likely to be generated. So, that is the first requirement; the second requirement is that; not requirement, second desirable thing is to have something efficient; and where is that this process may not be very efficient is when you are choosing a random placement for an object, it may conflict. What is the guarantee that you know, there will be unique number produced; in fact, you think about it; as you sort of. So, you are finding a random assignment in the first element and second element and as this array fills up, the chances collision or conflicts with an existing thing increases. So, this is also known as, I mean a variation of this is known as the coupon collectors problem

So, if you do the analysis, you will see that it will take about and also if you just do this process, take an element, find a random placement and then if there is a conflict in the random placement, you generate another random number for that object and towards the end, as I said, you will have to struggle more because the number of vacant places are reducing; means that lost n objects and if you are trying to place 1000 objects, you are going to really struggle to the find the last few placements.

So, if you do that analysis, you will see the expected sense, will do out n log n kind of calls. You get a random number to (( )) then I (( )) among the remaining x minus1. So, then, you have to have a random number generator with different properties. So, here I have assuming I have a random number generator which will give me a random number between1 through n, or whatever; some fixed range. Now, if you and there are ways where you actually you can reduce that range, so, next time I need a random number generated between 1 through n; now there is a complexity of this random number generator also because you are going to; what you have available is the random number, generated in a fixed rate. That has to be modified; as the range shrinks, that has to be modified and the cost for that also.

So, there are algorithms, where actually you take a random number generator and you can reduce the range; that is something called a dynamic random variate. So, there are algorithms for that also; that also has the cost because you do not get it for free. So, what is given to you is a random number generated between either 0 to1 and which we can scale appropriately, or 1 through n and that you have to sort of shrink as it goes on, 1 through n to1 to n minus1 and so on and so forth.

So, for all that this is the cost that you have to pay, algorithmically. So, the one that I mentioned is the most straightforward one; I do not change the range and you know I am going to generate numbers in that range; what you know there will be conflicts.

So, to make this more efficient than n log n; in the expected sense will take it n log n, because you know the number of trials, expected number of trials that you need to find a random placement basically increases as you fill more and more buckets. (It is) roughly about; the number of trials will be about, what? Can you guess?

So, the probability of success after you place the first one is n minus 1 over n, right; and then it reduces, successfully reduces; so, the expectation is 1 over that; so, when you submit up, it is just the harmonic again, n times harmonic. So, it is again n log n.

So, there is a trick that what you can do; instead of placing these elements between 1 through n, you take a slightly larger array, may be you take an array that is twice the size.

Take an array of size 2 times n and then you see that the probability of conflict is just bout at most half; that is even when you are placed all, let us say everything, but the last

element; there are still about n plus 1 vacant places, right. So, chances of conflict is no more than half.

And if you do the analysis properly, what you do; eventually you get a half filled array, right. You get all the n placements in an array of size 2 times n. And, then what you do, just shrink the array into, because the relative order is what the permutation is. Again you have to prove that it generates, say every permutation with the same probability of 1 over n factorial. And then, you can compress it; that is your permutation. And then, if you do the analysis correctly, here your success is always at least half.

So, expectation is order n. In fact, you can prove something even much better with high probability, it will be all right.

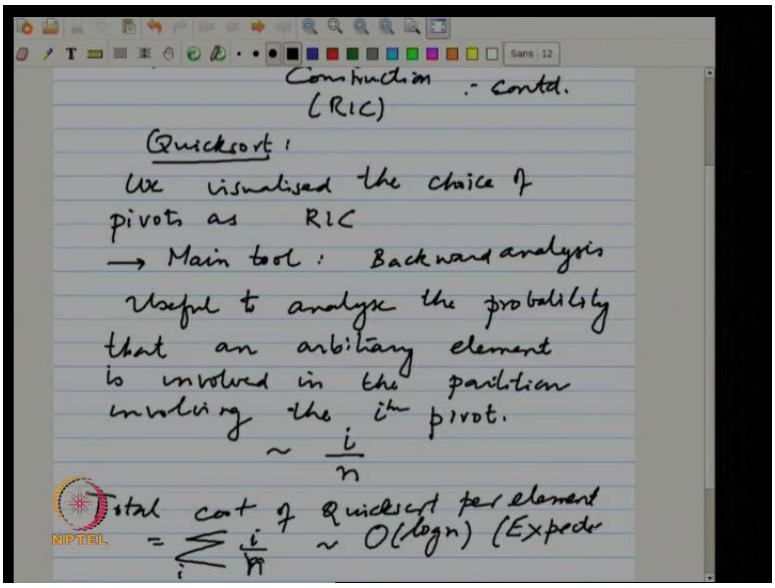
(()). Yeah. Eventually and finally, you just shrink the array.I just threw away the empty places; I just compress it.

I have told you everything, but still just to make sure that you never forget this or; you know this is a very useful thing, I mean it is used to left or right; some how, you know it is never taught in the course; some of people miss out.

(( )) Of course, yeah. So, you have to prove that; what you get is a random permutation.

So, when you coming back to this problem; the main tool used as backward analysis, which was following that when we are dealing with i x pivot, what is the probability that. So, I will just write it again.

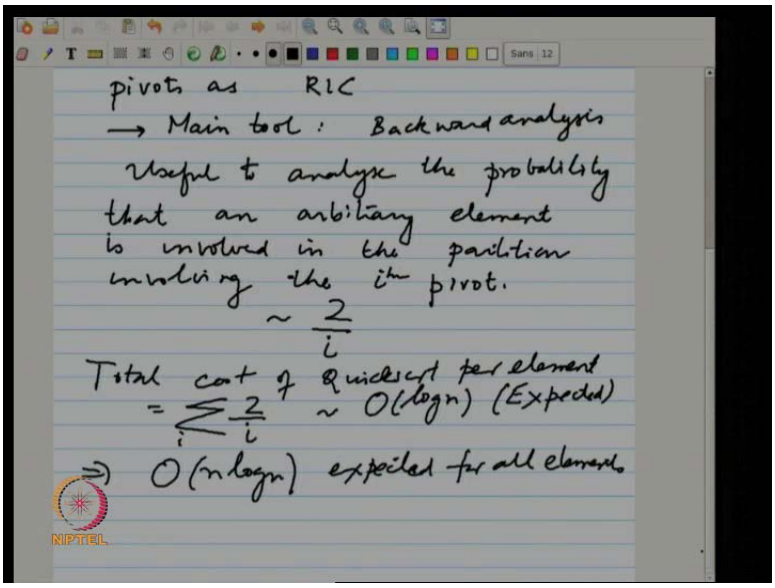
(Refer Slide Time: 08:50)



So, it is useful to analyze the probability that an arbitrary element involved in the partition. This is basically the probability that you could capture using this analysis that and this turned out to be a very nice a simple number which was about i over n. So, for an arbitrary element what is the probability that when we are dealing with the i th pivot, that element actually participates in that; that it get actually partitioned that pivot.

So, that is what we analyzed and with the backward analysis, we came up with the nice simple number of i over n and when you sum up, i over n from i, go to1 through n. So, the total cost of quick sort per element, summation over i, i over n, this is again log n; this is expected, of course.

(Refer Slide Time: 10:35)



So, for the n elements, it is n log n.

(( )). No, sorry. Thank you. 2.

Now, I just said this bound has been obtained by other means also; you know people have analyzed in different face and then able to get this n log n bound, but this is one of the cleanest analysis quick sort actually known to me. And, what the lessons that we want to take out of this is that in the very general process of Randomized Incremental Construction, when we add the next element, what is the total work that is being done?

So, whether it is sorting; in the case of sorting, the structure that we are getting out of it actually is a search tree. You think about it; the first element, it is basically the quick sort defines the search tree. The first pivot is a root of the search tree and then in other 2 partitions, again when you choose the pivots, they define basically the children of the root and so on and so forth. Finally, when you have completed this quick sort, what you get out of quick sort is basically a binary search tree. And although this.

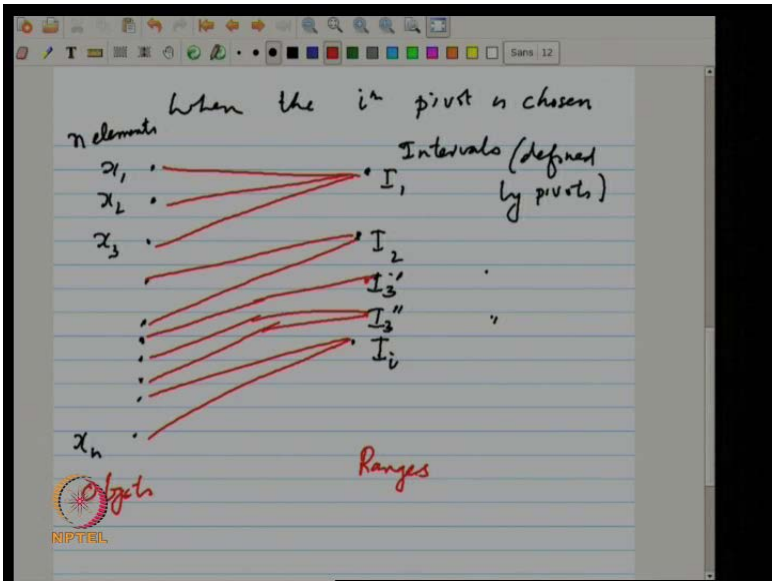
So, this analysis actually says that since no element is partitioned or in the expected sense, element participates or involve in the partition about log n times, the height of this binary tree, expected height of the binary tree is only log n.

So, it is a nice binary search tree actually. It is not one of those, you know long, skinny, skewed binary search trees; it is a nice binary search tree, where the height is about log n.

It follows from the expectation right that since no element is involved in more than about order log n partitions, it means that you take any path from the root to a leaf node, that path has a expected length order log n. In fact, one can prove (( )) stronger bound that with very high likelihood (( )); I am not getting into those technical details right now.

So, what you get out of this very simple process is also a any kind of quick sort process, not just this; quick sort also leads to a basically a search tree. But, for other problems, the other geometric problems, we will the structure may not be that simple. So, let me just point out few things. So, one is that what happens.

(Refer Slide Time: 13:32)



So, when the i th pivot is chosen, the scenario just before that is; we will view that as you know, a kind of this whole structure as a kind of a graph. Look at these elements, there are n elements and there are certain intervals. (No audio from 14:06 to 14:12) So, these intervals, initially you know you can to be. these intervals are being defined by the pivots. May be, these are the elements x1, x 2, x 3 all the way up to x n. Let me call these intervals, you know I 1, I 2, I 3. So, when the I th pivot is chosen, just before that how many intervals you have? You have chosen I minus1 pivots, so, that will define about I intervals, right. And the remaining elements; these elements are basically mapped to these intervals; certain elements belong to certain intervals.

So, it kind of defines some certain graph (No audio from 15:14 to 15:20) there is some kind of a graph, before we insert the I th pivot.

Now, when the I th pivot is inserted, what happens? One of these intervals splits; may be, this interval splits. So, I 3 is now; let me call it, you know I 3 prime and I 3 double prime. So, this interval kind of disappears. If you look at this graph, this graph gets modified when we choose this I th pivot. Some interval get partitioned, other intervals are untouched and the one that it gets partitioned; which is not a good example, so let me just put in a few more elements here.

So, 3 elements in this interval I 3, before the pivot was chosen, the I th pivot was chosen. Once the I th pivot was chosen, these 3 elements basically will get redistributed into these 2 things. right.

So, the graph now gets modified, where we have, let us say I 3 prime and I 3 double prime right and then these get redistributed. So, over the course of this; you know this, adding more and more pivots or the course of the Randomized Incremental Construction in the context of quick sort, you have essentially this changing graph where finally, it looks like each element belongs to one interval. That is the final process.

So, any intermediate process basically can be viewed or visualized as a bipartite graph; the bipartite based is not important; I am just saying that this is the bipartite graph because it express the relationship between the elements and the intervals.

The good thing or so what. So, how did you actually do the analysis? We did the analysis, assuming that the cost of the partitioning is proportional; the cost to the partitioning is proportional to the number of elements involved in that particular interval that is getting split; whatever was the size of the interval, because there redistributing exactly those elements. So, that is what we analyzed; that is the figure that we analyzed. We actually did not care of any kind of data structural cost; kind of assumed that in the algorithm, the algorithm behavior, the running time (of) the algorithm is actually proportional to the number of elements that are getting affected when the I th pivot was being inserted.

Now, for some other kinds of applications, who knows? Because, the cost may not be exactly the number of elements, but may be, you know 10 times the cost, may be log n times the cost, some such thing. But, here when we did the analysis, actually that number actually only captures the fact or that is true under the assumption that the cost of the I th partition the I th pivot is proportional to the total number of elements that are actually

involved in that. It is only the number that we are counting; we are not counting any other kinds of miscellaneous cost. Assuming that whatever algorithm we have, whether we are using a linked list or kind of array, you know we can actually implement that step, n time proportional to the number of elements that are getting involved.

If there is any more cost, you know with some associated data structure, that has not been accounted for. This analysis is true and you can actually kind of justify that yes we will be actually; even you know whether it is an array or whether it is a linked list that contains these elements; we will be actually able to implement that test , sorry implement the pivot, the I th pivot in time proportional to the number of elements involved

So, the generalization that we will looking for, when we talk about more complex problems; you know this is very simple 1 dimensional problem. If you view as a geometric problem, you know these elements are the points, points in a line.

Go to higher dimensions; talk about convex hulls; talk about Voronoi diagrams; talk about you know; what else you know, Triangulations, Delaunay Triangulations, whatever; even in high dimensions. You know these are no longer points; these are you know something like objects; it is basically a relationship between objects and what are these? What are the generalization of intervals? This object could anything; could be points, could be lines, could be segments, whatever, but what is the generalization of the intervals?

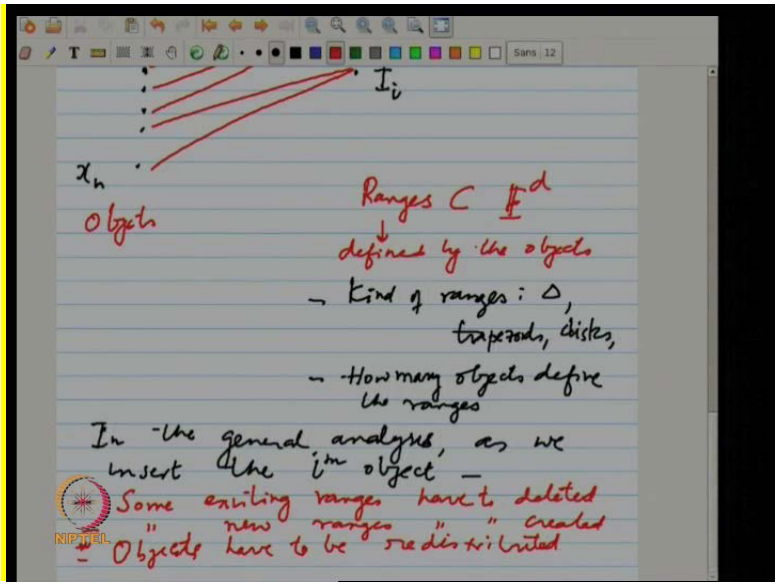
Well! It will depend on the problem itself and how we actually go and define these intervals? That itself will be crucial.

In the common terminology for these intervals; in the general case, they are called ranges. There we will something like a finite number of ranges; here we are dealing with finite number of elements and finite number of intervals; that was very easy to define or it is a natural thing to happen in the case of sorting. But, when we talk about geometric problems, we will have to struggle a little bit actually to define formally what the ranges are?

Ranges in this case are intervals; ranges in you know, for higher dimensional geometrical problems you know could be something like triangle or quadrilaterals or some such thing; some subset of the Euclidean space.

Ranges basically are subsets of Euclideans, d dimensional spaces, in the most general case. And so, Randomized Incremental construction will be maintain this kind of incidence informational, relationship between the objects and ranges, as we keep on inserting the object. Now, these ranges will actually be defined in some sense by the objects themselves. So, what those what how many objects define them?

(Refer Slide Time: 21:54)



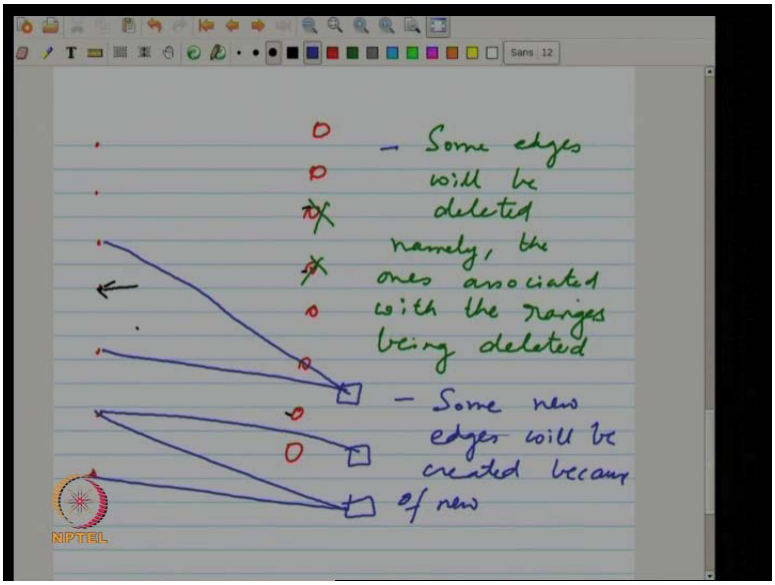
So, all these useful matter ; the kind of ranges as I said, you know could be triangles, trapezoids, disks; you know these are in 2 dimensions, in higher dimensions analogues of these things. So, the ranges you know, we will define them depending on the problem. So, one is a kind of ranges and also how do the objects define the ranges? Or, how many objects, let us say that is more important.

So, in the case of sorting, how many objects define an interval? 2, right. So, an interval is basically defined with the left end point and right end point. So, these are things that we will have to generalize, as we deal with more complex construction process, right. And, then finally, the analysis again will be done like this, you know you can think about the whole process like an evolving sequence of bipartite graphs. As we insert a new object, some old; as in the general analysis, as we insert the i th object, we will have to basically account for the following changes; one is that, some existing ranges have to be deleted, some of the old intervals disappear, some new ranges have to be created. And, when we do this, we have to reallocate or basically we have to change this edges basically; some

of the old edges will disappear, the relationship between the objects and the ones that are affected, the ranges that are affected by this; ones that are deleted, those objects will have to be now redistributed to the new ranges.

So, objects have to be redistributed. So, you can see the real cost of the i th step, comes from this step, third step. In the case of sorting, all those elements that you know, were affected by the partition; that was the cost associated for that i th pivot step. But, in the general case, you know the objects; which object should be redistributed? some of these ranges disappear.

(Refer Slide Time: 25:16)



So, suppose these were some of the existing ranges and these are the objects. Now, there is one more thing that can happen which we did not consider; in the case of points and intervals, a point you know was associated with a single interval.

Now, when we generalize, it may be case that a single object you know, it may not be a point; it may be a line or a segment, it can actually may be intersect more than one range. So, you will no longer have these simple, nice one is to one mapping, but you could have a situation like this. You know the same object can belong or can be associated with more than one range.

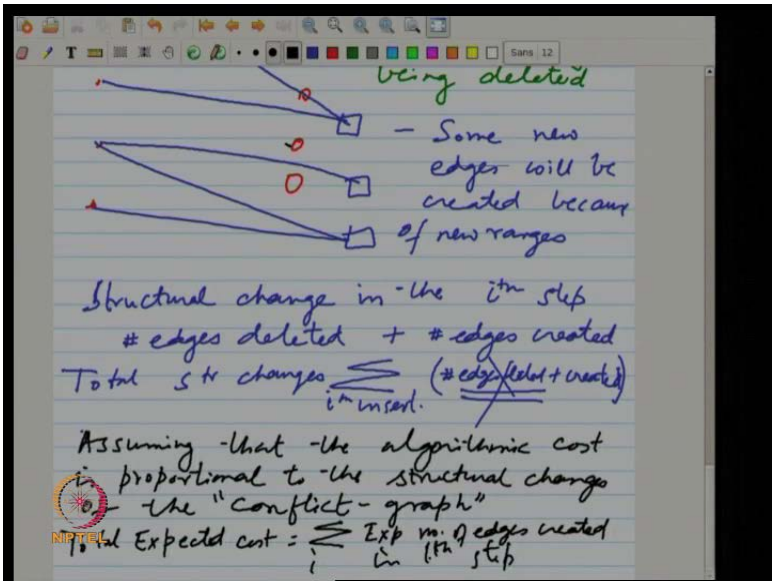
So, therefore, when the changes happen, when some of these things disappear; suppose this disappears and this disappears, you have to basically look at how many of these

edges are affected? So, all the edges incident on to the ranges that are going to be deleted; they are the affected edges. So, we will have to count total numbers of edges.

So, some edges will be deleted, namely the ones associated with the existing ranges; sorry that not existing, the ranges being deleted. So, those are going to be affected and then, there will be some new ranges that are created. So, may be some new ranges will be created. So, these will essentially will be reallocated or reassociated with some of the new ranges. And, so, these will disappear; so, those edges are affected and there are some new edges which may basically, the incident to the new ranges.

So, if you look at the structural changes in the graph, we will have to count for the edges that are deleted and the edges that are created. In the most simple case, we should count both of them; the edges deleted, the edges created, right.

(Refer Slide Time: 28:44)



So, the structural change in the i th step is number of edges deleted plus number of edges created; this is the i th step. And, then when we sum this up over all the; total structural changes, basically our all insertions, i th insertion and the same thing, the number of edges deleted plus created.

Now, here I claim that the first term that is number of edges deleted is something that I can forget about. Then edge can only be deleted once, right. So, we can charge it to when it was created and is not going to affect the overall analysis, right. An edge can only be

deleted once and if it is deleted, it must have been created at some point of time, so that cost of one is not going to make any changes; it just show up some constant factor. So, asymptotically let us say in the (( ))sense, we do not lose anything.

So, we can completely forget about the first term, number of edges created and you can claim to the total structural changes is the total number of edges created over all the I insertions of the Randomized Incremental Constructions process.

So, this the way we are going to analyze the overall cost of the Randomized Incremental Construction and of course, we are looking at the expected cost. So, we are going to look at the expected number of the edges created.

This is another thing that I mentioned before. Assuming that the algorithmic cost is proportional to the structural changes of the; this is called the conflict graph. So, once if you assume that there is no additional data structure cost, it is only the changes in the way the elements are associated with the ranges is what is the crooks of the analysis or basically that what determines the total running time. Then, we have that you know, total expected cost is nothing, but where all i steps, expected number of edges created in i th step.

So, this is the overall scheme of things for analyzing the Randomized Incremental construction and the Quick sort was a very special and a very simple example of that.

Now for this generalization, I will have to introduce some definitions and terminologies. So, I will go ahead and do that; and for that I will actually use some pre-prepared slides that I often use for giving talks on this topic; simultaneously with that I will use ; I will try to give some commentary and explanation. But, those figures are much better drawn than what I am going to do, right now.

(( )) Sir, in the quick sort case in the (( )) graph at the final stage. So, we had a very nice (( )) one to one elementary cost from the (( )) exactly. In the general case, is there always such a nice relation with (( )) or .

No, but I just mentioned that we are actually going to deal with the situations, where you know, the same element;

No, no. Yeah! At any juncture, you know the same object could be you know associated with more than one range. So, the generalization is the true generalization; we do not have any… We are not dealing with any special.

(( ))

Pardon.

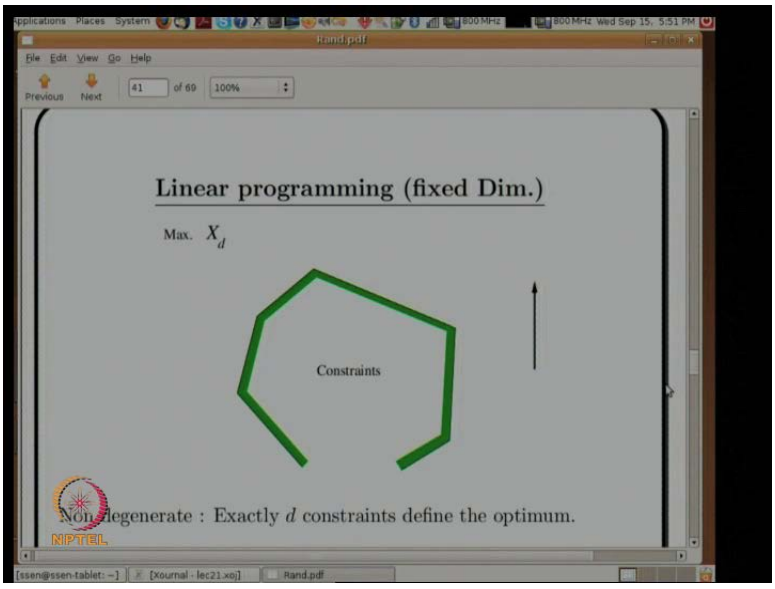
(( ))

Yes, it could happen that way, not necessarily, but it could happen that way. Actually, in the final stage, what happens is the following. So, as we go through the definitions, you know this kind of things will become clearer. Once, you have added all the objects then this notion of association, you know kind of becomes redundant. We have the final structure, so, the notion of association kind of becomes redundant at that point.

(No audio from 34:30 to 34:48)

Before I going to that, let me give another nice example of it; illustration of backward analysis. So, this slide diagram for 5 minutes, but I think it is worth it.

(Refer Slide Time: 35:03)



So, here is this problem, that is you know very well, right; and the Linear programming problem and you can view this; this is very much a geometric problem. Your constraints are basically inequalities or hyper-planes; hyper planes define the constraints with that is

the hyper sorry what you called it that half-spaces, the half-spaces basically are the constraints. And, what is being shown here is that you know it is just a two dimensional case of the linear programming. Where all the constraints are half-planes and this convex polygon is the feasible region.

And you can always assume that; I think we have discussed this briefly before; that you can assume that I do a rotation of axis, so that my objective function that I am trying to let us say, maximize is exactly the y axis; find in the top most point.

Some arbitrary; of course, a function, but you know I can just rotate the axis, so that you know I am trying to find the top most point of the convex polytope. And, I think I had mentioned this in the context that although the feasible region is the convex polygon or a convex polytope, the answer to the linear programming is only just 1 vertex of that polytope. I do not need that entire polytope; I need no need to construct the entire poly tope. In fact, it would be disastrous, if we try to construct the entire polytope, because we know that the size of the polytope can grow exponentially, if the number of constraints rise. So, we do not want to do that actually.

So, this is a very simple geometric view of this problem that; we are trying to maximize X d, you know the coordinates are X 1 X 2 up to X d.

And what we are going to do is basically, let us also assume that exactly d constraints define the optimum. In this case, 2 constraints define the optimum and this will follow, if you assume that no 3 lines intersect at the same point; no 3 lines are concordant; this is again the assumption we are making. Even if they are concordant, you can still handle it, but you know just then it becomes little more messy to do the analysis or explain.

So, we have the nice case where you know, only 2 lines intersect at a point and the optimum is essentially defined by 2 such constraints because it is a vertex of a convex polygon.

In d dimension, exactly d constraints define the top most vertex. So, finally, or you know even if someone told me that; here are the d constraints that define the optimum. can you verify it?

Suppose, I give this; I give you a set of constraints and tell you; here is the d constraints; so, actually define the optimum of the linear programming; can that be verified?

(( )) No. In this case, there are 2 lines that define the optimum and suppose, I just give you those 2 lines and say that you know, the intersection of these 2 lines is what the optimum is for the linear programming. Can that be verified, quickly?

(( )) (( )) (( ))

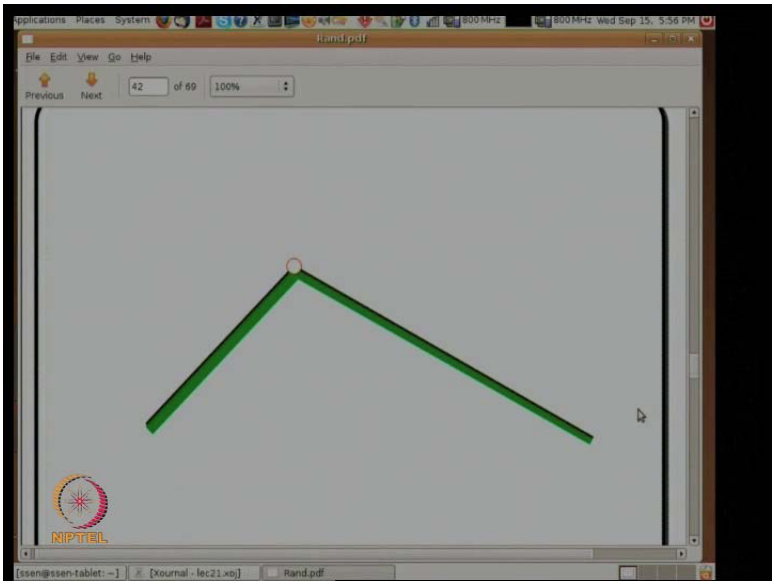
On 1 side of the from one side of the (( ))

That will amount to computing all the vertices; I do not want to do that; can I simply just verify? Well! It is not clear. But, one thing we can certainly do is that we can certainly check if that vertex is a feasible vertex, because I can simply check whether it is satisfies all the constraints or not; that is a quick check.

So, we will kind of we basically using that observation you know. We will do run this incremental construction kind of algorithm.

Let us assume that we add, we generate a random permutation of the constraints and pick up some, let us say in this case, 2 of these constraints; you know they will define an optimum and then you add the third constraint and the forth constraint and so on and so forth; and if necessary, we have to recompute the optimum. So, let us run this.

(Refer Slide Time: 40:12)



So, this is my first vertex that I computed from that 2 arbitrary constraints, you know.

the if you take point from the 2 lines a very it (( )) will be a local maxima (( )) and if it is a linear programming or a local maximum into (( ))

I am not sure about what you mean here.

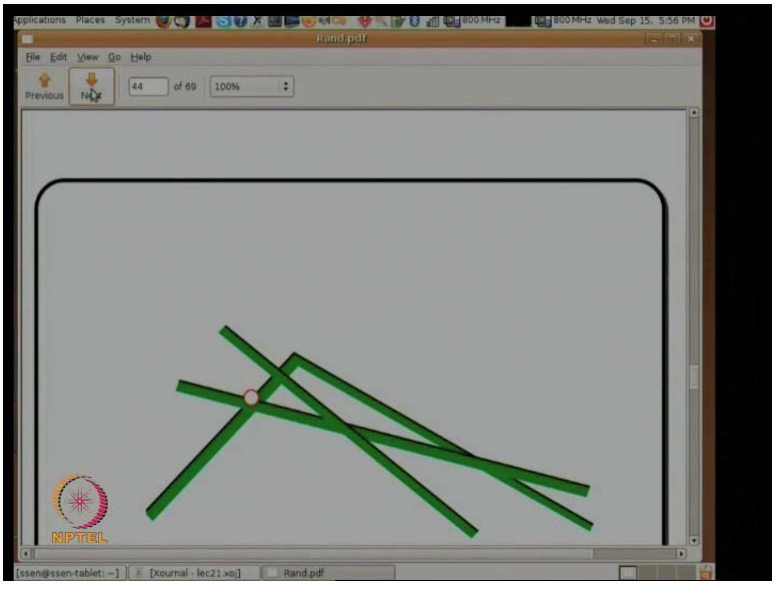
I mean at other corner points then neighboring points on those lines will be do not be local maxima and the other corner points

Which other coordinate points you are talking about?

and in the (( )) and the other corner points which are there those corner points will not be local maxima from its (( )).

I am not sure what you mean here. Just continue this. There is something about local and global optimum in the case of linear programming. Of course, that is what makes a problem work, the algorithm work efficiently, but I am not sure about what you mean here.

(Refer Slide Time: 41:31)



So, here is my first step; I have taken 2 arbitrary constraints, you know from the random permutation; these are my first 2 constraints, denoted this. Then I add the next constraint, consider the next constraint. Now, the next constraint you know somehow does not see , if this optimum satisfy the next constraint and so, like this one. So, the optimum that was

on the, the previous optimum if it satisfied the next optimum, then we could have continued with that optimum, right.