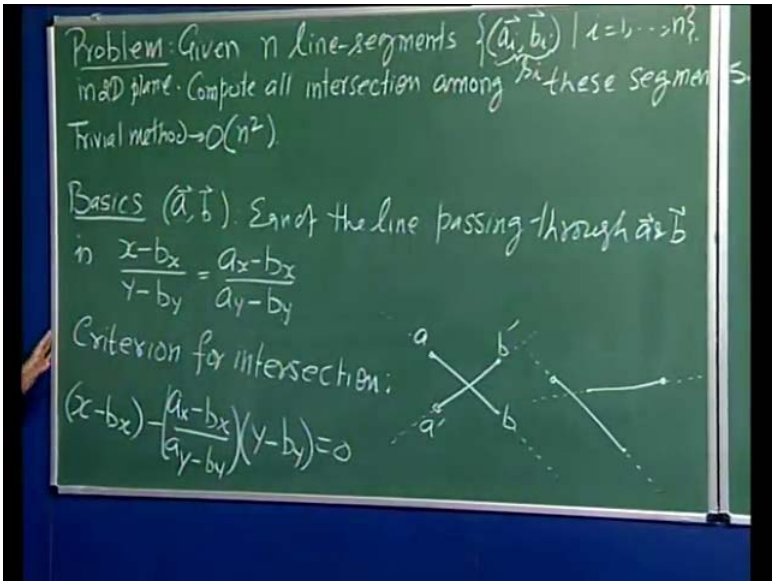
Computer Algorithms – 2
Prof. Dr. Shashank. K. Mehta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 30
Geometry II

Hello, today we will discuss a problem regarding intersection of line segments given in a two dimensional plane, let me describe the problem.

(Refer Slide Time: 00:26)



Problem is that given n line segments in terms of their n points, so let us say we have a i b i, in two dimensional plane, and our task is to compute all the intersections that can occur among these line segments. So, we may also call the segments from where two points as s i, for convenience sometimes, so compute all intersections among these segments. Now, the trivial method would be to take each pair of segments and find out if they intersect, and if, so compute the intersection point and output there. Well so the trivial method will cost order n square time, because we will have to check every pair and computing the intersection between two segments will take constant time.

Now, here we are going to see a method which will do much better than this, if there are

very few intersections, so let us actually begin with a few basics and then we will give some background how exactly we are going to proceed. So, a few basics here, then let us suppose we have a line segment a comma b, so the two end points are a and b. Then the equation, of the line passing through a and b is x minus b x divided by y minus b y is equal to a x minus b x over a y minus b y.

Now, what is the criterion for checking whether 2 segments intersect, so criterion for intersection. Now, obviously, one way is to compute these two equations for 2 line segments, compute the intersection point of the two lines and to check whether that intersection point is within the span of the segments both the segments. Alternatively, we can do the following, let suppose we have two segments take two cases.

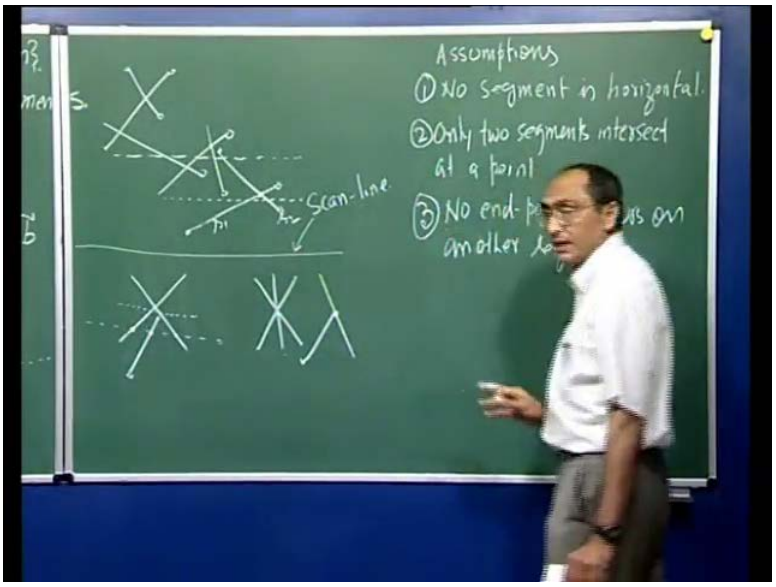
Let us consider this and the other 1 say this one, what you notice here is that let us say this is a a b and a prime b prime, what you notice is that the line corresponding to this segment, separates the two end points of the other segment. So, if you check you take the equation of this line and insert the coordinates of a prime, if the sign of the expression is positive, then the sign for this expression will be negative.

So, what would that expression be, so expression we can write down is x minus b x minus a x minus b x over a y minus b y into y minus b y equal to 0. Any point x y that satisfies this equation is on the line, but the points on this side if they are positive, if inserting those values of any point here is positive then inserting the values on points on this side will be negative. So, we can actually in constant time decide whether 2 points of this segment are on either side of the line, similarly the line passing through the second segment splits these two end points.

So, what we notice is that if that is happening that is each line is splitting the other segment into two parts, then the two segments are intersecting. As against this case you notice that well indeed this line is separating these two end points, but this line is not separating the two end points of this segment. So, there is a simple test without computing the intersection part. Now, let us take a look at the basic idea that we will use

in devising an algorithm, which we will actually our algorithm will depend on the number of intersection points that are present in the picture.

(Refer Slide Time: 07:48)



So, how do we actually go about it, let us take a picture, suppose we have some line segments and they have some intersections here, these what we wish to do is to take a horizontal line and pass through this picture from bottom upwards. And focus on those segments which intersect our scan line, so this is our scan line what we should notice is that any time two segments intersect.

Then before the intersection happens let us say somewhere here we notice that the two line segments are immediate neighbors of each other, among those lines which are intersecting the scan line. So, if at this point the only two lines, which are intersecting are these two line segments and they are of course, neighbors of each other. Let us say at a later stage say the line segment is at this stage, we notice that these two are adjacent to each other and they are going to intersect later on.

In general let us suppose we have a situation such as this and line segment the scan line is here, what we find is that these three segments are intersected by this line. And they are occurring in this order 1 2 and 3, but after this point when the line passes through the upper end of this segment, these two segments become neighbor of each other before

they intersect. In other words we can predict whether these two are going to intersect if you only we focus on who all are neighbors of each other.

So, we will only try to do the following, we will keep a stock of the fact that these segments are intersecting currently, and who all are adjacent to each other. We will check if these two are adjacent we will check whether they intersect they may not in this case they do not, we will check whether these two intersect in this case they do not. At the end here we have to drop this out because it is seizes to intersect the scan line, we will remove this from consideration after we move on here.

Then we find now suddenly these two have become neighbors, we will check that, and in this case we do find that they intersect. So, the scan line algorithm is as follows, we will start at y equal to infinity minus infinity the horizontal line at infinity. And then we will bring it to the lowest point among all these lines, let us say this one at this point, we find that this line intersects our scan line. Our goal is to determine the status of those segments which intersect our line, we will keep two things in our record we will find out who all which all inter segments intersect this.

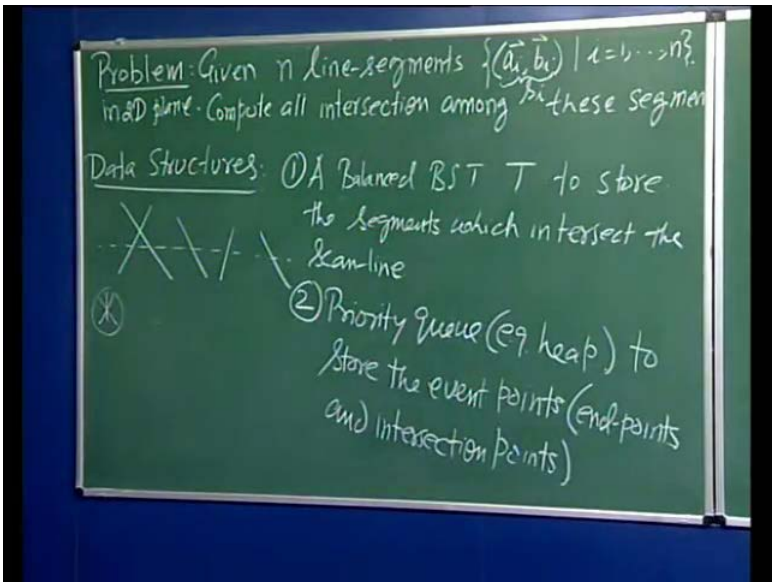
And in which order we will keep these two things in mind in our information, this information can only change at very special points, when we hit a lower end of a line segment. Then a new line segment enters into this list of line segments intersecting the scan line. When, we hit the upper end a segment drops out, when we hit an intersection. Then the order in which they were intersecting switches, that is to say when the segment is the line here then say s 1 and s 2 are intersecting in the following fashion s 1 is to the left of s 2, where we go past the intersection, then s 2 is on left of s 1.

So, if we keep updating this information, we will know all potential neighbors that will occur, and from that we will extract information about the intersection. In geometry there are situations which are often hard to sort out, we have to give a special consideration. So, those are the generate cases which one should avoid in the first discussion I am going to avoid some of those special situations in this picture. While, discussing this algorithm

later on one can try and fix that only adds a little bit of complexity in the algorithm, but not the time complexity of the (( )).

We are going to make certain assumptions, our assumptions are first of all we will assume that no segment is horizontal, because in that case the two end points will occur simultaneously. We will have little bit difficulty in taking care of, although it is not that hard. Second assumption is that only two segments intersect at a point, so we will avoid these situations any number can actually intersect. We also avoid anything like this, that is to say the end point of one line touches another line. So, no end point occurs on another segment, so no touching no multi intersections no horizontal line.

(Refer Slide Time: 16:15)



So, now let us talk about what data structures we will need, in this algorithm there are two key things that we have to take care of in terms of information storage. One is we have to store the status of intersection, what we have just seen is that if this is the scan line, I got to know which segments are intersecting the scan line. And what is the order in which they intersect, so I am going to store this information, so first will be a balanced binary search stream T to store the segments, which intersect the scan line.

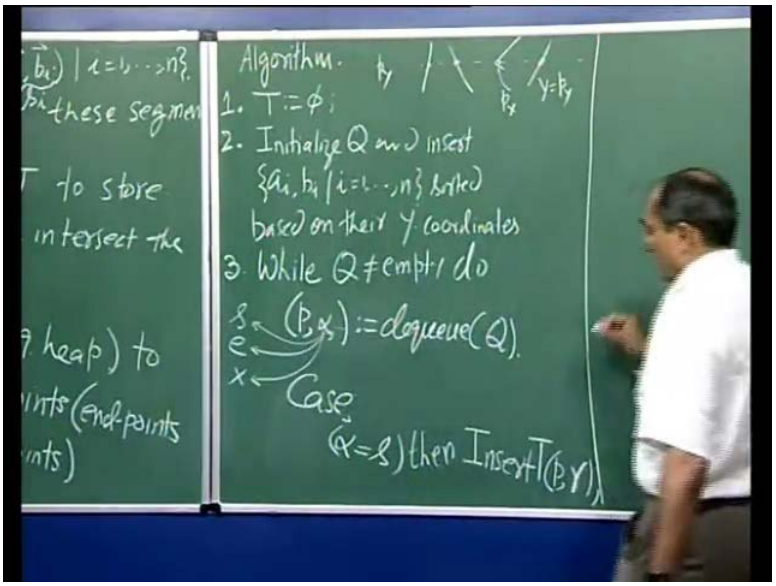
Since, we have to keep the information of the order in which they intersect, I am using a binary search stream. Since, we are going to work on it for inserting and deleting the

segment we want to do it efficiently, I am using a binary search stream. The second structure is called event queue, now notice that the status of this intersection only changes at lower end point, upper end point and intersections. These are called the events and I am going to store these events in a priority queue, we are going to scan from bottom up.

So, the lowest y value terms first has the highest priority, we have to process the event after event notice that there is no change happens outside these events. So, there is no reason to look at this possible changes in t, so we will keep a priority queue may be a heap to store the event points, which are the end points and intersection points. Now, it is important to note that, when we start we have all the end points, we note all the end points all the two end points. And we do not know a single intersection point, this is precisely what we want to compute, but in order to progress.

We need these as well because they are important events where the changes occur, but as we saw that the we can make a forecast about that happening, because we notice that we can check adjacent segments, and check whether their going to intersect. If two segments do not occur adjacent to each other ever, then they will never intersect. Under our assumption there we do not have multiple intersection, we do not allow this these can be handled separately. So, we will be able to forecast all possible intersections, hence before that even happens we will know it we will determine it and enter into this Q. So, that we can process it when it is turn comes, so we have these two data structures, the low measure data structures other than this required.

(Refer Slide Time: 21:15)



So, now let us first of all come up with a overall picture of the algorithm, to begin with have to initialize both the structures. So, we set T to be empty, initially the binary search stream is empty, because our scan line is at y equal to minus infinity start intersecting vertically. The second thing is insert initialize Q and insert a i and b i sorted based on their y coordinates, once again assume that no two events have the same y coordinates for simplicity.

Now, we are ready with the two structures and from this point onwards, we are going to go through these events one at a time, update our structures both T and Q as required. As a bi product we will detect new intersections, that we will output. So, while Q is not empty, we do this look p is a dequeue pick the highest priority event, p is a point note that this is a point which can be either 1 of the end points of some segment or it is an intersection point.

Later on we will have to fix this slightly, because we have to have the information what kind of event this is, whether it is an intersection point or start point or end point that is a lower end point or upper end point. So, in fact, maybe at this point itself i will express what even this is, so will say p and say alpha. So, alpha we are assuming is one of the 3 letters, that is to say s or e or x, indicating that this is the start point of a segment, end

point of a segment or an intersection event. In addition to that I may have some other information which actually gives us the segments involved in this.

Student: Sir in into where we end an initializing queue, we need to store alpha also there.

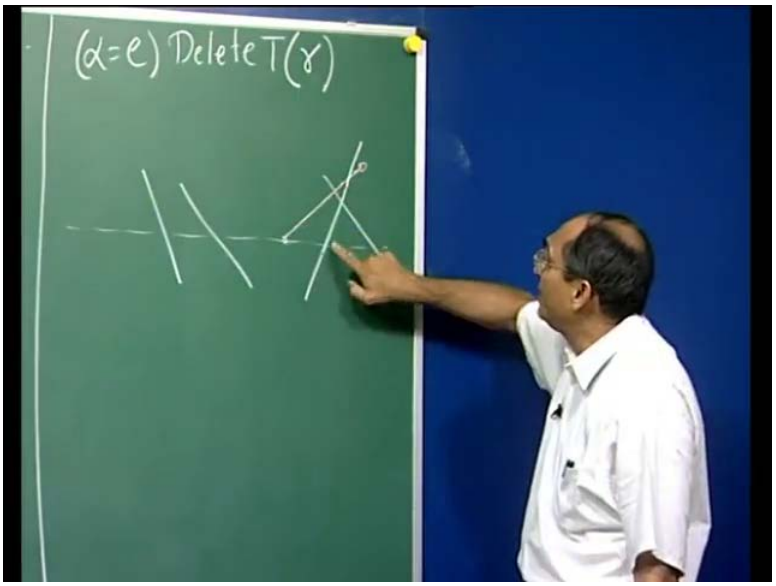
That is right, so I will actually fix that assignment right, the event that we are going to insert will be not just the points, but we will have some more information. So, as and when we realize we will actually we will put things. Now, based on what kind of event this is I will have to handle them separately. So, let us now consider each case, so if or rather we can actually write down a case statement that might looks likely easier to, so we can say a case.

And the first case would be when alpha is s, so it turns out that the event is a start point of a line segment, in this case, so this is the condition then if alpha is s. What we have to do is to insert, so here is the scan line and there are already certain segments sitting here turns out the new segment that we have just discovered is this. We just hit upon the lower end of first segment, we have to insert this between these 2, the coordinates this is the point p, so we have both x and y coordinates of p.

What we have to do is determine the x value of each of these segments at y equal to p y, where this is the y value here this is p y. At this point what are the x values we know that with respect to x value these are already sorted, so when we search where does p x go where is the position of p x in these values we will know where does this new segment fall between these 2.

So, we will say insert and we will actually write down small routines later on simply say insert in t, the point is p and we have a segment s, I should use something else. Let us call it segment r, so in this case I will have my event as p s r this is the starting point p is a starting point of segment r, so we will know what to insert.

(Refer Slide Time: 29:03)

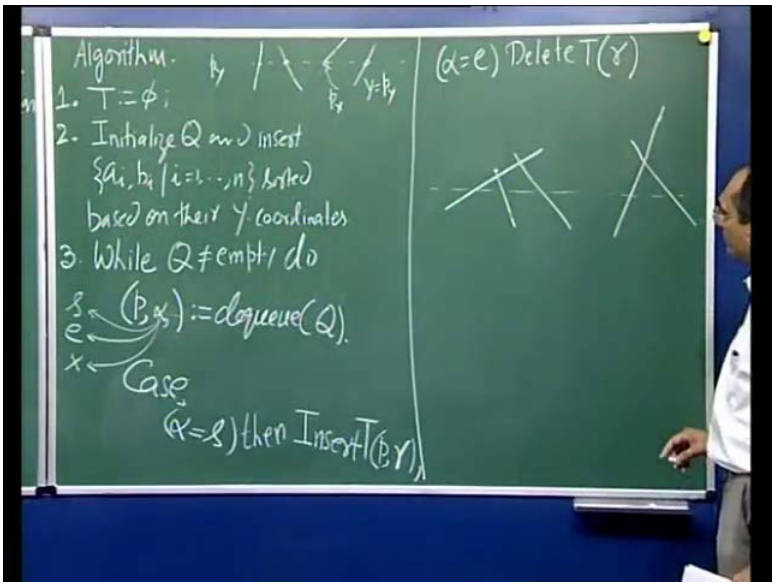


Second case is when alpha is the end point, we will delete r there we do not have to worry about the position, we just have to delete the segment r, once again our event will be p e r point p is the upper end point of segment r. Now, let us get back to both of these again, what else do, I have to do in these besides of course, removing or inserting or removing the segment. There is one important task that we have to account, notice that if this was the segment.

We can just rewrite this and the new segment is this, so we have the new segment. When, we are doing insertion our job is to check with two neighbors, whether this intersects them or not or it is possible that this segment was longer and it is indeed intersecting this segment. It is not intersecting this segment, but since a new neighborhood has been created, we will check this with this and we will check this with this.

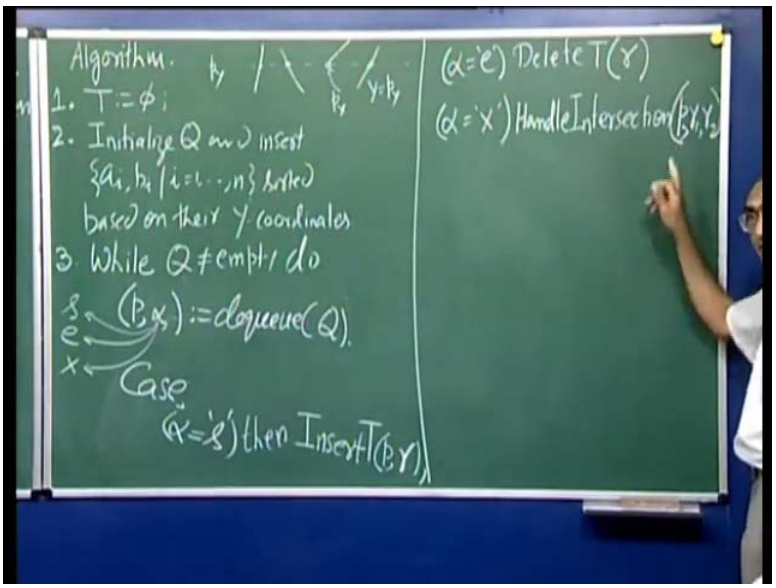
If at all we determine intersection happens then a new event will be created, event will go into the event Q cube, and that event will be the intersection event. We will put the coordinate of this point, we will put the letter x to indicate it is an intersection, and we will put the relevant segments namely this segment and this segment into the event.

(Refer Slide Time: 31:24)



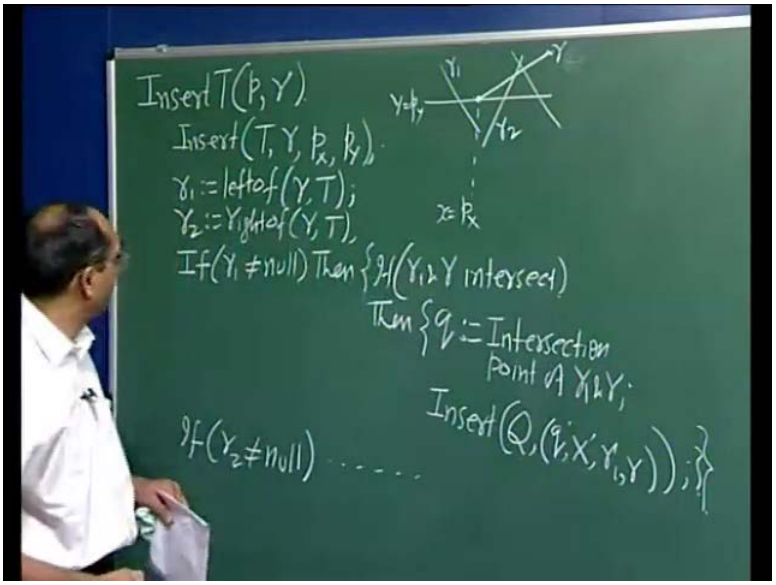
Now, similarly in case of deletion, let us for the sake of giving an example, let us suppose we are deleting this line segment at some stage. Well, it is unfortunately let us say here is the end point and we are about to delete this segment, the moment we delete this we figure out that these two have become neighbors. And as soon as we find that these two are neighbors, we will check whether they intersect and if they do we are again going to create and intersection event and insert into the event Q cube.

(Refer Slide Time: 32:25)



So, these are the two additional tasks associated with these two steps, the third case is alpha is x, so in this case I will simply say handle intersection p comma r 1 comma r 2. These two segments are intersecting at point p and we are supposed to update the data structure based on that, so we are essentially done this is all.

(Refer Slide Time: 33:15)



So, let us now find out exactly, what are these routines these things are routines, so first one is insert t p comma r. So, as we said first thing we have to do is insert in the data structure t segment r with respect to the coordinate p x. This is the value we have to set and p y is the y value at which we have to evaluate the x coordinates of every segment in T and based on that we have to determine the position of the x coordinate position of the segment r.

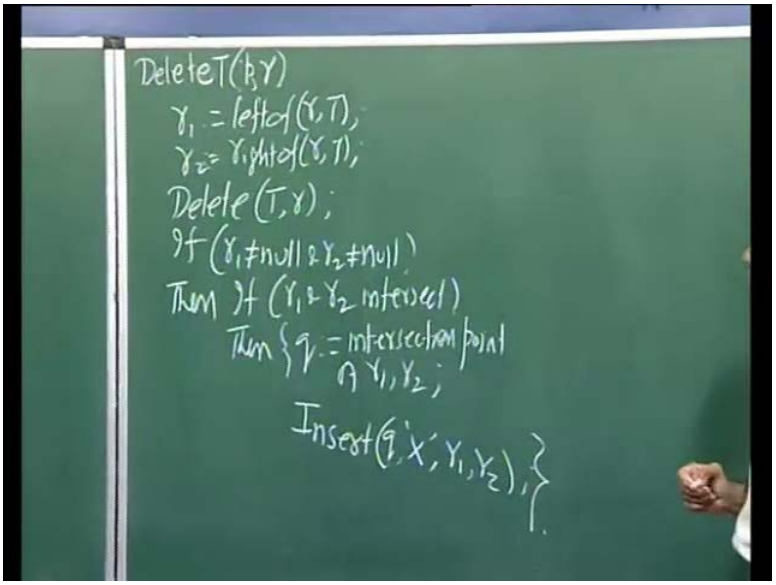
So, in our example suppose we happen to have this and the new segment is this is y equal to p y and this is p x, we will plug in value p y into each of these lines find out what is the x value of each of these segments. And based on that we will determine where falls the value of sorry p x here and accordingly, we will insert into the binary T and then we will find out s 1 or I will call it r 1 now, r 1 is the left of r in T.

So, if there is neighbor to the left of r in T, that we written as in goes nil and we will similarly find out the right neighbor of r in T. Then check whether r 1 intersects, this is

my r 1 this is r 2 sorry this is r 2 and this is r the new segment that we are incorporated. We will check whether this and this intersect we will check this and this intersect, if r 1 is not null then if r 1 and r intersect. Then insert in Q the event p sorry the intersection point of the two, intersect, so maybe we will need to r 1 and r intersect, then let us call Q as the intersection point of r 1 and r.

And then we are going to insert in Q the event namely Q comma x because it is an intersection point. And we will keep the two line segments here in the order since we are having r 1 to the left of r all right. Similarly, if r 2 is not null, we will do exactly similar thing we will insert in this case at Q an intersection event r 2 because r is to the left of r 2.

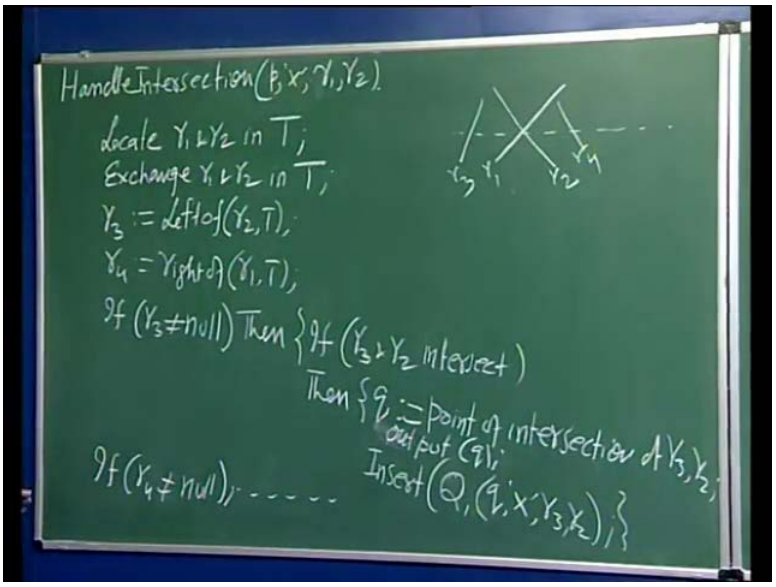
(Refer Slide Time: 38:48)



Now, deletion or we say delete T event p comma r or just r makes no difference, in this case delete from T the segment r. Now, the first thing we have to determine the neighbors, so I will first do that this time I will check r 1 as the left neighbor left of r in t r 2, would be the right neighbor. We will delete from T segment r, and now if both them are actual segments then we will check whether they intersect. If r 1 is not null and r 2 is not null, then if r 1 and r 2 intersect then Q is the intersection point of r 1 and r 2. Then we are going to insert Q x r 1 r 2 notice that when we insert an event, we look at the y value of

this point because that is what decides the priority the lowest y value has the highest priority, so this is how we will perform the delete.

(Refer Slide Time: 41:28)



So, the last one is handle intersection p comma x comma r 1 r 2, so we know that r 1 and r 2 are intersecting at point p r 1 is to the left of r 2, in our data structure let us just in the in the geometric sense we have this situation. We have r 1 we have r 2 here this is an intersection point, so both these line segments extend beyond this point all, we have to do we are sure that r 1 and r 2 are adjacent in T, because T carries all the intersecting segments.

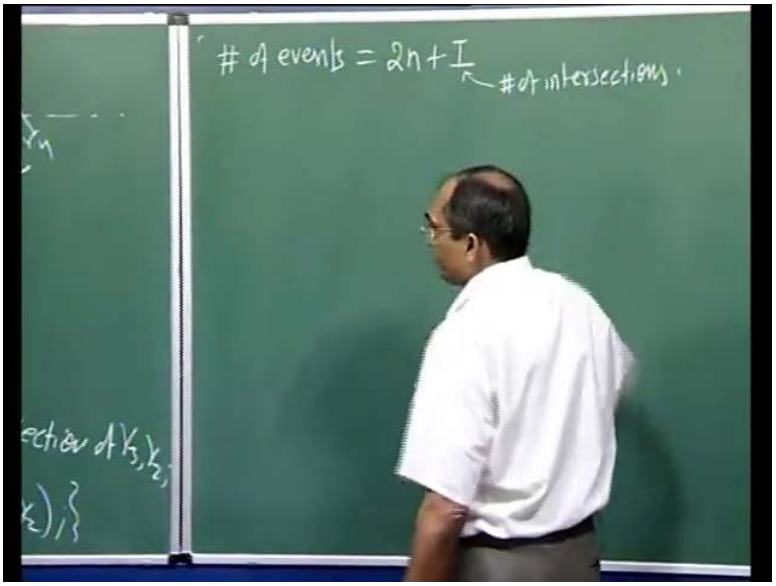
They must be adjacent to each other, all we have to do is swap their positions in T then that will reflect the position after this point, so we have to do that. Another thing we are going to do is if there are line segments here r 3 and r 4, then we will check whether r 2 intersects with r 3 we will check r 1 intersects with r 4. So, I will first locate r 1 and r 2 in T basically we do not have to make any structural changes in T only swap the values r 1 and r 2 in the respective nodes. So, then we will exchange r 1 and r 2 in T once we have swapped them we have the correct values in T. Then we are going to find r 3 left of r 2 in T r 4 in the right of r 1 in t.

Student: Shall we do this after the exchange.

So, we have done the exchange, so you know r 1 and r 2 are, so now, I am checking whether to the left of r 2 is there r 3 and to the right of r 1 is the r 4. We could have done the similar thing before exchange, but then I would have checked what is to the left of r 1 and right of r 2 which is similar.

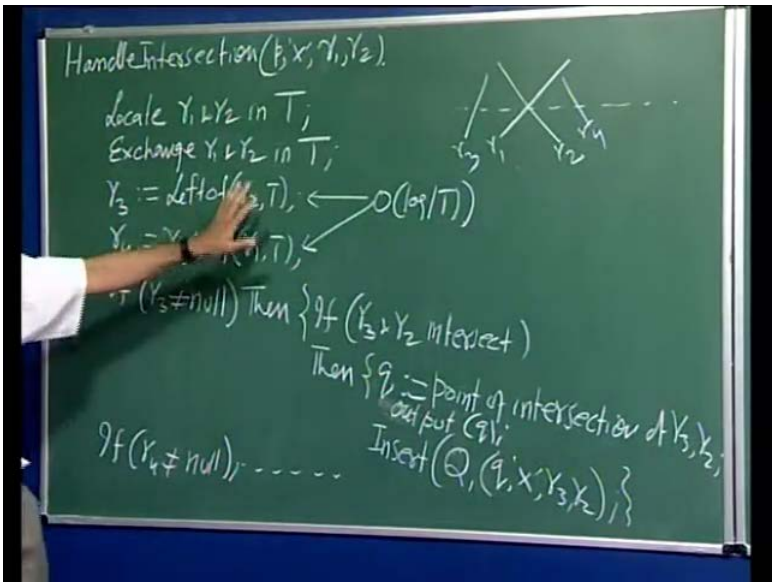
Now, we will check as we did in the earlier cases, if r 3 is not null, then we are going to check whether r three and r 2 intersect. Then find the Q, the intersection point of intersection of r 3 and r 2, and then insert in Q the event namely Q comma x comma r 3 r 2. So, this one thing that I forgot to point out after discovering these intersections, we should report in all the cases report or output Q that we will do after every discovery of the intersection point. Now, similarly we will do if r 4 is not null exactly, the same thing now that takes care of the complete algorithm, now let us talk about the complexity of this process notice that the entire algorithm is nothing but after initializing the different structures is nothing but processing events.

(Refer Slide Time: 46:58)



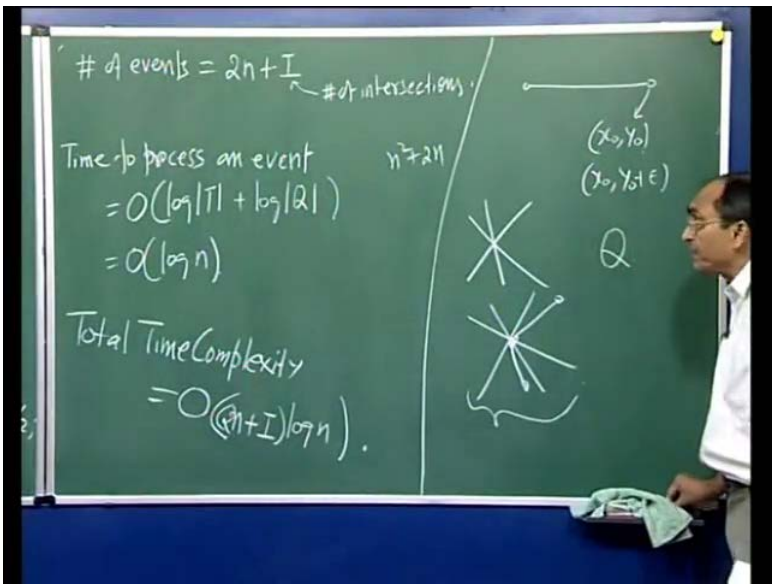
So, we are going to process as many events as number of events is precisely 2 n plus i where i is the number of intersections number of intersections.

(Refer Slide Time: 47:20)



In case of intersection which we have right here, all these are constant time operations, so these are login time operations. So, these take order log of T time because we may have some number of is at most n elements in T. And value locate search insert delete everything takes this much time because these are balanced values search streams. Over here whether they intersect takes constant time, computing intersection point is constant time, but insertion again we have log of mod of Q time involved here.

(Refer Slide Time: 48:22)



So, the time to process and event now, the cost associated with the other events is also a similar is order log of mod of Q plus log of mod of Q. Now, we know that Q can have at most n square plus 2 n events, you know where actually n what n square, but n c 2 which is order n square, but log of that is still log n. And this also cannot be more than log n, because that cannot be n segments, so this is the nothing but order log n, log of n square is 2 log n.

So, makes no difference and the total cost is, so total time complexity is nothing but order n plus i or 2 n plus i log finally, a few remarks about the degenerate cases, which may have ignored here. First thing is what if you have a horizontal line segment, one trick that you can always play in these cases is to modify this point. And if it is coordinates are x naught, y naught, then you change it to x naught and y naught plus epsilon.

A very small number such that the relative positions of the line segments never change, once you do that now you have different y values of these. So, this will occur first then this will occur as we go from bottom upwards, the second thing is what if you have multiple intersections. Now, I what we notice is that in our method, we will notice these two intersect here, we will also notice these two intersect here. We will never notice these two intersecting, but the two will at the same point, when we will insert that point into Q.

When we are searching for the position of that point we will find exactly this point provided we have accurate values of Q of the intersection point, and we will conclude that all 3 are intersecting at the same point. So, in principle we can do that, but the data structure needs to be modified slightly, when we have multiple intersections with respect to each of this intersection point we will have to keep.

Now, in more general situation could be, so this is the point which is end point of some line segments, which is also intersection point of several segments. We will have to keep a list of these line segments, which are below the intersection point, and the list of line

segments which are above the intersection point. And then we will have to appropriately delete them, and insert these in that order yet the time complexity, does not change if we suitably modify the structures the complexity, remains the same. So, this is where we end today’s discussion and we will resume, we will continue to talk about geometric algorithms for the next lecture as well.