Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Module No. # 03

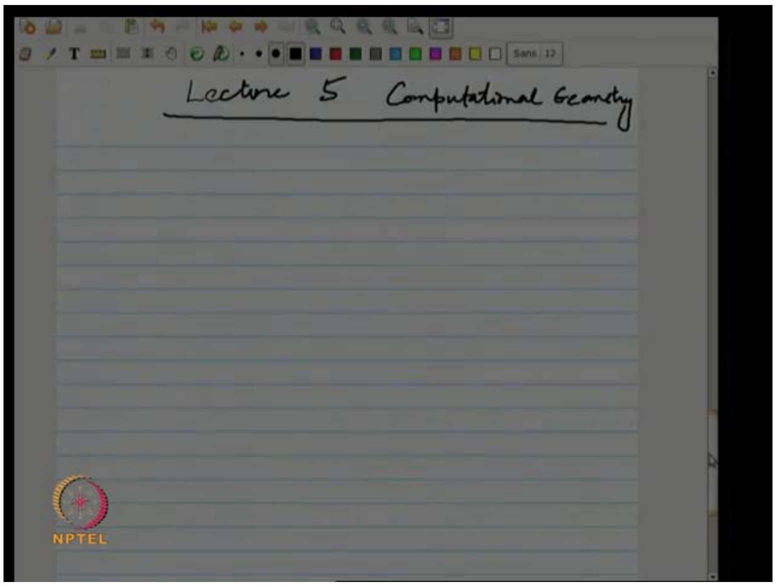
The Plane Sweep Technique and application

Lecture No. # 02

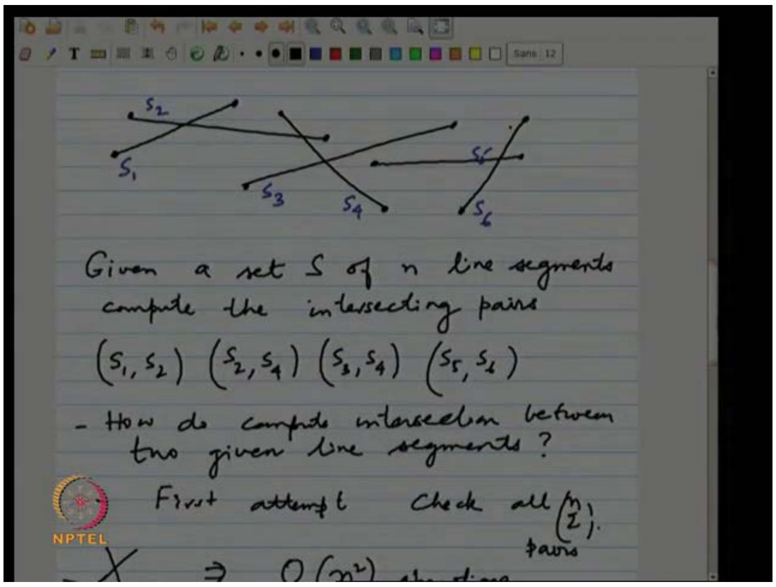
Segment Intersection problem

Welcome to the lecture 5, let me do a brief recap of what we did yesterday, that was essentially you know illustration of the technique of line sweep for the problem of finding pair wise intersection of given line segments, right.

(Refer Slide Time: 00:33**)**


Lecture 5 Computational Geometry

(Refer Slide Time: 00:46)



Given a set S of n line segments compute the intersecting pairs

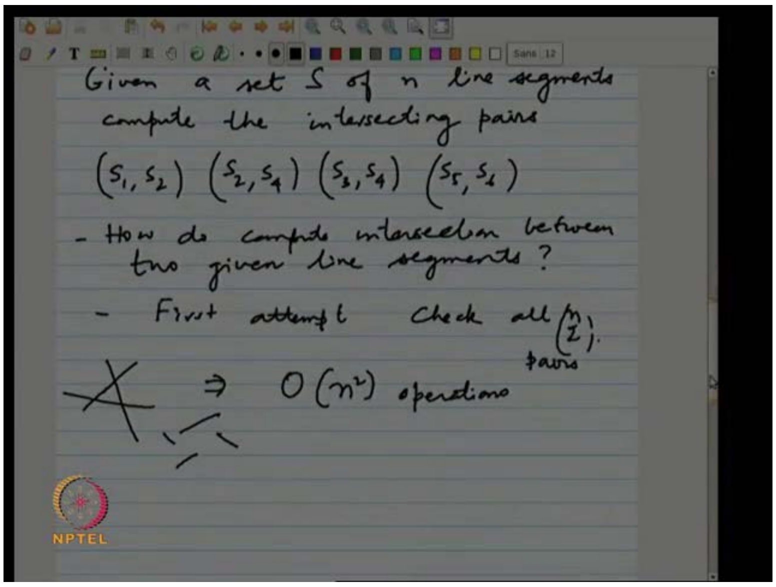
(S_1, S_2) (S_2, S_4) (S_3, S_4) (S_5, S_6)

- How do compute intersection between two given line segments?

First attempt Check all $\binom{n}{2}$ pairs

$\Rightarrow O(n^2)$

(Refer Slide Time: 00:55)



Given a set S of n line segments compute the intersecting pairs

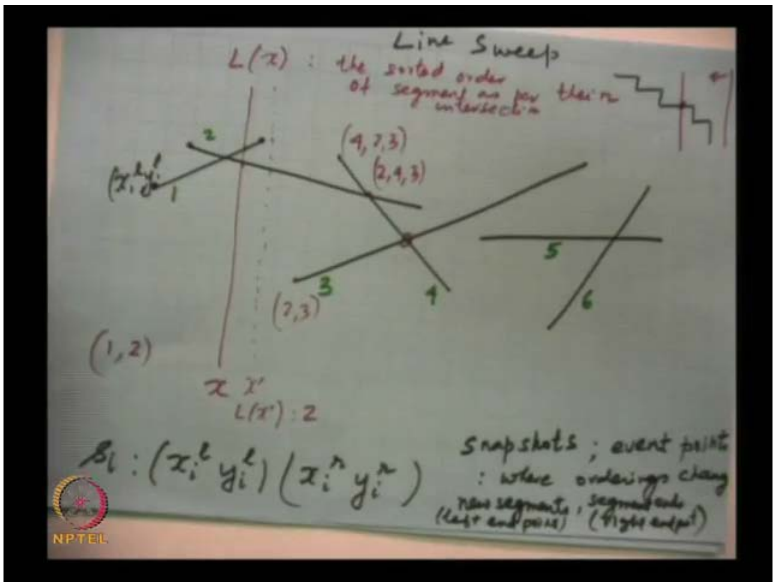
(S_1, S_2) (S_2, S_4) (S_3, S_4) (S_5, S_6)

- How do compute intersection between two given line segments?
- First attempt Check all $\binom{n}{2}$ pairs

$\Rightarrow O(n^2)$ operations

So, given a set of lines segments, we want to find all the pair wise intersection all the intersection pairs and we would like to do it in time that is prepositional to the actual number of intersection the order n square. Algorithm is rather trivial in straight forward, but you know it is it is kind of over kill for the situation where you have very few intersection right. So, for that we developed this technique where, we do the line sweep and the line sweep method is… just going back to the yesterday's example.

(Refer Slide Time: 01:24)


Line Sweep
L(x) : the sorted order of segments as per their intersection
(4,2,3)
(2,4,3)
(2,3)
(1,2)
L(x'): 2
$s_i : (x_i^l\, y_i^l)\, (x_i^r\, y_i^r)$
Snapshots ; event point : where orderings change
new segments, (left endpoint)
segment ends (right endpt)
NPTEL

So, this this red vertical line is a line that we are hypothetically sweeping across this serial set of line segments like this and at the any point of time, you know the snap shot is essentially the intersection of the line segment to this vertical line and we also maintained what is called the sorted intersections, of sorted from top to bottom of the line segments with the the sweep lines. So, for instance, at this juncture, you know the sorted order is 4, 2, 3 and after the intersection it actually they comes 2, 4, 3 and so on, so forth. And in this process, as we sweep past all the intersection points and we we can record this events correctly and then essentially we have done our job.

Yes.. Conversation between student and professor (Refer Slide Time: 02:22) – Not clear

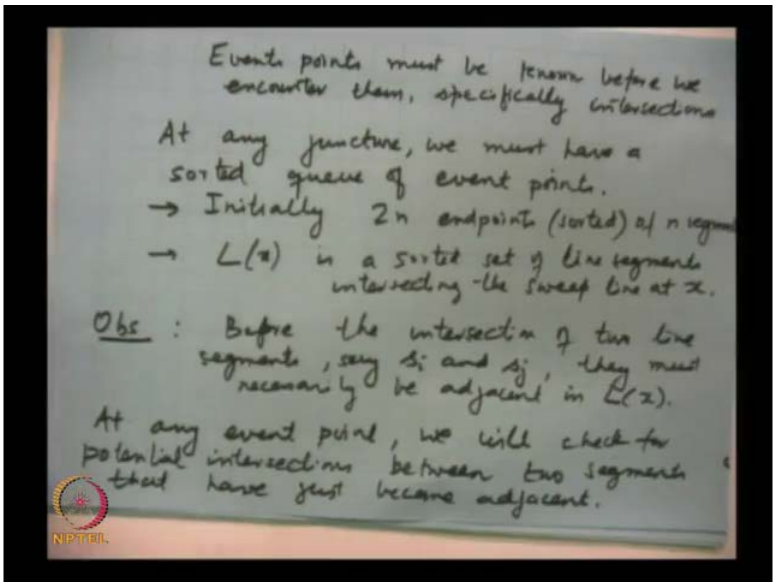
No no, so I had also made this assumption, let us assume that there is no vertical line and how we can make through this assumption, you can just do a random rotation, I can get get over this. You just take any given set of lines segments, not that we cannot handle it, but you know these are very special cases for which I am saying that, easy fix is that you just rotate the coordinate axis you by some random amount theta and that we will fix everything.

(( )) we may have to rotate the line many times (( ))

No no, there only n line segments, right and you have infinite choices of slopes. So, most for most of the slopes there would not be any vertical lines, that is where we exploiting the fact that we are dealing with it with the real plane. So, the questions like this will come of once in a valley, you know that, so you know remember this technique of you the random rotation, but there are other we can also directly deal it deal with it in algorithm, except that I do not want to deal get into these special cases.

So, this this is now how do we record this events, you we observe that so we observe that there are three kinds of event points, one is when a line segment begins, second when a line segment ends and when we encounter the intersection point.

(Refer Slide Time: 03:45)



Event points must be known before we encounter them, specifically intersections

At any juncture, we must have a sorted queue of event point.

→ Initially 2n endpoints (sorted) of n segments

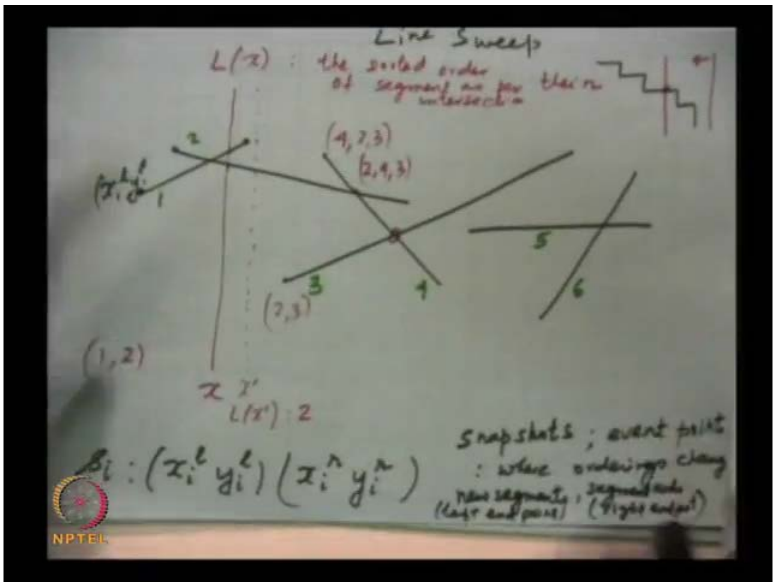
→ $L(x)$ is a sorted set of line segments intersecting the sweep line at x.

<u>Obs</u> : Before the intersection of two line segments, say s_i and s_j, they must necessarily be adjacent in $L(x)$.

At any event point, we will check for potential intersections between two segments that have just become adjacent.

So, at all these events the the L’s of x, which is the slotted set of the line segments intersecting the sweep line, you know they that under goes some change. So our events are exactly those points where the sorted set under goes a change, of course the actual intersection points will change continuously.

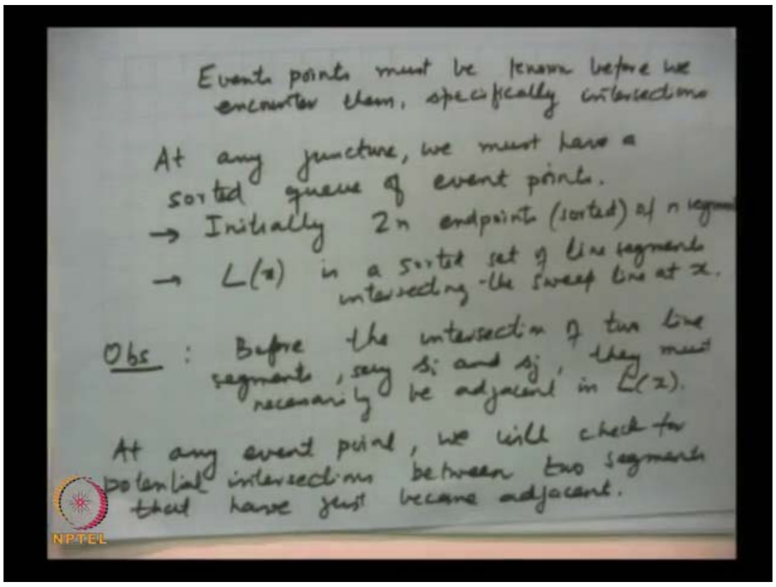
(Refer Slide Time: 04:19)


Line Sweep
L(x) : the sorted order of segments as per their intersection
(4,2,3)
(2,4,3)
(2,3)
(1,2)
x x'
L(x') : 2
$s_i : (x_i^l y_i^l)(x_i^r y_i^r)$
Snapshots ; event point
: where orderings change
New segments (left end point) ; segment ends (right end point)
NPTEL

Because as we move the as we move the vertical line, the actual intersection points going to change, so as we move from this from x, so from this line to this, sorry not even that, just a infinite similar change, obviously the actual intersection point are changing. So, from here these are becoming this intersection points, but we are actually only bothered about or we only record where the the sorted order is this changing, the combinatorial information is what is important to us, it is not exact point of the intersection. So, therefore, we only record are though events are only meaning full to us where there is a Change in this combinatorial information, not the actual value of the intersection points, this must be understood, alright.

So, then we know the that then therefore there is only three kind of an end points, new segments, you know when a segment ends and when there is an intersection point, now these two event points are known to us, at the time input is given to us, all the end points are known to us, left end point and right end points.

(Refer Slide Time: 05:28)



Events points must be known before we encounter them, specifically intersections

At any juncture, we must have a sorted queue of event points.

→ Initially 2n endpoints (sorted) of n segm

→ $L(x)$ is a sorted set of line segments intersecting the sweep line at x.

<u>Obs</u> : Before the intersection of two line segments, say s_i and s_j, they must necessarily be adjacent in $L(x)$.

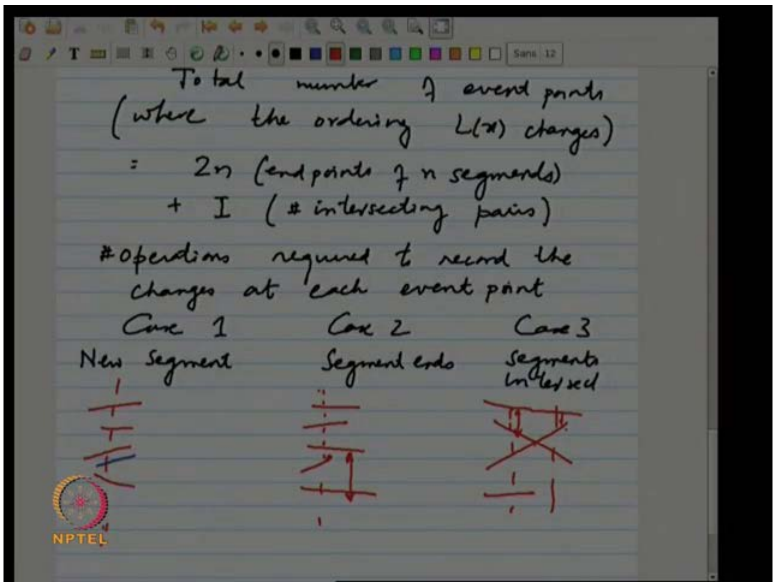
At any event point, we will check for potential intersections between two segments that have just become adjacent.

It is the interception point that we have to determinate and we must determinate them before we actually sweep pass the intersection point, so we must determine before, as we do that sweep, we must detect them before we actually hit the intersection point. And how do we do that and then we made this observation that before the intersection points of two lines segments they must necessarily become adjacent. So, this you know you can think about it just before you you cross the interception points, just the infinitesimal before that and just after you crossing interception points, these two intersecting pairs must be adjacent, there cannot be anything in between them alright.

And this is also again there are some confusion about what happens if there is a situation where there are concurrent line segment, so something like this going through a point, what do I really mean by that? So, once again easy assumptions, simple assumptions is to avoid these cases, that we assume that this does not happens, but even if this prior to happen, we can pretend that these are actually three intersection points, the two pairs, let us say 1, 2 and 3, so I can treat them as 1, 2, 2, 3 and 1, 3, alright and I can order them in any fashion that I want.

So, we can handle it either by assuming that we do not they do not exists or even if they exists we can actually hypothetically order them and deal with the in in some order, again this is this is not going to be any bottleneck in the algorithm. So, so then at any event point we will check, so what happens to event point? the the I said the definition of the event point is such that it is exactly those places where the the ordering can change. So, to detect this interception between s i and s j, we know that this information must, I mean if there were not adjacent before they must have became adjacent after some event points. So, if from you know if you just you know that they just an inductive prove, if you are sweeping from the left to the right, starting from a base scale, either left end point is recorded correctly and suppose we have recorded all the event points up to this, up to a certain junction correctly, then you you know all the orderings must be correctly maintained at any point of time and therefore, when you hit the n plus first n plus first event point, which could be an intersection point, you know our the the two line segments that must intersect must have became adjacent because of some prior event points, which by induction we have recorded correctly, that is the proof of correctness of the whole thing. So, how many event points to encounter?

(Refer Slide Time: 08:10)



Total number of event points
(where the ordering $L(x)$ changes)
: $2n$ (endpoints of n segments)
+ I (# intersecting pairs)

#operations required to record the changes at each event point

Case 1 New Segment

Case 2 Segment ends

Case 3 Segments intersect

So, we encounter total number of event points, ok, where the ordering L s of x change and there is a 2 n end points plus let us call this let us call this capital I number of intersecting pairs, alright. So, these are the total number of event points that we encounter and I just argued that we identify all the event points correctly. So, now to analyze running time of the algorithm, we need to only analyze how much work, how much time we spent for dealing with each event point. So, let us say number of operations required to record the changes at each event point, so if we get a bound on this, then we multiply this by the total number of event points and we are done.

So, here is where again, even again we we we can if we make this simplifying assumption, but I I also mention that it is not necessary, there is no concurrent three segments at a given point, the total number of changes that can happen at any event point with this assumption is constant, right, either you know some segments, one segment will be insert it so you know two adjacent will be effected or one segment could be deleted, again you know just two adjacent will be replaced by one adjacency and if there is an intersection then exactly one adjacent is effected, that is well not quite. So, let let us let us do this either, so these are three cases, I would say case 1, case 2, case 3, this is new segment, segment ends, segment intersect.

So, when a new segment comes then so the situation could have been you know something like this, you know some this was the situation at the at the previous event point, a new segment comes in, maybe it is inserted somewhere here. So, then what happens is that precisely there is one adjacent separated between the top rate and so; the blue segment and the one, on top of it and the blue segment in the red segment below, it right. So, these are the two new adjacent that is created, it if a segment ends, then you have a situation like this, let us say this segment ends.

So, if this segments ends then wants this drops out, you know these two becomes adjacent basically. So, there is only some local information or this is local changes are happening, alright and if a segment intersect, two segments intersect, clearly this is just swap, if if the ordering here and the ordering here they are just you know, for these two segments they have changed. And if when they change, then of course you know something that was below it and above it, so there is this change has to be recorded you know because now instead of these two being adjacent, now sorry these two will become an adjacent, ok.

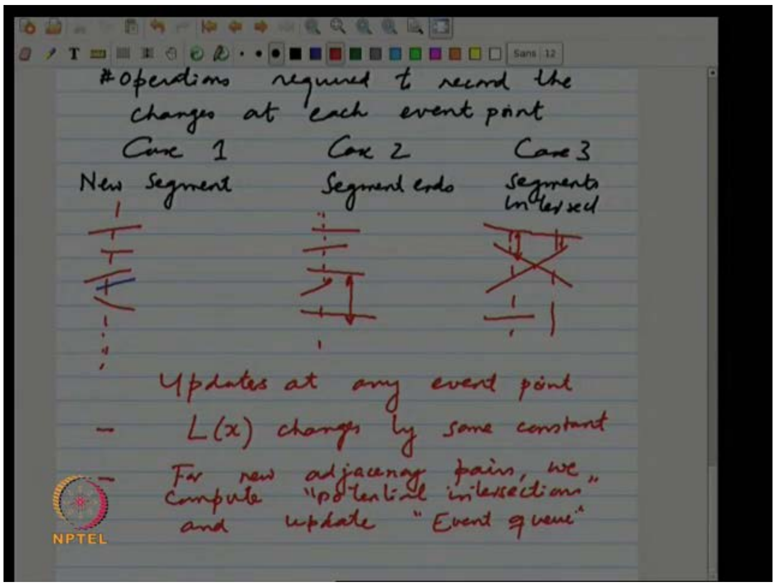
(( ))

You know again the assumption is that there will be there would not be any two interception points are in the same vertical line

(( )).

Yeah, yeah again if there is one, you just rotate the axis, there would not be anything. So, exactly. So, that for that for simplifying that analysis we are making all this assumption, otherwise even that is not required, right. So, so only some constant number of adjacent information is changed at any event point right, so since there is constant number of adjacent is changing there is not a whole lot of work we need to do to update this information, but whatever it is, we have to do some update, What are the updates that are happening?

(Refer Slide Time: 13:38)



#operations required to record the changes at each event point

Case 1 New Segment

Case 2 Segment ends

Case 3 Segments intersect

Updates at any event point

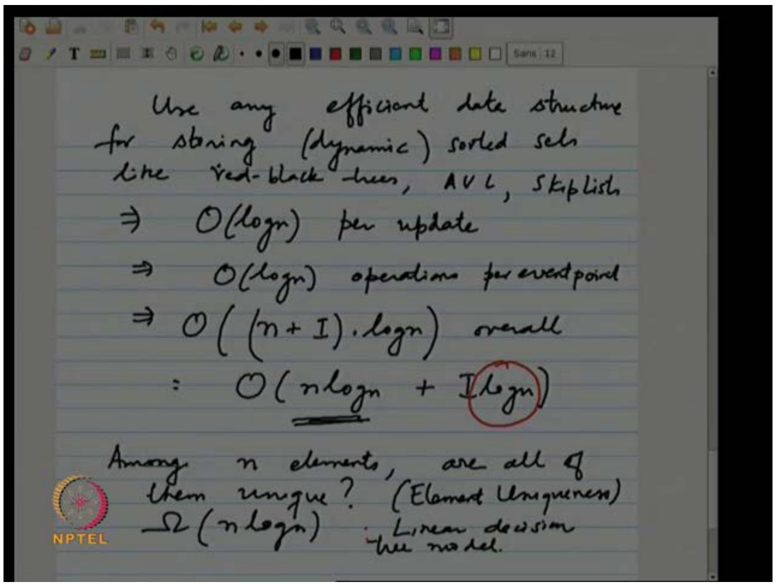
- L(x) changes by some constant
- For new adjacency pairs, we compute "potential intersections" and update "Event queue"

So, the update is happening to essentially update at an event point, so the updates that an event point, one is there is some change to this L’s of x right, L’s of x changes by some constant, relative constants, no L’s of x and L prime of x. You know there will be some constant number of changes in the adjacencies and of course, if we detect the movement something become adjacent, we we have to record, so the potential intersection point, right. So, for new adjacency fears we compute the potential intersections and we must record this in the there is a q that we are maintaining for the q of event points, right and update.

So, L’s of x and this event q both are what? Both are some sorted sets, L’s of x is sorted according in the in the y direction and the event q is sorted in the x direction, right, because event q is telling us which is the next event in increasing x and L’s of x is telling us you know what is the ordering of the segments when they intersect the vertical line, so that ordering in the y axis. So, you see we are dealing with the two dimensional problem, so we have we are actually maintaining some kind information both dimensions, one in the x axis for the order of the events and one on the y axis in the ordered segments intersects the sweep line.

So, what is the standard way of? So, when we have to update this information at any event point, we have to update these you know we need to update this sorted sets, in other words, we need to update some kind of data structures. So, what is the standard data structure for marinating the sorted sets? So, you just maintain some kind of dictionary you know whether it is it is (( )) or it is a (( )) whatever. So, you you have, you basically need a data structure right, so the data structure required.

(Refer Slide Time: 16:55)



Use any efficient data structure for storing (dynamic) sorted sets like red-black trees, AVL, Skiplists

$\Rightarrow O(\log n)$ per update

$\Rightarrow O(\log n)$ operations per event point

$\Rightarrow O((n+I).\log n)$ overall

$: O(\underline{n\log n} + I\log n)$

Among n elements, are all of them unique? (Element Uniqueness)

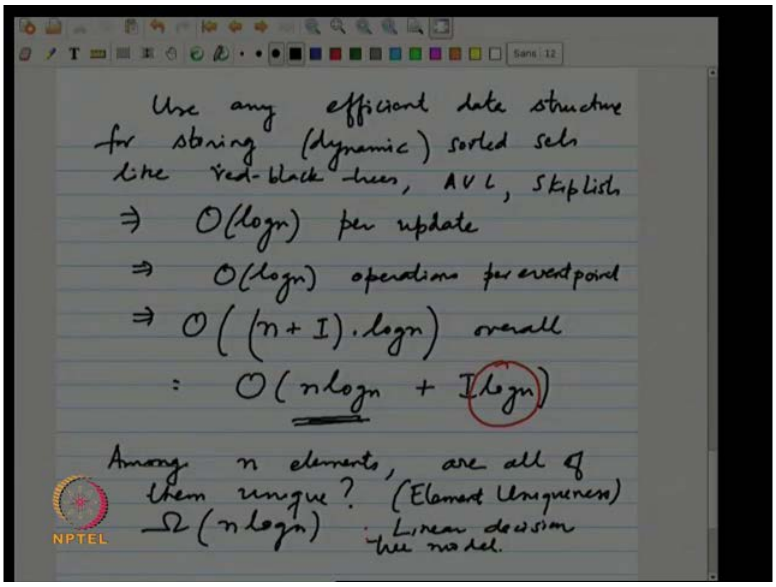
$\Omega(n\log n)$: Linear decision tree model.

So, use any use any efficient data structure for it is a dynamic set, right, dynamic red black trees A V l trees or even something skip lists, which is essentially implies that order log n per update, implies order log n operations per event point and implies order n plus number of intersections times log n, any doubts. So, we have succeed in designing and algorithm which is not a particularly complicated thing, but you know one would, one if one did know about these lines sweep methods, you know one you know one could not just come up with it just like that, but after you know this basic idea, it is not that difficult to come up with this. And this is also very straight forward and what you have got there is something that is certainly the running time in is is sensitive to the number of intersections.

So, it is you can write this whole thing also as order n log n plus I log n, just to if I open out this terms and it turns out that for finding even one intersection, if I ask you this question, tell me if two lines out of the given line, n line segments is there. Even one intersecting period, this is a decision problem, it is also saying you know if I give you n numbers and ask you is there either two number that have the same value, so this is a is a very basic fundamental equation, so among n elements are all of them unique, it is for the element uniqueness problem.

So, this element uniqueness problem is a very basic problems and it is particularly useful for you know devising or you know for for arguing about lower bounds particularly. So, this this line segment intersection problem is the special case of the element uniqueness problem that given n line segments is there even one intersection. So, this problem known to have a lower bound of something like omega n log n, to find out if there is a duplicate, you essentially need to sort, that it is basic idea and so, therefore, the first term in our line segment algorithm is unavailable, the n log n part.

(Refer Slide Time: 16:55)



Use any efficient data structure for storing (dynamic) sorted sets like red-black trees, AVL, Skiplists

$\Rightarrow$ $O(\log n)$ per update

$\Rightarrow$ $O(\log n)$ operations per event point

$\Rightarrow$ $O((n+I).\log n)$ overall

$: \quad O(\underline{n\log n} + I\log n)$

Among n elements, are all of them unique? (Element Uniqueness)

$\Omega(n\log n)$: Linear decision tree model.

The second term, clearly I is the unavailable, because it must be at least propositional to the total number of intersections, so it is only this one term slide (( )) here, basically if we did not have this term, then this was an optimal algorithm, but it turns out that actually this improves this algorithm. Of course that improvement to this optimal bound that n log n plus I makes it about ten times more complicate and practically impossible to imply implement, it just becomes that much complicated. So, if you are willing to live with that extra log n with I, then you have a very practical and elegant algorithm (( )).

No, no no I am not saying that, you have to sort for element uniqueness. see

We are getting the (( ))

No, n log n mode is not, it it it is not that you get it from sorting, you know there are technique by from which you get directly this n log n bound, what I am saying is that if the natural thing that would come to your mind to find out if in all elements is unique, is by sorting and and there is a lower bound, you know it is not by reducing sorting to this, but but essentially there is a independent lower bound with a different kind of an augment, which says there is a n log n lower bound. So, all I am saying is that, so it essential means that you know is that you have to sort (( )).

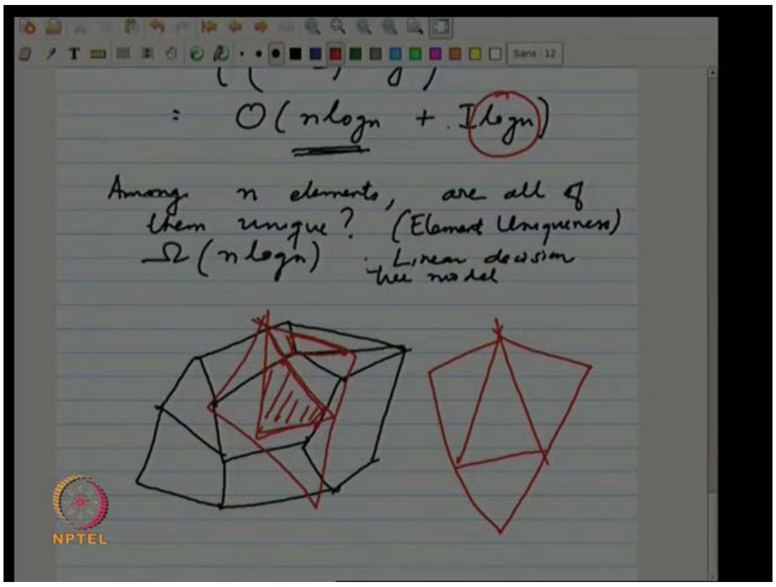
Yeah, I I will go through the argument when I discuses about models of computations ok. It will it requires what is called, you know it is called, in the least we have to consider what is called a linear decision free model, in the least we will have to deal with this. See, in sorting we are considering ourselves to a very special kind of linear decision that if x i or less then x j, but yeah just comparison comparison is a special case of a linear decision entry model, you we cannot restrict the since we are in the in the domain of geometry we should not restrict ourselves too just that. So, we we will.

But what you know, why would you restrict your yourself to comparison, I I we will actually domain. You know you can prove the lower bound with a you know more in a in a more power full model, not only in a linear decision entry model, we will be able to actually prove it in what is called the algebraic decision tree model, where you know things can take quadratic. You know you can deal with quadratic equation, but I do not want to address it right now, just try it to wait may be to the middle of the semester, you know I will take it up.

So, so in that linear decision tree model, you know you can prove directly a lower bound of n log n, what I am saying is that so then it amounts to saying that you you need to sort. If it is n log n then why not sort? See lower bounds are extremely sensitive to the model of computation that I I often mention that even this n log n bound for sorting is a miss leading lower bound, it only holds for comparison model, the movement you allow things like arithmetic operations, particularly divisions and multiplications and indirect address, that n log n lower bound does not hold and your programming language actually supports all these instructions to the n log n lower bound, is a completely miss leading lower bound.

There are actually algorithms that do better than n log n and though though though actually use the power of divisions and in directed result. So, this was the, so here this this basically concludes our discussion on how do you find pair wise intersections in a set of given set of lines segments. Someone could have asked me why know, you know what is what are the applications of this problem, right, so there are many applications, as we as we go in the courses there is basic many fundamental applications.

(Refer Slide Time: 25:56)



$: O(n\log n + \boxed{I\log n})$

Among n elements, are all of them unique? (Element Uniqueness)

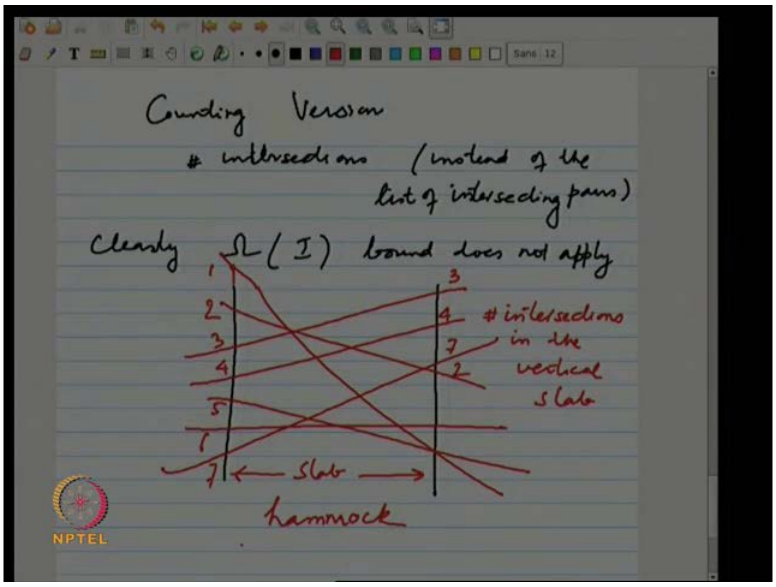
$\Omega(n\log n)$: Linear decision tree model

But one you know sort of very natural application is that if have let us say two planer maps, which means that you know I have a map like this, something or some states, some countries, whatever you know some information you know that appears in a map and I have let us say another map, it is looks like this. And for some reason I need to overlay it, so it mean that you know I will I will be doing something like this. So, the movement you need to compute any kind of over lays of this kind of maps that will amount to compute in these intersection points, so this can be solved using what we discussed first, right.

So, if I need to over lay them essentially, I am redefining my faces, the planer map whatever I am have to the smaller ones you know, so these these becomes my faces, right so and these, the intersection points actually contributes to the vertices, this is of course a special case where there are some what similar, yeah somewhat similar algorithms, because we are actually dealing with two different sets of non-intersecting lines segment. So, even this can be exploited to get something similar, but then I do not want to do, that is this one of the basic applications that if if I can do line segment intersection I can solve this over lay point.

Yeah, so that is a good question, so what if only I am only interested in finding the number of intersections, rather than the intersecting pairs and this problem of course does not have the lower bound that we are discussing n log n plus I, because the total number of intersection is only a number between 0 to n square. So, we can no longer claim that we done this algorithm and then after you get the intersection pairs we just count them, but then we have already then spent time prepositional to the total number of interesting pairs which is not necessary, so the counting version of this that is a good question, right.

(Refer Slide Time: 28:11)



Counting Version

# intersections (instead of the list of intersecting pairs)

Clearly $\Omega(I)$ bound does not apply

So, clearly omega I bound does not apply, so I will give I I will give a very special case and that will also demonstrate that you know how one can actually solve this problem much faster, ok. (( ))

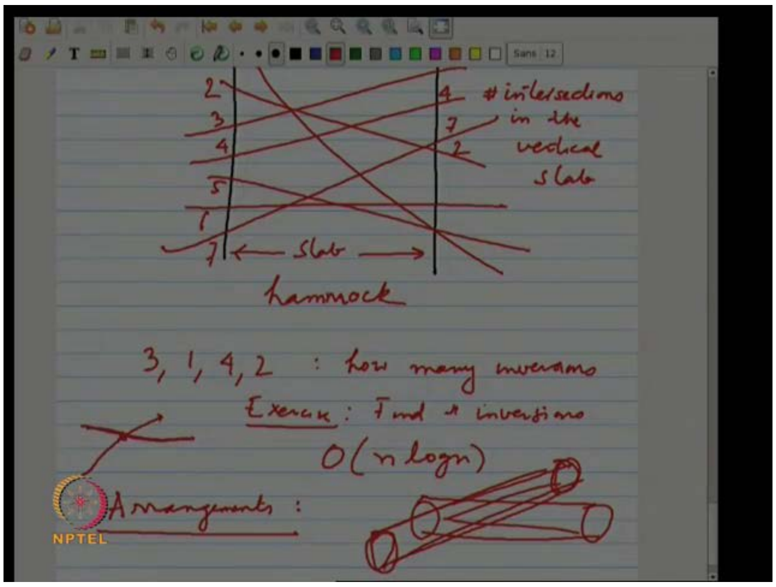
Yeah, so if if you are only looking that problem of detecting, there is if there is any intersection solving the counter counting version of ICs. Something (( )) Could be but then. No, so it it appears, that counting is harder, but it turn out that, you know you can you can actually also count very quickly, so you do not have to, so you know that just to detect is there an intersection, I I told you that any element uniqueness lower bound applies that is an n log n. And for the counting version let me go through that and then we will see you know what I am saying. So, suppose I have is a very special case of counting, where I have, I I want to do the following, I have a vertical slab and I have line segments.

Now, it is all clotted up, you know so sometimes actually geometry the structure is called a hammock structure, it looks like a hammock. So, the line segments do not end or begin within this slab, all line segments completely go through this slab, alright. And now I am interested in this particular problem of counting version, I want to know how many intersections occur within this vertical slab, number of lines in this slab.

So, one you know one way to solve this problem would be to apply our line segment intersection problem algorithm, do the line sweep, report the intersecting pairs and of course, if I know the intersecting pairs, I know the number of intersections also, right. But this one can be solved much faster, I do not need to actually report all intersections and why should I report all the intersection pairs, I am not interested in that. So, what did we actually want, what was the basically observation that we used to develop the previous algorithm, that when there is an intersection, there is a swap in the ordering.

So, can you generalize the observation to this, right, so I I know let us say ordering, if I know the ordering of line segments here, let us call this 1, 2, 3, 4, 5, 6, 7, then I can figure out what is the ordering on the other end, so this is 3, this is 4, this is seven, this 2, there is some ordering. So, I know the, I if I know the ordering of the segments on the both sides, it is and how quickly you can find that? Yes, I am saying that all segments go across this vertical slab, there is no no segments starting or ending in the vertical slab. So, how do you find this ordering? Just just sort, right, I find the intersection intersection points with the left boundary, I find the intersection points with the right boundary and sort the intersection points in the y direction and I have a ordering? So, I know this ordering on both sides, so how can I use that to find the number of intersections. (( )) Inversion is happen, right.

(Refer Slide Time: 33:25)



2
3
4
5
1
7
4
7
2
# intersections in the vertical slab
slab
hammock
3, 1, 4, 2 : how many inversions
Exercise : Find # inversions
O(n logn)
Arrangements :
NPTEL

So, so do you know how to find inversions? It is just a problem that, you know I I could have, suppose I have four elements that I want to sort and it is given to us in this fashion 3, 1, 4, 2, how many inversions are there, pair wise inversions? Yeah, so the the actually sorted set is 1 2 3 4, so this is what is given to you and I am asking you how many inversions? How can I find it? How can and how quickly can I find it? Can be more than three and what is the maximum number of inversions?

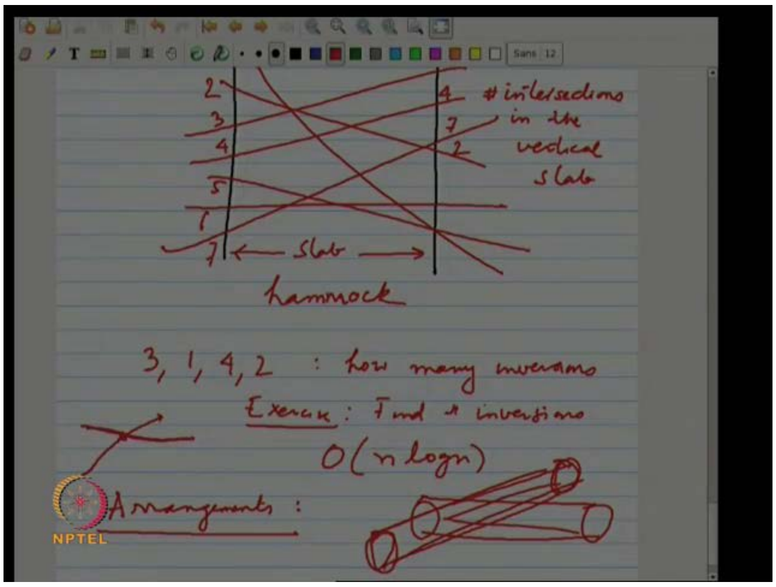
And choose to right the reveres sorted set every pair is inverted, right, so are you confused that the number of inversions will give you the number of intersection points. So, no if two lines are inverted, I mean for any two segments are inverted they must have an intersection in that slab, right, so that is obvious, but how how do you find this number of inversions.

(( )) Good, so I think this is an exercise for you, find number of inversions, forget about you know line segments, forget about intersections in a just give a set of elements, any set of elements and you know which can be can be ordered. Find the number of inversions and we should be able to device an n log n algorithm for that, which means that we can actually solve this problem in a log n time, which is much faster than actually reporting the intersection. Of course, this is does not solve the general problem, this is only for a vertical slab; because now we have we have this situation that you know segments could intersect, sorry end or begin within the slab, right. So, we have a more general situation, but this should give you an idea that that problem can also be solved much faster and in fact, that the general problem itself can be solved an n log n.

Well, it is it is not straight forward, let me say that it is not straight forward were it could be done so any more interesting variations of intersection problems that you can think off? More dimension; higher dimensions, great, so what happens in higher dimension? So, in higher dimensions is this problem, will a relevant problem, first of all lets think about it. If I give lines in three dimensional spaces, why would you be interested in, a how likely is it that they are going to be intersect (( )).

Yeah, in in yeah if you ask you know a probability theorist then, I have given some lines of space, in three dimensional space, the probability of intersection is 0, because the measure of the lines in the three dimensional is 0 actually, right. So, so that of course you know that we are not looking at average cases something like that.

(Refer Slide Time: 33:25)



2
3
4
5
7
7
4
7
2
# intersections in the vertical slab
slab
hammock
3, 1, 4, 2 : how many inversions
Exercise : Find # inversions
O(n logn)
Arrangements :
NPTEL

So, there are, So, it it yeah those solving the problem for what are called, these are called patches. So, it is not planes again, planes will have, you know have n planes in a two dimensional planes, in three dimension, of course, every pair is going to be intersect in lines. So, we will talk about these kinds of things when we discussed something called arrangements. So, this is a again a very you know sort of what we say very basic basic problem in in geometry, that if I give you you know some what, this it could be anything, it could be all kinds of any kind of algebraic verities, you know and I want find out how they interact between themselves that whole structure you know.

So, segments are on on on plane, so they partition the plane into some phases you know it is like a planer graph, so that basically the structure, if you go to higher dimension you talk about two dimensional planes in 3 D. You can go to even higher dimensions, any d dimensional space and talk about basically you know zero dimensional thinks which are points, you know one dimensional things that are lines, two dimensional things that are planes. So, in d dimensional, we will be dealing with about d minus one verity and they can also make may be linear, may not be linear, so in general this problem is a very messy and difficult problem and you do not want to actually handle it.

But at least for the the linear verities, that as you know planes or higher planes or higher dimensional, people have looked at this problem and we will also do some discussion about it, because it turns out the many problems can be tackled by studying arrangements. The (( )) structure of arrangements it carries lots of interesting information, because they are geometrically constrained, you know I cannot just make two hyper planes intersect in more than one place, you know because they are linear structures and so this actually imposes the basic geometric constrains and that can be used to solve you know many interesting problems right, but again I will postponed this discussion and we will talk about arrangements, it is a point of course.

However, I still go back to the original problem, how about intersecting lines in three dimensions, why do you think you should be interested in that problem? I claimed that it is a very interesting important problem, although I you know said something to the country, it is actually very important problem, you know think about any practical applications for that.

Intersection of rays, you know why would you interest in intersection of rays? I mean there is intersect, fine let them intersect, have you done a courses in graphics, that is why I think you are talking about rays intersection, ray tracing also. Even if two rays intersect, it does not make any difference to at least the way the you know people solve the graphics problem of visualization, exactly very good.

So, these days you you hear about these these collision paths, right, so you know aircrafts taking off from air ports it is it is one of the most important thing is to you know sort of the the the ATC control, their traffic control is to figure out, if there is any pass potential collision of the of the landing path and the reason is that they have to be very care full all that it could happen. So, it is not arbitrary lines in space you know you are actually flying out of air port or landing into air port.

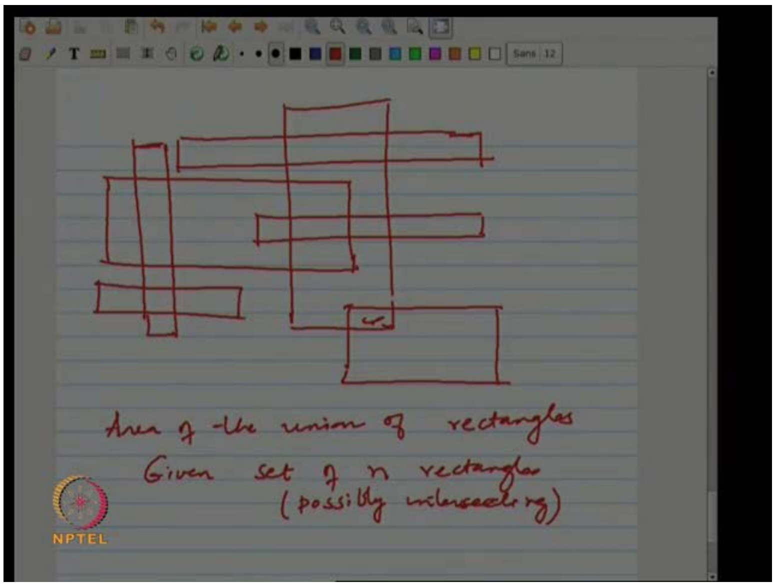
So, everything is converging into the air port, so the chances of the intersection is very high and more over the coeditor of the intersection is no longer just that zero dimensional line, but it is actually you know it is it is considered to be, I think if you come with even with a 10 kilo meters, another air craft or something that there is some definition that is called a potential collision.

So, you are actually talking about you know something not really zero dimensional thing, but you know corridors like this, which is essential the landing path or the take off path of the air ports and they have to actually figure out, that is the one of the problem people actually solved. I mean how well they solve it I do not know because there have been this kind of collisions, but this one practical application that you know if I if I know that where these aircrafts landing and taking off you the big air craft, air ports in world they have you know a few hundred aircrafts, you know landing in about you know within an hour or less then hours time. So, there are this scheduling has to be done very carefully and at a time, some time more than one air strip, so you have to figure this out very carefully.

Yeah, so I am saying that it is proximity in some sense, it is it is not zero dimensional, it is kind of any corridor, it has it is like a tube, but then it is a similar kind of a problem where I I have to find out whether there is an intersection also. It is not just intersection just like that, if they are staggered in time there is no intersection, so there is another dimension that is coming into the picture.

Not just three dimensions, but also the dimension of time is important, right, so it is actually four dimensional problems in some sense. So, collision detection needs another extra dimension, so even in two dimension it is actually a three dimensional problem, because you know I even if this one path and this is another path, you know there two cars are going or two trains are going, they will intersect or they will collide only if this happens at the same point in space and time, right. So, let me introduce another problem for which lines is is very truth fully used, let me switch to this one and that is the problem of, so let me just introduce the problem first. So, especially people who are interested in real s i layouts do the circuit design. So, what are they doing? Your transistors are nothing but you know orthogonal rectangular.

(Refer Slide Time: 43:14)

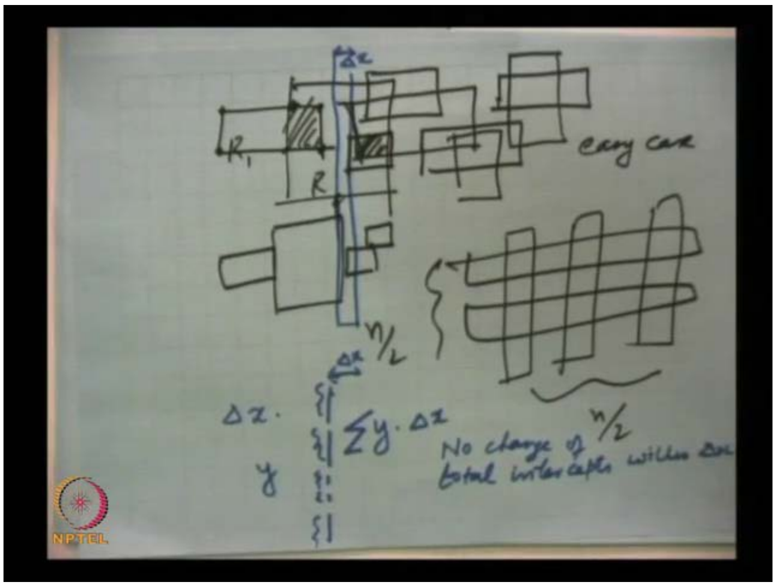


So, you have you know this is something, this is what is basically fabricated, something of this kind, so this meant to orthogonal, right. So, whenever these two rectangles intersect actually there is some kind of connection etcetera, so really it is a very complicated huge circuit, millions of transistors, millions of rectangles, right.

So, one very difficult problem and for which you know there is no real solution people use, (( )) people that you want to design, you want pack as in many rectangles are possible, but of course in a under certain constantans you cannot bend them closer than something etcetera etcetera. So, that is a problem that we are not going to, I mean we are not discussing right now, but a more basic problem, I mean more simpler looking problem is that if I if I give you this some lay out, suppose someone has designed this lay out, one of the parameters of this layout is to actually measure the area of the union of this rectangles. Union of rectangles you know will be a measure, the area of the union of the rectangles, union of the rectangles you know it would be a measure of something like you know how much current the circuit is, observe something like that.

So, area of the union of rectangles, so how would you give you know a set of n rectangles, given a set of n rectangles possibly intersecting, if they do not intersect trivial, alright.

(Refer Slide Time: 45:39)



Δx
R_1
R
easy case
$n/2$
Δx
$\Delta x \cdot$
y
$\Sigma y \cdot \Delta x$
$n/2$
No change of total intercepts within Δx

If all my rectangles, now let me switch over to this on, now if my rectangles are like this disjoint, easy case some just compute the area of the individual rectangles at demand (( )). What are we given as an input? We are given as input the description of the each rectangle, which means that. (( ))

Yeah yeah absolutely, so once I know the I know the coordinates of the rectangles I can conclude the area and I just sum them up and I am done, alright, but then that is not what we are interested in, we are looking at things like this, interception any way, you know any number of rectangles can intersect any where. How would you even attempt to solve the problem?

(( )) (( ))

Oh my goodness that is an inherently exponential thing, you know principal of inclusion and exclusion is inherently exponential, that right (( )). No, so geometry also geometric objects also have intersections, but if you if you rely on inclusion exclusion principal for any problem, any algorithmic problem, any computational problem, may you have already sort of you know we are aiming for something that is this this not not efficient.

(( ))

No, the movement you do this exclusion inclusion and…

(( ))

Ok

(( ))

No, so I have suppose these… So here these rectangles intersects, suppose this is R 1, this is R 2, this is the part that intersects, so what is that you are suggesting?

(( ))

Yeah, I find the area.

(( )) adding R 1 plus R 2, that is precisely inclusion exclusion, you know sometimes you know, so especially when you have three rectangles interesting, you must you know do that alternating sum right that is the problem, so that that that expansion is of terms, is in it exponential.

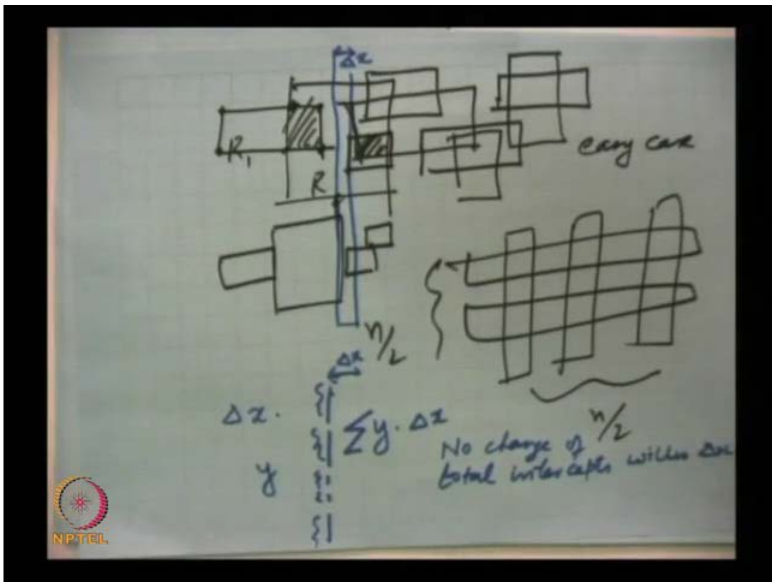
(( ))

ok very good very good.

(( ))

Very nice, so it is not the area, there is something, so what do you have is right area. So, the idea here is that, I thought that maybe we will go through another iteration of you know development, but this is a solution is is been already proposed. So, it is based on line sweep, which I said you know that was the purpose of this this example and it is what is being said.

(Refer Slide Time: 45:39)



As we sweep you know we will have to somehow keep track of the area that we sweep, that that is what is being said, right I though you would actually said that you know using line sweep I can I can actually compute all the interceptions and actually reduce this case to a situation of disjoint rectangles, that can be done, right. Suppose I actually end up computing all the intersection points, then I can actually, yeah, I can get you know this whole thing could look like this this you know. So, each small thing you know could be split up and these could I can I can make it to disjoint rectangles, then of course I can I can sum it up.

But if I do that, this is (( )) certainly, if I do that then I have this again this problem, that my solution, final salutation will actually be proportional to the number of intersections, whereas area is just one number, some you know so many square meters, for that is why should we need to compute all the intersection points.

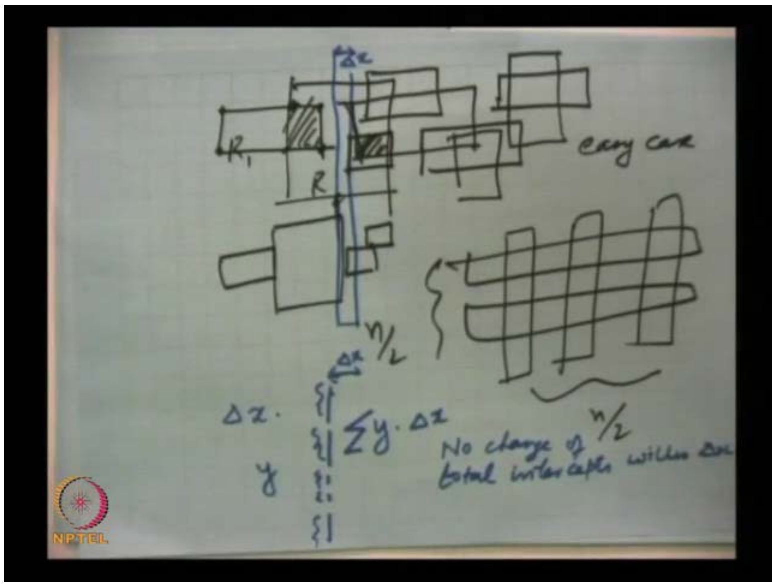
We can compute all the intersection points, reduce this to a case of disjoint rectangles and sum them up, alright, but in doing so you could have ended up computing all intersection points, then you know there could be a situation where there are just too many intersection points, no I can have this kind of thing, the cross bar situation, you know every. So, there are about n square, you know n n about two rectangles this way; you know what two rectangle this way and there will be n square intersections here.

So, then if we compute those intersections our solution will be at least n square, right, I would like to get a better faster solution. And what is being proposed that is just proposed, in the lines sweep somehow we need keep track of that. So, let we elaborate on that or at least what I what I thought was been proposed, so we we we do a line sweep again, we take this thing and and kind of you know sweep from left to right, what are the event points here?

The event points would be you know essentially begging of a rectangle or end of a rectangle here, by the way you are dealing with the vertical lines, so also. So, here vertical lines are not a problem, because everything is orthogonal, either the the side of the rectangle is parallel to the x-axis or the y-axis. So, there is no issue about that, so as we sweep across, suppose my line is somewhere here, suppose they move it by some infinite symbol delta x, alright, if I move by delta x, suppose we are here, so the question is that when I go move it by delta x, how much area have I swept? Delta x times, No (( )).

Not maximum, see there are gaps here, so this line may have lots of gaps, not just one gap, many gaps, but if I know, you know it is it is let me again amplify this, so essentially the gaps could be like you know let us let us look at the intersection of this line with the rectangles, it could look like this, this, this, this, this, right.

(Refer Slide Time: 45:39)



So, some areas being swept by delta x amount this plus this plus this plus this plus this, alright, only if by the way there is nothing that begins or ends here, alright. So, if we know the total measure of what is being swept across, suppose that is y, then the areas swept when we move by delta x is y times delta x.

And then the area is nothing but summation of this whole thing, but I mean suppose (( )) is delta x (( )) not whole thing. (( )) No, no, then we cannot apply, so this delta x, I am saying this in this delta x only if there is no change in these, in the intercepts yeah, only there is no change then then it is delta x time y and I have to sum it over that delta x where there is no change. So, there will be some pieces that I have to sum up, so no change of total intercepts with the delta x, then this will works. So, let us stop here today and we will think about it you know how we can do it, this actually requires also solving another interesting problem.