Computational Geometry

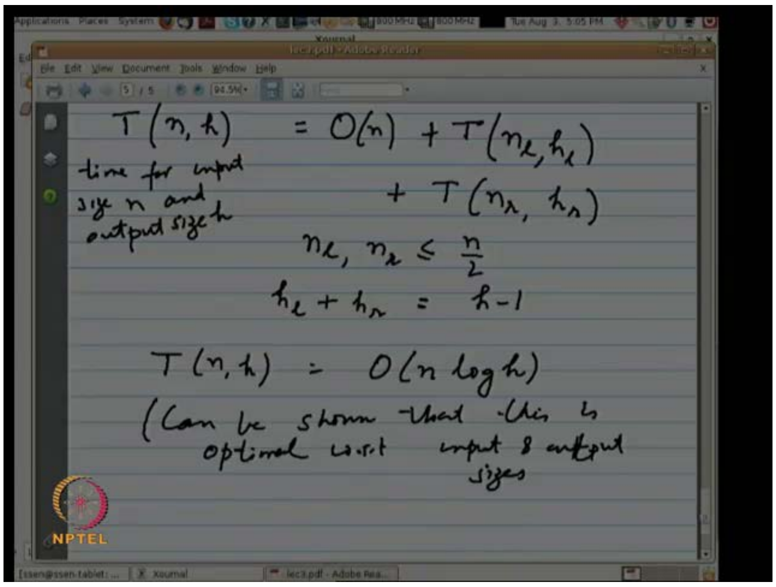
Prof. Sandeep Sen.
Department of Computer Science and Engineering
Indian Institute Of Technology, Delhi

Module No. # 03
The Plane Sweep Technique And Applications
Lecture No. # 01((04))
Line Sweep Method

So, welcome to lecture four. A quick recap of what we were doing last time. So, we were discussing the problem of the finding maximal in two dimensions and I did come across if you questions later regarding what happens if you do the maximal in three dimensions. How do things change and it turns out that in three dimension is a problem is of course, little bit more difficult as a view of in dimensions problems become more difficult, but the algorithm remains n log n.

So, in two dimensions is n log n, in three dimensions also have n log n. So, what I will do is, I too will not go through that particular problem in the class, three dimensions. I make it into an exercise problem I may give an enough hints so that you may concern. You should be able to solve it on your own and you can solve it essentially using the technique that we discussed. Let me just recap that.

(Refer Slide Time: 01:34)



$T(n,h) = O(n) + T(n_l, h_l) + T(n_r, h_r)$

time for input size n and output size h

$n_l, n_r \le \frac{n}{2}$

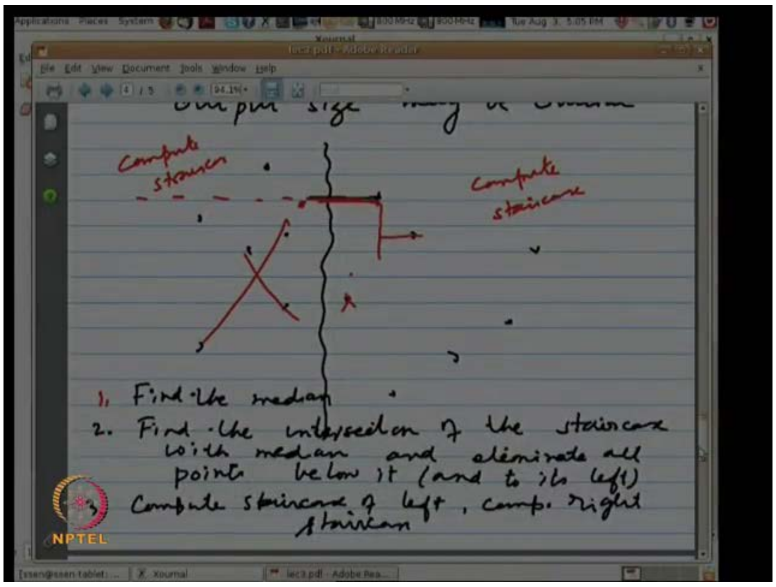
$h_l + h_r = h - 1$

$T(n,h) = O(n \log h)$

(Can be shown that this is optimal w.r.t input & output sizes

So, we discussed essentially three techniques. So, one was the traditional divide and conquer. Then we discuss something that was; well I did not give it a name, but when we started scanning from the highest x coordinate towards the lowest x coordinates and maintain some kind of information of a as a move from the highest to the lowest x coordinate maintain the information of the highest y coordinate.

(Refer Slide Time: 01:47)



1. Find the median
2. Find the intersection of the staircase with median and eliminate all points below it (and to its left)
3. Compute staircase of left, comp. right staircase

So, the basic principle of solving the problem in using dividing conquers or using this in a scanning from right to left was the same. That is we that information required was maintaining the highest x y coordinate of the points that have high x coordinate. So, we will be looking at it from right to left.

So, these are particular techniques. So, one reason that I actually use this problem is not because of problem in itself, it is a is a very useful problem, but it has of course, some application I said, but most so in terms of some of the algorithm techniques that you know. We will use later for other problems in computation geometry and this particular technique where we going from right to left has a name of a line sweep.

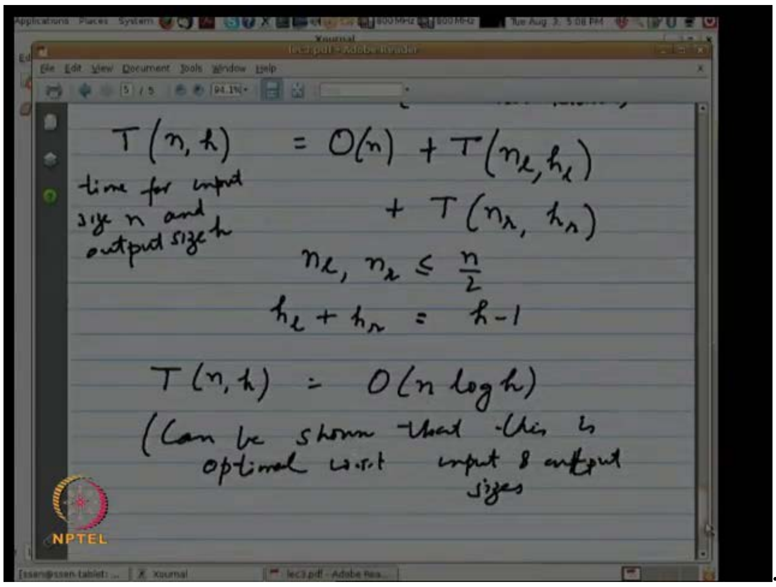
So, I will expose the technique using another problem today. So, line sweep is what we used and if you use line sweeping in three dimensions, then you can actually match the n log n algorithms in three dimensions also. The only complicated part or the one that you have to re-work is that when you again so, you sort let us say along x axis. So, you are going to sort along one another axis. So, here also when you when you have three coordinates, suppose you sort along x axis and then start from the maximum x coordinate to towards a lower x coordinates, at any point of time the points at you have already scanned cannot be dominated by the points that you have not scanned that remains.

But then you have to somehow maintain some more information about the points you scanned so for because now you dealing with three coordinates in not two coordinates. So, the current point it is scanning has a lower x coordinate then the other points, but it also must have a lower y and a z coordinates for that point not to be maximal. So, this additional information needs to be stored in a way so that you can quickly process each point. So, in the case of two dimensional is just one information highest y coordinate.

Here it is somehow a co-ordinate it is not simply highest y co-ordinate- and z co-ordinate, it is a mix of both. You cannot separately look at the highest x axis as y because they may belong to two different points. So, the same technique of just storing the highest x and the highest y is not going to work. So, will have to kind of store more information in terms of not just a two point, but it is actually is going to be. So, if you have think about it, it going to be actually a chain itself. So, it will be like a cross section. So, the first imagine how the three dimensional maximal looks like when three dimensional maximal will look like again some kind of stair case, but it is a three dimensional stair case.

And what you are going to store is a cross section of the three dimensional stair case and you have to store it in a data structure so that, you can quickly process a given point in let us about a login time. So, that eventually you can match that n log n bound. So, this is something that you I leave in an exercise with some hints for you, but you known it is interesting that you should get more experience in terms of visualizing some of these problems and solving it is in line sweep.

(Refer Slide Time: 05:13)



$$T(n,h) = O(n) + T(n_l, h_l) + T(n_r, h_r)$$

time for input size n and output size h

$$n_l, n_r \leq \frac{n}{2}$$

$$h_l + h_r = h-1$$

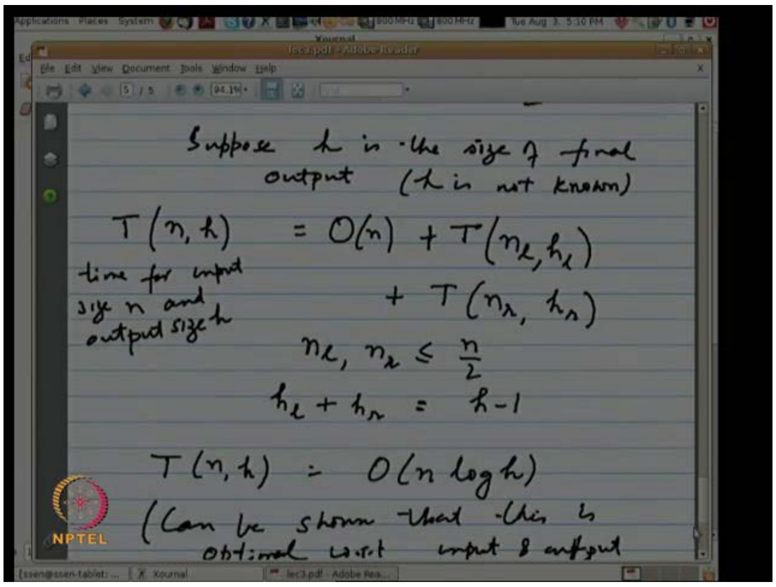
$$T(n,h) = O(n \log h)$$

(Can be shown that this is optimal w.r.t input & output sizes

And of course, the last probe the technique I discussed was you know something that was sensitive to the size of the output and this is very typical of geometric problems that running time if you can make a sensitive to the actual output size, then we get something that is something that is desirable because then, we are trying to match the best possible time both in terms of input and output sizes.

So, this n log h is found, where h is the output size and n is a input size, has order n running time when output is small and when output is large, let us even about square root of n you are you have about n log n running time n. It can also be shown that this is optimal.So, I will just make if just few more remarks about this a last thing this n log h algorithm.

(Refer Slide Time: 06:01)



Suppose h is the size of final output (h is not known)

$$T(n,h) = O(n) + T(n_l, h_l) + T(n_r, h_r)$$

time for input size n and output size h

$$n_l, n_r \le \frac{n}{2}$$

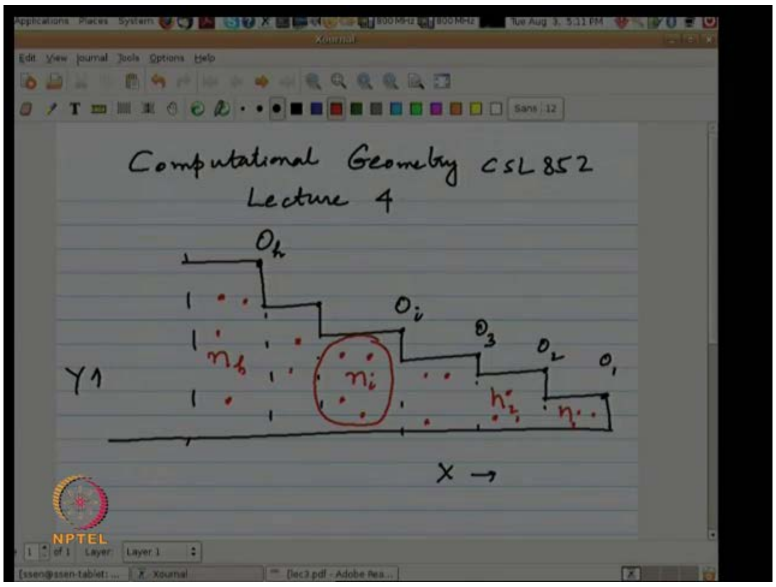
$$h_l + h_r = h-1$$

$$T(n,h) = O(n \log h)$$

(Can be shown that this is optimal w.r.t input & output

So, the way I analyzed it was writing a recurrence, this is a recurrence, yeah this is a recurrence of a wrote t is of n is equal to o of n a sorry. Next one, that is only in terms of inputs. So, t of n comma h where n is the input and h is the output size, although both h is unknown, I can still write this recurrence and then I claim that you know you can show prove rigorously that it has a running time and this recurrence is a solution of n log h. So, this is one way of proving this bound.

(Refer Slide Time: 06:36)



Computational Geometry CSL 852

Lecture 4

There is another way and a let me just give you some hints about it. So, your maximal looks like the finally, the output, its like the stair case and when we say this output size is h essentially their each points as I can call it let us say you know the output 1, output 2, output 3; in all the way of let us say output h, this is an output I let us say. So, these are the points so that there the each point is the outputs and the remaining points must be essentially under the stair case and they are distributed in some fashion. This is my x axis, this is my y axis and the remaining points which are not maximal, they appear under this stair case in some way. Now if you focus on just any of this a single step of this stair case, you know for instance let us say you are looking at this. So, all the points are specifically just under this step. Let us say that there about n's of i points. So, under the ith step there are n's of i points there. So, n's of 1 points here, n's of two points here and so on. And so forth. And there is n's of each points. Now the way that we solve the problem that is known we will found the median line wherever is the median lines. So, median line and you can calculate you know which is the step that process a median line.

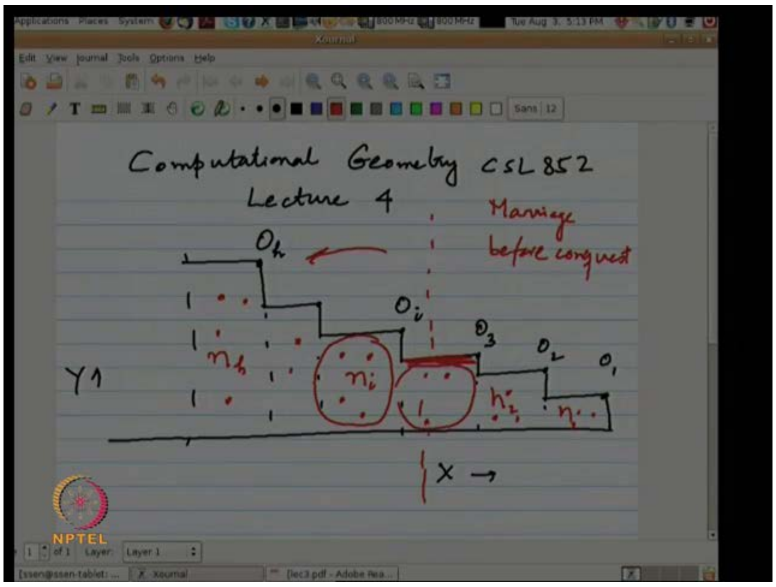
And how do you do that? We simply just looked at all the points right to the median line, found the highest y coordinate and we could compute this particular step and then we did the divided and conquer not it is not real divide and conquer because eventually we just it to paste. So, this is this actually has a name in the literature called marriage before conquest.

I do not know how it can happen. So, even before you actually solved the problem, you have you know that how they were paste it together rather than first solving for half the points solving for the half points and then some are trying to merge, that the way that do it do it in merge sort.

Here you actually see the doubt that this step is the one that crosses the median line and I only need to whatever solution I compute to the right of it and left of it we simply just paste it together.

So, I have actually married them even before I have actually solve the problems the two halves. This is called marriage before conquest. So, then. So, this points that are under this steps, at some point the moment you discover the step certainly all the points below the step we know.

(Refer Slide Time: 06:36)

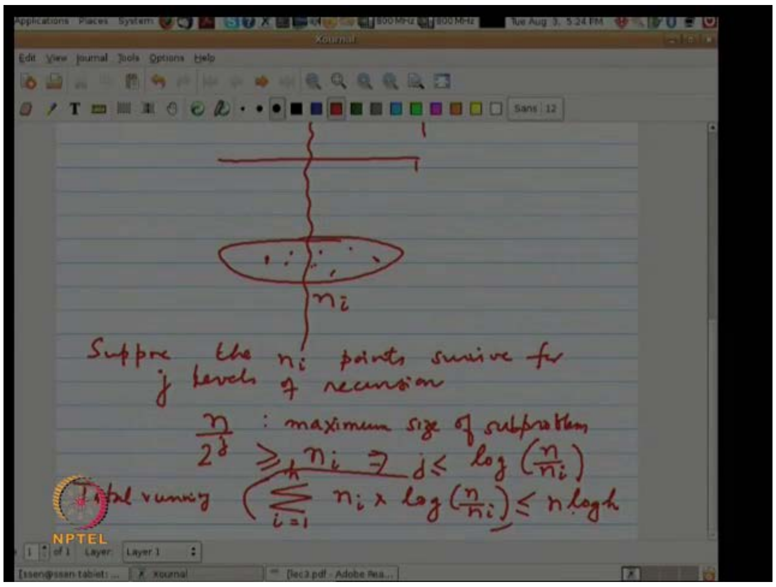


So, here all the points that come fall into this region should disappear the moment you discover the step because we know that there can never be the maximal points. So, now the question is that if you look at the ith step, there n's of i points some rate of course, all these numbers are unknown to us. We do not know no anything n 1, n 2 or even h. We certainly know that the moment the ith step will be discovered those points are going to disappear. And certainly these points, the moment I actually somehow manage to…So, initially started with the median line. So, this is the median line. So, the moment and then I am going to find a median of two halves. So, next time you know may be this will be the median line; this will be the median line.

But the moment I have a median line going through this block of points, these block of points are not going to survive after that. This is what I am saying. So, initially of course, the median went through this block of points. So, this block of points have to we handle that that they will disappear because that step we will discovered. So, if you think about this cluster of points, n sub i, the moment we have a median line going through the cluster at some point because divide and conquer we will break that cluster.

I will have a median line going through the cluster by just carrying on the recursion. The moment it goes to that that clusters, those points need not be processed any further because at that point you know that these are not the maximal points and they will actually be eliminated. So, we only need to handle these points as long as we have not chosen any point from the cluster for the median computation. So, let me illustrate this. So, I have let us say.

(Refer Slide Time: 11:47)



Suppose the n_i points survive for j levels of recursion

$\frac{n}{2^j}$: maximum size of subproblem

$\frac{n}{2^j} \geq n_i \Rightarrow j \leq \log\left(\frac{n}{n_i}\right)$

Total running $\left(\sum_{i=1} n_i \times \log\left(\frac{n}{n_i}\right) \leq n \log n\right.$

Of course, I do not know the clusters of points, but just a hypothetically you know I have a large cluster of points let us call its some means of i. So, there are some points under some step that is going to be discovered some time in the future. How long can we will not know how long can these points survive. Certainly, not beyond when we have some median line going through this cluster.

So, question is how long can this points survive without being a processed. I mean, without when you know having been speed up into two sets. Remember the median lines splits points into two halves, morally sequel halves. So, if this has n i points, how long can these not be split up because you doing well something more specific than that.

N by l I e l log (( )) yeah write.

So, essentially what we are saying is that suppose the n i points survive for j iterations. So, j levels of recursion. So, what can we claim about this j. Every time we do a recursion; the size of the sub problem is halved. So, at the jth level of recursion, the size of the maximum sub problem can be n over two to the power j and so, this is the size of the sub problem, maximum size of the sub problem. And the cluster has size n sub i. So, what is a relation between the cluster size and this maximum size? If the cluster has to survive, then this cluster master size, smaller than that or what larger than that.

Smaller than that, right. Otherwise it is a definitely going to get split up. You cannot have any sub problem larger than n over two to the power j. So, this cluster should have, if it has to survive j a recursion, then this should exceed the cluster size. Or in other words, j should be smaller than or equal to lets n log of n over n sub i. So, what we establish is that we cannot, this cluster cannot remain cannot survive beyond j activations and j has be to less than. So, it can only be processed for so many steps of the recursion.

If you think about it, what is that we are doing? I mean very similar to quick sort kind of thing. At any point of time, the processing that is being associated, you can think about it like that every point is essentially being compared with something whether it is to left of the median or right of the median, when you do the processing of course, median finding is linear time.

So, you can think about charging the cost to each point. At any level of recursion, the charge the total cost for division of the problem is linear. In other words you can average it out as order 1 per point. It is a order 1 per point. That is another way I can sum up the cost by looking at the total charges approved by each point at the end of the algorithm. So, initial step, we did the median finding, we did the divide step. The total amount the total the work we did was order n their n points.

So, if you have averaged it out, charge the average cost to each point, there is going to be order 1 per point. So, similarly at every level of recursion, a point can be charge at most order 1. If the points survive, if it disappears in the cluster breaks, the point is gone. I do need to charge it any more. So, if it survives for at most so many iterations. So, a point in cluster you can survive i for at most so many iterations which means that the total cost of algorithm, total running time can be bounded as sum over i for all the i clusters and for each cluster that has n of i points, the cost is this, log of n over n survive. So, I then I can simply just say n of i times log of n over n’s i. i equals you have if the each output points is this. So, no complicated recurrences, no two dimensional recurrence is nothing. Now then again not so, but then it is a its it is look like really you know you know it is not very transparent formula you know what is that we achieved unless some people has seen it before.

So, this actually has a very nice connection with what is called multi set sorting. Multi set sorting means if your sorting some numbers where there can be reputations so there are let us say the each kind of a sorry each different values, but the n elements and let us say that n sub i elements the of the ith kind. So, sorting actually becomes easier if you have lots of elements which have the same value. For instance case I mean the reveal case in all elements of the same value then sorting is the actually like in one step you have done. Even if they have two kinds of values you can still argues that just by doing one just by doing one actuation of one quick sort you done actually you just divided once in all the zeroes and all the one's will be clap together. So, it is a concept. So, you can see that you know if you repeat point, sorry repeat values sorting actually because faster the n log n.

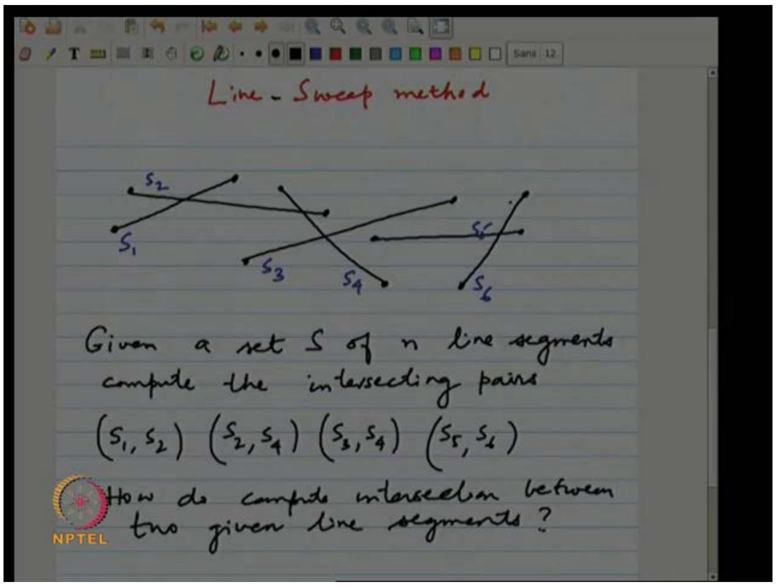
And this is essentially the formula that you get if you have seen that before that if you have n i points of the ith kinds sorry elements to the ith kind, then this is the running time and this is all way and this is again you can you can proof is easy this is always less than n log h. And in many cases depending on how the distribution is, this could be much better than n log h. So, what we have actually argued is the bound that is even its prior n log h, but it uses a further finer parameterization of the point set. That is how many points is below every step.

So, this formula is this even more characterization of the points said, and then the n see the n log h uses more characterizations in the sense that I am taking care of output size plus input size.

The formula that just derive you know has even finer kind of characterization that I know how many we do not know, but we are making use of number of points below each step. Of course, none of this information is required in the algorithm. The algorithm does not know anything about either h all the number of points in every step. This is just that in the analysis you know this falls out. And this formula and this thing, the nice thing this is actually bounded by n log h. So, I do not have to separately prove this is a n log h. This is always bounded by n log h and many cases this better. So, I will just conclude with this observation about the 2 d maxima and already mentioned that 3 d maximal can also be done in n log n time using just a more refined implementation of the line sweep algorithm.

(( )) this is the cost of multiple problem (( )). So, this is a lower bound also. So, you just do a (( )) yeah yeah, but of course, you know for a problem in geometry if you prove a lower bound, you have to work much harder. So, I will and I am not sure if I will get that for in this course. But I will talk about lower bound may be in one or two lectures. I can, I will write attention to this kind of things, but this bound is a classical information theoretic form.

(Refer Slide Time: 21:09)



Line - Sweep method

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 ?

Today I will introduce another problem and I said I want to basically expose this method of what I am calling line sweep a little bit more. So, consider the following problem. I am given a set of lines segments; do not confuse that line segment with these line sweep. This is just a name of a method this given some line segments. Lets only look at this is rather a small set because (( )) otherwise get catalog if I draw them.

Now, the goal is that given a set of lines segments, so, we want to compute the intersecting pairs. So, I can name them I can call this s 1, s 2, s 3, s 4, s 5 and s 6; a six line segments and this particular figure and we want to compute the intersecting pairs which means in this case, I want to report at the end of the algorithm s 1, s 2; is one intersecting pair. s 2, s 4, s 3, s 4, s 5, s 6. So, there are 1, 2, 3, 4 intersections. So, that this four intersecting pairs. So, how would you all with try to compute this intersecting pairs. First of all, let just for a movement, reflect an how do you given two lines segments, how do you compute the intersection. This is very basic question how do you compute intersection between two given line segments?

How the line segments described? How did you describe them the two end points? The natural description is look at the two end points the two line segments. So, here your input is given in form of two pairs of points, end points that each line segment. So, two pairs of such pairs.

So, how do just compute the given the intersection of two line segments. That is this must be the primitive. Just tell me yes or no I mean it can be done. So, whether you remember your analytical geometry or not you should you should to know how to compute the intersection of two lines? And therefore, you should be able to figure out whether two lines segments intersect. If both the points are to one side of the line contained in the segment, they do not intersect. Otherwise intersect way for some formula simple formula basically it say solving a pair of linear equations. So, that is it. So, there could be degenerate cases of pair lines and you know coincidental lines and all those things I am taking for granted that we know how to handle that all the special cases.

(( )) all the equation of the line you get whether the extended line refer (( )) right right.

So, after the after the extended line, it is very easy to find out if both the points are to the same side or not. So, some constant time, constant number of additions, multiplications, subtractions, divisions which is in our language is order one time.

So, given any pair of lines segments; these are basically primitives that we take for granted. Whenever you are solving any geometric problems. So, that the good thing about the or simple thing about the previous problem maximal problem was that the only operations we are using was comparisons. We still in the in an old nice (( )) frame work that you know only comparison.

Here unfortunately, we are to solve these you know using division, multiplication and then there is a problem with you know overflow and floating points and so and so forth, but as I said we will not discuss those issues, the floating point issues, we just keep them a side so that we can done constant time. So, that is the basic primitive now of course, we are not given one pair, but we given n line segments and I want to report all the intersecting pairs.

So, what would be the most reveal approach to this (( )) fine. So, the first attempt would be to check all n choose two pairs just any (( )) and for to check every pair we know that it takes x constant time. We just discuss that that implies order n square. So, we have the first attempt to the simple straight forward approach should be order n square (( )) let us see.

Now, at end of the algorithm if we know that may be all the segments into actually (( )) there are n choose to intersection pairs, all segments can intersect, in a simple situation will well I mean sorry that is wrong way.

Now, you can have everything intersecting essentially you can always have cases easily construct cases. That is one possibility, the other case could be know none of them intersected you all disjoint. So, it can have entire structure median. So, you could have all the way from n choose to intersections to zero intersection.

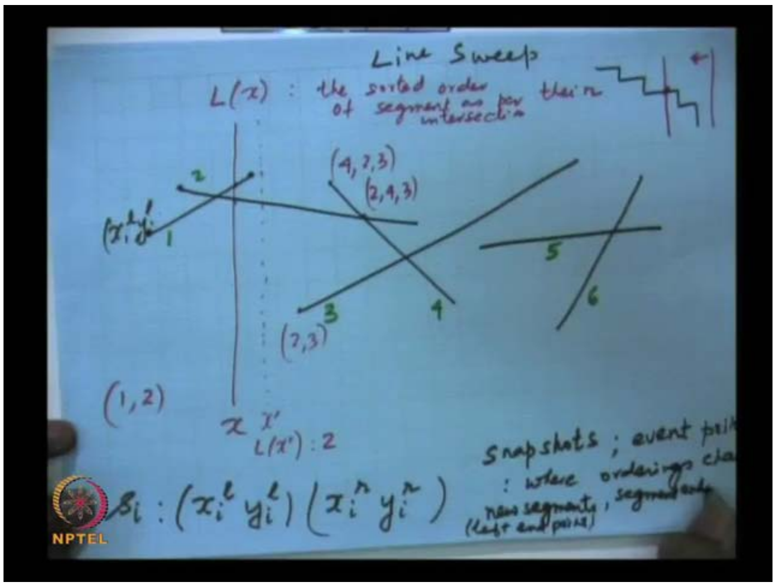
How do we argue that this simple nice algorithm whether it is a good or bad. Do you think you will you should you should be satisfied with this or there is some short coming? (( )) yeah. So, it is good if mostly pairs actually intersected because the output size is about n square, you cannot do anything better than n square. I mean you have to report on all the intersecting pairs. So, you bounded by the output size and therefore, n square algorithm is absolutely fine in a acceptable if you know there were many intersections.

But in the situation that they hardly any intersection, then what I will be looking for? I mean we just made some redundant checks for every pairs. So, there are no intersections to report in the ends certainly not bounded by the intersection size.

So, we should then aim for something that is much superior to n square, possibly linear time so, whatever. We again have this spectrum of the varying output sizes and a because we just seen in the prior example you know we can you know using some (( )) techniques exploit the fact that involve output can vary and we can make the algorithms of some of sensitive to the running time sensitive to the size the size of the output actual output. So, this again this problem is another one which kind of begs for this kind of approach that you know we should be very careful for this is very huge variation from order n to order n square.

From zero (( )) for number of intersection from zero to n square. So, running time cannot be better than order n. So, it could vary all the way from n to n square rather than just from n to n log n. After the previous example was only about n to n log n you know you may not be. So, much concerned about that. Here it is all the way from n to n square. So, then we should be really careful. So, how do we then address this? Let me do one thing; let me a switch over to a slightly larger people. So, I will switch over to this one and leave this aside, I can draw more easily here.

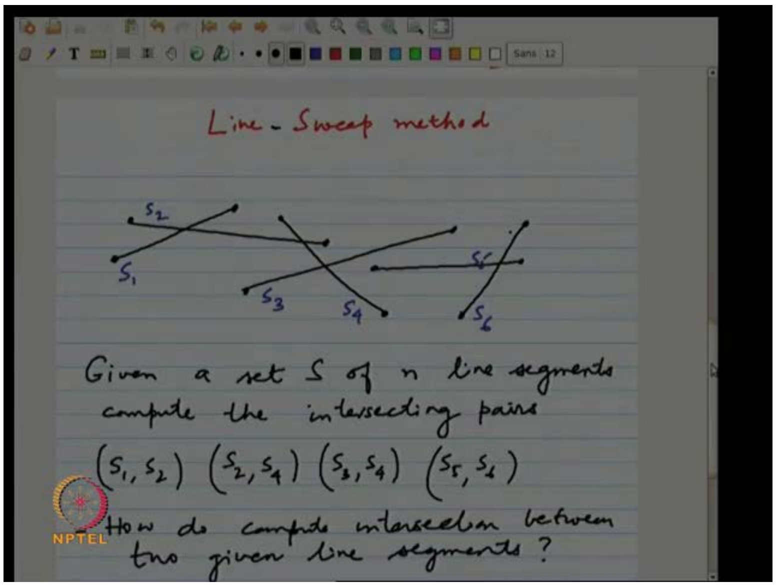
(Refer Slide Time: 30:21)



$s_i : (x_i^l \, y_i^l)\,(x_i^r \, y_i^r)$

So, I have these line segments. Its straight at least split on the at the same, you have something like this. Can we switch to this? So, I have told them will be difficult for me to actually you know use a small place to write can someone just want them. That again this is the (( )) watch let me try to draw this any way.

(Refer Slide Time: 31:16)


Line - Sweep method

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?

Fine. So, this is now visible. So, let me shift over here. The reason I want to shift over here is that know you can actually view my hands and everything unlike my tablet and I need that.

So, for the line sweep actually I am trying to basically explain how line sweep will be used in a problem like this and what is actual line sweep. So, somehow we need to order the points like we did in the case of two dimensional maxima that rather we start from the maximum x coordinate and move towards smallest x coordinate or you know some coordinate x axis here, you know we will actually start with lowest x coordinate and move towards the highest x coordinate of what the x coordinate of the end points of the given line segments. That is the information that we have. So, let us say that every segment s i is represented using let us say x i left y i left and x i right and y i right. So, this is basically the representation of the line segment s of i.

So, some segment will have the lowest left… Again let us assume that you know we do not have vertical line segments. Even if we did we locate you know coordinate axis make sure that it is in vertical. So, then we will we will start from the lowest x coordinate. So, in this case, line segment 1 has the lowest x coordinate. This is actually by this saying it is x 1 l and y 1l this is the n points this is this is smallest x coordinate. So, when I say we start from here, we actually hypothetically what I am going to do is actually I will take a vertical line starting to the left of the left most point and pretend that basically I am moving it and translating that vertical line from the left to the right.

It is like essentially I am taking snap shots as I move from left to right. So, at any point of time, my snap shot is the intersection of this vertical line with the line segments. If my vertical line is where this pen is, the snap shot essentially or this two line segments 2 and 3. 2 and 3 intersect this line vertical line nothing else does. So, similarly you know if you think about this lines sweep in higher dimensions, again it will be a cross section we can sweep let us say from x 2 again if you sweep along x axis the y and the z coordinates will be basically the cross section. So, at any point that be a plane instead of line. Here the lines sweep is actually a vertical line they if you have a vertical plane. So, whenever we do this sweeping, the information of the snap shot is the intersection of this line of the plane with the input. So, when we start from the left most to the left of the left most point, when we are not going to start from minus infinity because there is no input there and there no interesting information there. So, essentially we begin with there is some interesting input.

So, will start from the left most end point and move all the way to the right most end point. That is my that is the way line sweep will work. Now as we move this line of course, I am saying we are moving in a continuous fashion will not will not able to. So, actually simulate this thing you know because it will take infinite time continuous moving line will take finite time. So, we will only takes snap shots, you know at some junctions which will be interesting to us you know these are called event points.

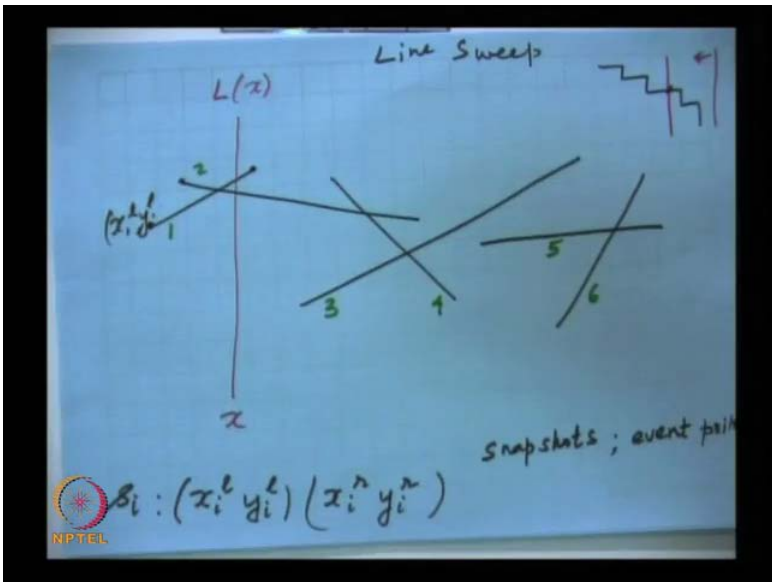
So, I am using the word snap shots and I will be interesting snap shot only at the even points. I will discuss I will tell you what the interesting events are. So, and that will depend on the contest to problems. In this the contest to this problem will be able to tell you what the event points are. As we sweep this the line and the plane whatever again not in not really set of simulate this continuous things, but only looking at what the changes happen at event points will be recording the changes of the event points. And using those recording only at the even points, somehow whatever information we accumulate will be enough for us to solve that particular problem. Certainly very high level a set description what lines sweep method is. And it is geometric method because we are talking about left most right most and intersection and so on so forth. It is really useful for geometric problems.

So, in the case of that maximal problem you know if you think about like a lines sweep where we swept from the highest y x coordinate to the lowest x y coordinate, what do you think is the snap shot here?

(( )) yes stairs, but what is the intersection here stairs? Exactly. So, if you if you just think about it, you know there was this stair and if you sweep a line at any point of time again is a vertical line, but just the begin from the left right to left is the left to right. So, when it intersects just intersect this stair case at one point that is nothing but a highest y coordinate to the right of that. So, at any at any junction the highest y coordinate is what is being is recorded and that is the snap shot.

Without even getting into the nomenclature of the line sweep, we actually divised the method of we use the method of that is essentially lines sweep method, it just a very elementary line sweep method because the problem is itself for the simple problem. But at this for this problem you know will it will slightly a more sophisticated.

(Refer Slide Time: 39:00)



So, let see what we need to do in the contest to this problem. So, when I hit, then left end points there is not much to do I mean just is a small line.

Only when you know we where there situation where there more than one line instead of crossing the vertical line, things start becoming interesting because we known that move then one segment in the input and therefore, there some potential possibility intersection.

So, what we known again I am saying, the snap shots are that that any junction, I will basically record the intersection of the vertical line with the line segments, at any given x axis. So, if I call this l the line as l. So, the intersection of the segments with the line at any coordinate x let me call it l's of x. Can we rename the end points or levels of x axis so that the each end point is I mean (( )).

Not necessarily you know. So, it depend on a what kind of actions you using. So, this is not clear, not it is not something that we can always do what we you saying that given a… So, this is this is real numbers right. So, how do you how do you even number you will be irrational right you just become makes life really very hard. So, let us not get into to that. Whatever we are given whether its real number you deal with real numbers, then the input is integer then its fine, but the input is real number. So, you cannot say that I can convert into always into finite. It can also be irrational numbers, it is not even simply multiplying something making it large enough. But you know this algorithm will not depend on any of those things, it is it is a dual kind of an algorithm and you know you will not really you know bother about the floating point nature of the inputs. So, l's of x is the intersection of this vertical line or the line sweep, the sweeping line with the line segments at coordinate x. In fact what I will do is I will use l's of x to denote the sorted order of segments as per their intersection. In other words, the l's of x at this point would look like for this thing it will look like. So, it first intersects 1 and then it intersects 2. So, let us say from top to bottom i keep it order whether its top to bottom top it does not really matter, but you just follow 1 convention here l's of x is going to be some line 2 sorry 1 comma 2, at this x you know l's of x is 1 comma 2 because its first intersects 1 and intersects 2. You can calculate the exact intersection point or you can simply keep record of the fact that you know is a line segment 1 and this line segment 2 and you have the description of the line segment 1 somewhere and line segment 2 somewhere, whether you want exact intersection on that is not a point. It is the ordering that we are going to use. So, at this point it is 1 2 as we keep moving, what happens is that at the right end point of line segment on the line segment ends.

So, here the as just as you cross this, then suppose, this I am calling some x prime. So, l's of x prime is only now 2. 1 is disappeared. Because 1 is gone I mean 1 is sweep fast 1 is gone. So, this only one line segment in l x prime, if this is x prime, x prime appears beyond line segment 1. You sweep further and then of course, at this junction as then again you know you have this line segment coming in. So, then it has to be revised and then it will become 2 appears over 3. So, it is going to be like 2 comma 3.

Then a then there is a problem no sorry, after that you know you encounter 4 and 4 actually is the top most intersection. So, then it should look like 4 comma 2 comma 3. That is how the ordering is changing. Till now the ordering of changing we are only construct constrain the end points, but then again the ordering is going to change here again intersection point. So, the moment we cross the intersection point, the ordering is going to change as 3 is going to remain below everything, but the ordering of 2 and 4 will swap. So, there will be 2 4 3. In fact, if I tell you that the ordering was 4 2 3 and ordering is 2 4 3, you can immediately tell me the there is an intersection line otherwise there is there cannot be a swap.

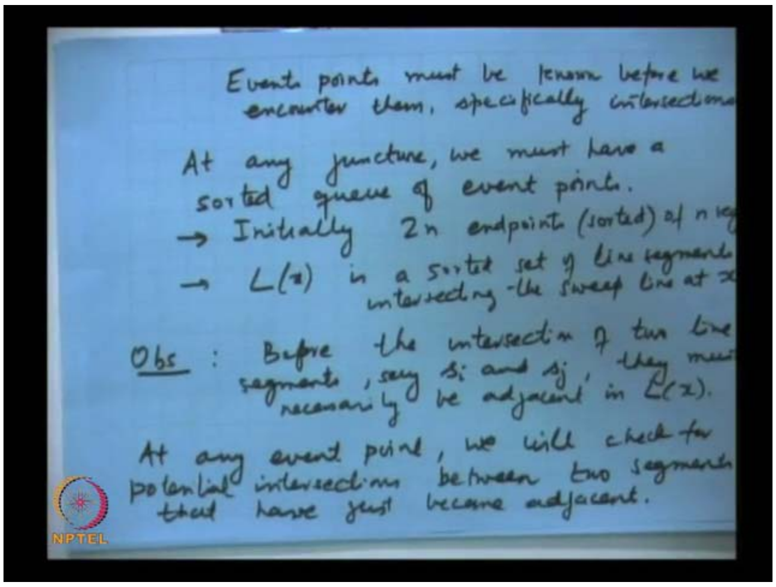
This is basically all that this line sweeps going to do. It is going to record you know this orderings as we sweep from left to right and the event points now you should be able to guess what the event points are. So, the event points are exactly those where orderings change. When ordering change means basically l of x changes, whether something disappears or not that is also change in ordering. So, we want to record everything about where the ordering are changed whether it is due to a insertion or the end of a line segments or bi-intersection. So, there are three kinds of event points.

So, event points of three kinds: new segments which is basically nothing but left end points, new segments and no segments ends, right end point and then the most important thing the intersection points for us, at least in the contest of this problems, intersection points or the events points are the device that we are looking at.

So, these are three event points. Now what is the difference between these event points? If you just characterize or all these all event points known to us first one first one. The third one that you are interested is in obviously, not known because we would not have a done this entire algorithm. We do the intersections we done.

So, somehow in the course of the algorithm when we are saying that essentially the job of these line sweeps is record all the changes at the event points. So, if we can actually perform this changes at all the event points you would also know all the intersections. It is not then intersections are known to us, but the intersection will have to be discovered as and when we encountered them.

(Refer Slide Time: 47:06)



Event 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 seg

→ $L(x)$ in 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, in other words what I am saying is... So, event points must be known before we encountered them, specifically intersections. The left end points and right end points are known to us and we want to also sweep them in order of the x of the x coordinate of these event points because we are sweeping in order from left to right. So, we need to somehow because we want to discover them before they get known. So, at any point of time, at any juncture, we must have a sorted queue of given points. So, initially of course, what is that we know? We know all the left and the right end points.

So, my initial event point queue is a sorted order of all the end points of the line segments. Initially 2 n end points of course, sorted in the right order sorted of n segments. So, this is my initial event queue or a event queue of for this line sweep and then I have to somehow as we sweep, we should be able to insert the intersection points, otherwise if you miss the intersection points then of course, the algorithm does not work. We have to precisely discover all the event points.

So, how do we discover all the event points is what this entire algorithm is about. This is a initial event queue and you know that l's of x is a and is sorted set of line segments intersecting the sweep line at x. So, these are the two information that we will keep; one is sorting of the event points along the x axis, the other is the is a snap shot, the snap shot which is a intersection of a line segments with the with a sweep line.

What we observed at least we made one observations if you if you go back to this, as we sweep across from left to right, any intersection that happens is going to switch the order of two line segments.

Not only that, you also observed the its going to switch the order of the two line segments, you just look at a the situation just before you hit the intersection of the intersection these two line segments. This will also happen between two adjacent line segments. Just look at the scenario be just before a hit intersection point and just subsequent to the intersection point. The two segments that intersect must become adjacent before the intersection. You are not talking about arbitrary line segments trying to find out which of them are actually switching the order. But we just the looking at only the adjacent pairs, we should be able to figure out at least compute whether or not they intersect. Essentially that what we are saying is before two line segments intersect which is the event point that you are interested in, observation before the intersection of two line segments say s i and s j, they must necessarily be the adjacent in l's of x.

So, this is the real question of observation and this is what is going to actually help us resigning an efficient algorithm. The reason why we are stuck with an n square bound or n cubes bound is because we have no idea which two segments can potentially intersect. But now if you look at this l's of x which is an order line segments, we do not have to look at every pair of line segments, but we only we look for adjacent pair of line segments to find intersections. Now adjacent pair of line segment does not necessarily mean they will intersect, but they are the only potential intersections.

So, at any point of time I do not have to check n square pairs, but I have to check only order n adjacencies. So, this is where the sharing is come from. Again this was the input as we sweep across you know 1 and 2 well it is it is simple case, but let say this particular intersection a 2 and 4 have become adjacent before they intersected and then look at again 4 and 3 . So, 4 and 3 of course, you know again became a adjacent before the intersecting before this intersection point, they had come together.

And when they have come together, that change must have accord in the previous event point because we are saying it is inductively that we will maintain this invariant that we are recording all the possible orderings where all the orderings of the event points. So, when they came together, that must have happened because of some previous event point.

So, if our algorithm was inductively right up to that point, then we should have recorded this and therefore, the moment that come together. So, this is what we are going to do. The moment we find two new line segments are adjacent we will check for potential intersection.

So, at any event point, we will check for potential intersections between two segments that have just become adjacent. So, the previously they were not adjacent, but they are not just adjacent because either we have inserted a new line segment or deleted an old segment or because of intersection also there could be new adjacent created. So, if you look at this example, as we sweep from left to right, we the moment that encountered 2, 1 and 2 adjacent. So, what I must do is find out if there is an intersection between 1 and 2. There may or may not be. So, 2 had ended here they wouldn't have been.

But of course, in this case it is. So, this we can check moment. So, then we go fast this and one disappears, but because of that there is no new adjacent you created in this case. Then I sweep I reach 3. The moment reach 3, I know that is a new adjacent between 2 and 3 I must check whether these intersection between 3 and 2 and actually there is no intersection, but I am checking that. Then I encountered 4. 4 have become adjacent to2. So, I must check for the intersection between 4 and 2 and actually they do intersect. The moment intersects I discover the intersection point and I actually insert into my event queue because I know there is an intersection and that is an event point and therefore, I will record that event point. So, after record that event point, that event point may have happen must (( )) this intersection could happen here also.

But whatever event queue I have at that point, I will insert that event point in the right order I have I must maintain the x the sorted x coordinates of the event points. The moment you discover there is an intersection point, I am going to insert into the event queue and then continue the line sweep. So, here actually this event point of intersection happens is a next event point. The moment there is an intersection, these two line segments have been swapped and therefore, I must check for adjacencies between this and this line segment that is 4 and 3. So, whenever there is these event points, there is some new adjacent created, but you see how many adjacent are created at any event points.

Yeah, some constant number. So, that what will be the essentially you know the way I mean that that will be the saving thing. So, for the entire thing you can kind of see that we are not going to the number the event points is how many. The total number of intersections plus 2 n, which is nice, that is the output size an every event point. There is only some constant amount of processing that we have to do at least we are checking for constant number adjacencies.

Right yes (( )) the changes. So, what is the change the change in l of x is only the ordering it is not the exact intersection point. So, the ordering can change only at the event points and if I am inductively doing everything correct up to a certain event point, then the next thing also have been done correctly (( )).

No, but I the question is the point here that we are going to discover every intersection point before it occurs because they have to become adjacent before the intersect and how did they became adjacent? Because of a pair event points (( )) the actual intersection is event point for me. So, my next top (( )) no no my next top is the, whatever is the event points, it could be an intersection also (( )) yes.

The total amount of processing that we are doing is about constant, a modulo data structures. If you we haven't talked about data structures yeah (( )) right (( )) right (( )) yes you yes you have a point and I will take that issue tomorrow. It can be easily handle I will take it up tomorrow. The question here is following suppose we actually had lines. So, you know vey very long lines. So, that they were actually this n square intersections n choose to intersections now I am always maintaining that i, but you know let I say that (( )), but a large number. So, you ask me how where are going to be begin from right, but then again it cannot the intersection point cannot happen before the left most end point and it cannot happen after the right most end point. So, I already have a finite sized block so whatever, begin to sweep. As soon as we reach some event point check whether the intersection (( )) is there is a potential intersection yeah (( )) that why I still first event point is first left most end into end point and the moment is sweep was that and reach the next line, I would have discover the first intersection point because it cannot occur before the 2..(( )). So, these are similarity conditions. That we can we can deal with by just rotating the axis. So, let us stop today here. We will carry on this discussion tomorrow.