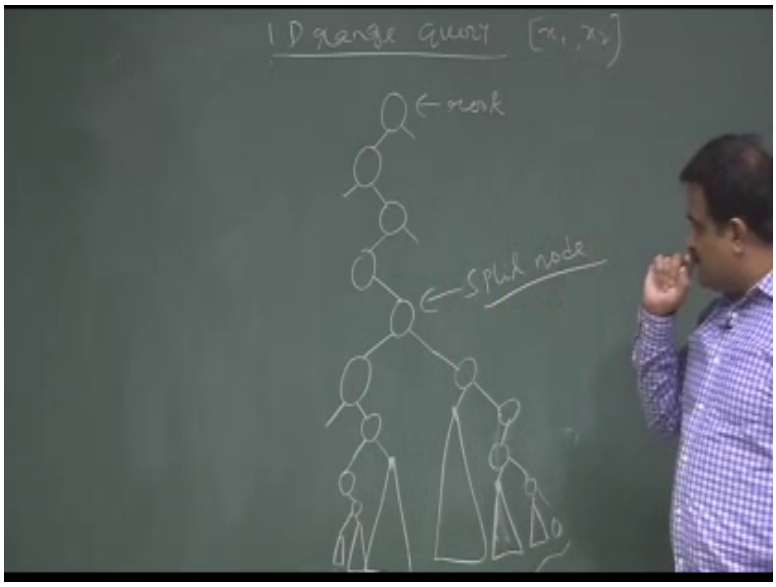
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 35
Computational Geometry (Contd.)

So, we are talking about the 1 D range search tree, 1 D range search. So, this is the continuation of the last class.

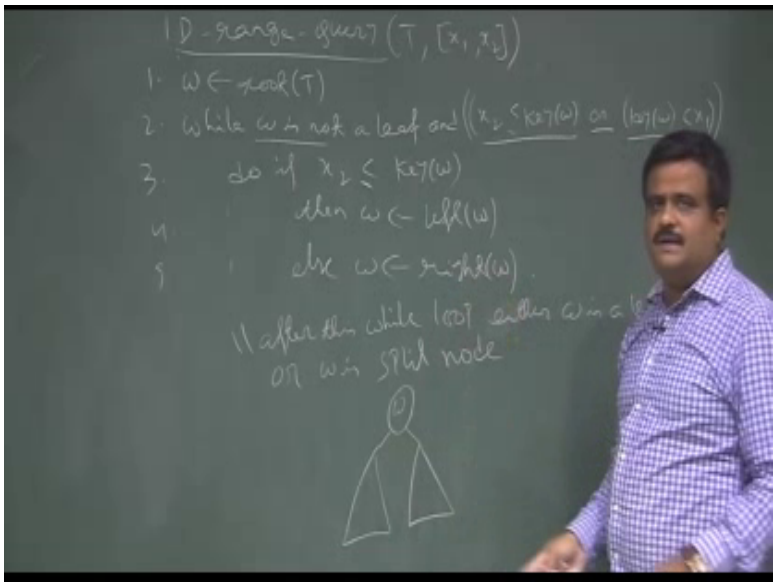
(Refer Slide Time: 00:31)



So, basically, so the idea is to start with the root with, so we have the 1 D range search tree. So, we start with the root and then we go until we continue until we reach to a split node.

So, split node means there is something interesting in the left path and there is something interesting in the right path; that means, there is some node which are in the range and there are some nodes in the right path also which are in the range this is called split node. And then after the split node we will we will keep on continuing the left and right both sides. So, in the left path, we again check, if there is nobody interested in the left path we go to the right and if we go to the left then everybody will be interested in the right path. So, we have to output this root this whole subtree, like this we continue. So, let us write the pseudocode for this. So, let us write the pseudocode for this.

(Refer Slide Time: 01:40)



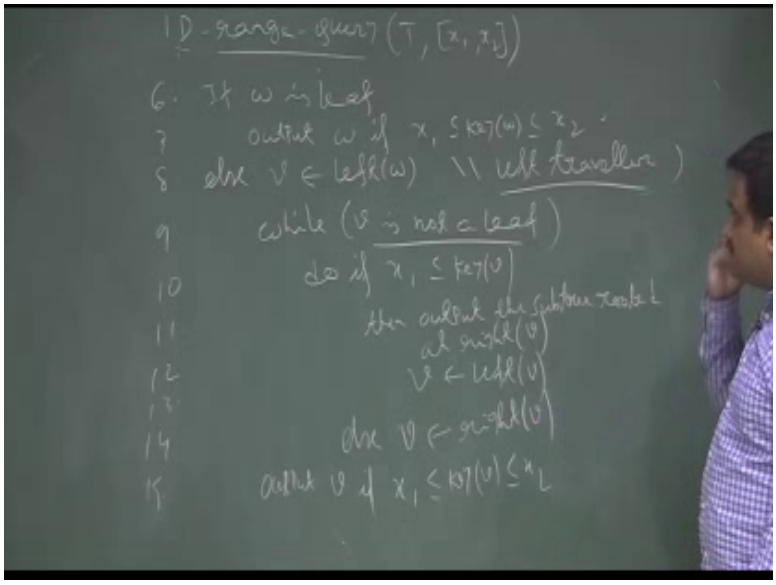
So, this is basically 1 D range search 1 D range query. So, the input is we have the tree the range search tree which we this is a static data structure which we form by the given n points and a interval query. So, this x 1 and x 2 is the interval. So, let us just write the pseudocode. So, it is w the root of the tree w, then we will continue until we reach to a split node then while w is not a leaf and x 2 is less than key of w or this or key of w is less than x 1.

So, either x is leaf if it is leaf we stop otherwise, otherwise we check whether this is a split node or not if it is a split if it is not a split this is by checking. So, that means, if, this is the root, this is the root this key of w and we are looking for x 1 x 2 there we are looking for the points which are overlapping with x 1 x 2. Now if x 2 is less than key of w; that means our interval is completely in the left side then we go to the left part.

Otherwise if x 1 is greater than key of w then our interval will be on the right side, so we have to search for this part we have to search for this part either. So, if it is so; that means, it is not a split node this is the checking that it is not a split node; that means, there is nobody is. So, either way you have to go so that is the idea. So, if this then we have to go either go left or right depending on this whichever is the matching. So, this is the this is the 3 do if x if x 2 is less than equal to key of w then we go to the left part then w is left of w otherwise 4; 5; otherwise else do if this is else w is right of w.

So, this will continue until it reach to a leaf we stop otherwise it reach to a split node. So, after this while loop we reach to a split node. So, then after this while loop. So, this is the observation, after this while loop, so either w is a leaf node w is a leaf or w is a split node. So, if it is a split node then we know. So, this is our w if it is a split node; that means, we know something is interesting here something is interesting here. So, then we will do travel both the way so, but in the algorithm we will we will show the right only 1 part and left part is the similar. So, let us just write this. So, this is 5 then we have a 6 over here and be deleting this we got the split node set. So, this will be 6, after that 5 ok.

(Refer Slide Time: 05:49)



So; that means, w is a split node now if w is a leaf then we will check the key of w with this and if it is matching if w is a leaf then output the leaf output the w if x 1 is than equal to key of w is less than equal to x 2. Otherwise if w is not a leaf else; that means, w is a split node which not a leaf w it has some child left child or right child or both it may have then. So, suppose we then we go to the then we take its v is basically left of w. So, this is the left traversal.

Similarly, we have to do the right traversal also. So, we will know it is not showing this then we continue while w is not a leaf what we do, do if x 1 is less than equal to key of v sorry v is not a leaf if x 1 is less than equal to key of v then. So, if x 1 is less than less than equal to key of v; that means, this is our v. So, we are looking for x 1 x 2 now we

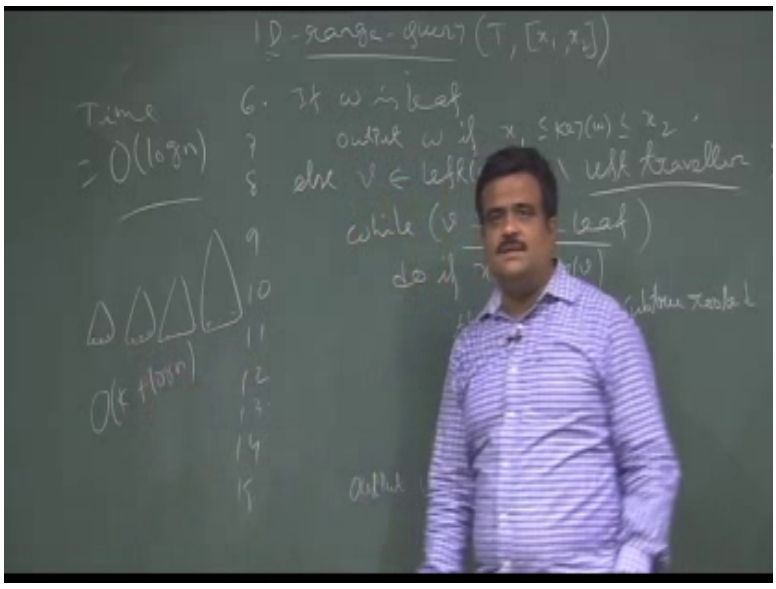
know that something is interesting over here now if 1 is less than equal to key of v then we have to go to the left part and we will output on the right part then that is the idea.

X 1 is then output the subtree rooted at rooted at right of v whole subtree rooted at right of v and we go for left again left of v else we go for right of v until we reach to the leaf node or we reach to the situation where we have something interesting in the left then we have to output on the right subtree. So, that is the idea. So, finally, once we reach to the leaf then we check. So, this is 9 10 11 12 13 14 15 maybe.

So, finally, this means will be released to a leaf then we check the output see if x 1 is less than equal to key of v is less than equal to x 2. So, this is the left traversal similarly you have to travel in the right part of the subtree. So, right traversal you need to do. So, now, we have to analyze this. So, what is the time complexity of this? So, this is basically a, basically our tree is balanced. So, there are n nodes and it is basically log n height. So, basically what we are doing we are traversing the tree, for each, and we are outputting the subtree.

So, in any case we are just start from the root and we are going to the leaf. So, it is basically taking log n time.

(Refer Slide Time: 10:04)

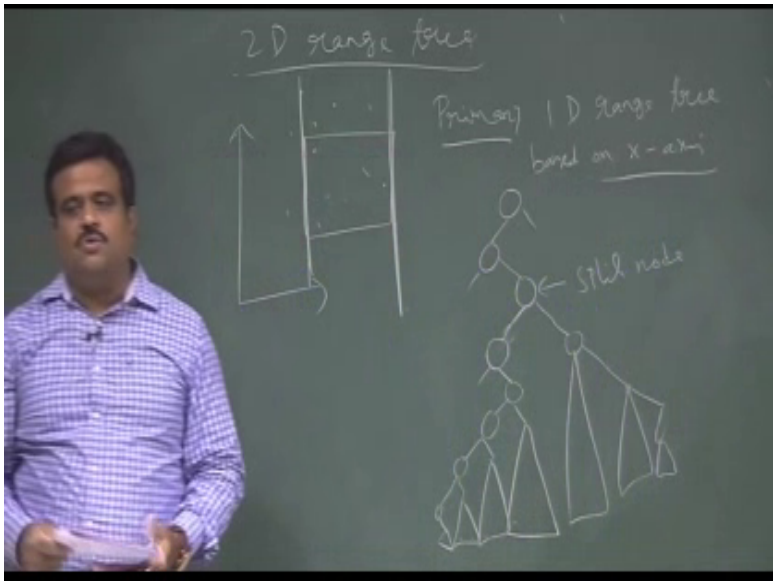


Time is log n for this traversal to go from the root to the leaf and for each time. So, now, after getting the output after getting the tree we need to, basically we are getting this

trees like this. So, after getting these trees these are the answer these leafs are the answer. So, to get these leafs we need to spend order of k again. So, this is basically the time complexity for this is requiring output; that means, we need to get these leafs. So, that is basically k plus log n is the time complexity for this. Now, what is the space how many space. So, to store this we need to have order of n is the space and to form that tree I we need have order of n log n that is the space complexity and this is the time complexity here.

Now the question is this is a very nice data structure the question is the advantages we can extend this for 2 dimensional or higher dimensional. So, that is the idea. So, that is the 2 D tree, 2 D range tree.

(Refer Slide Time: 11:25)



So, the question is how we can extend this for 2 D. So, basically now we have seen the 1 D range tree now how we can extend this for 2 D. So, what is the way? So, 2 D range search means what we do. So, we have a 2 dimensional points and 2 D range search means we have a rectangle like this we have a rectangular box a rectangle.

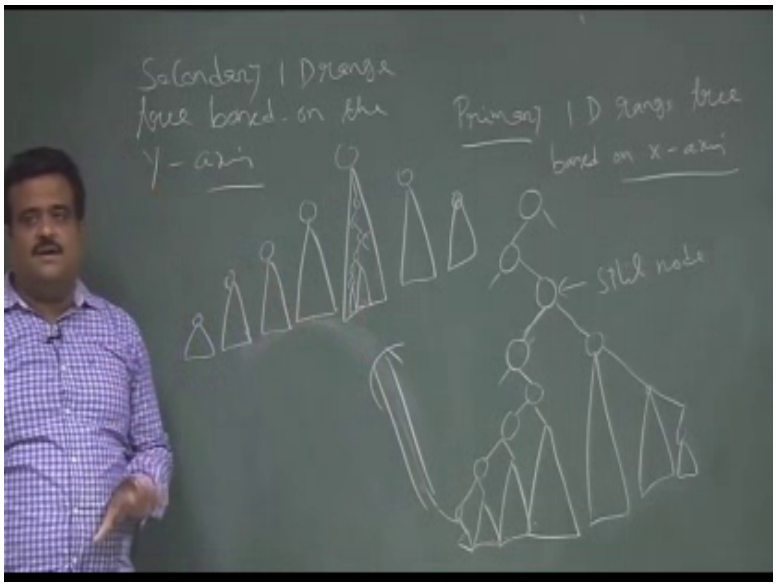
So, what we do? We first take a 1 D range search. So, basically the idea is we store the points. So, we store the point in 1 D range search tree. So, that is the primary tree. So, primary 1 D range tree is the primary tree based on based on x axis value. Now if you do that then and if we give the interval query then we will get the; so on that we just do this query. So, we start with the root, this is our 1 D range tree based on the x value x

coordinate. So, we have this root we start with the root we go left then we go right like this. So, until we reach here say split node once we will (Refer Time: 13:06) split node.

So, once we reach to a split node then again we continue. So, it we go right then if we go left then this all these are relevant then again from here like this like this like this. So, similarly we will go here. So, like this, like this, like this. So, basically these are all outputs. So, these are our points whose x axis are which are in this strip where x axis are on the range x 1 x 2, but we want y axis to be on the range x 1 x 2. So, these are the points whose x axis are matching now what we do we have a secondary range tree. So, we have a secondary tree structure and we will store it based on the y axis.

So, another secondary range tree will store and that will be based on the, so secondary 1 D range 3 this is also a pre processing space based on the y coordinate y axis.

(Refer Slide Time: 14:18)



So, if we do based on the y axis so; that means,. So, we have the points we have n points each point is having x coordinates y coordinate. So, on the y coordinate on the x coordinate we have this range tree and we got this subtree and on the y coordinate also we have the range tree. So, we have lots of data structure is going on. So, on the y coordinate also we make the range tree and there. So, we have for this points we have similar kind of structure is there for the y coordinate.

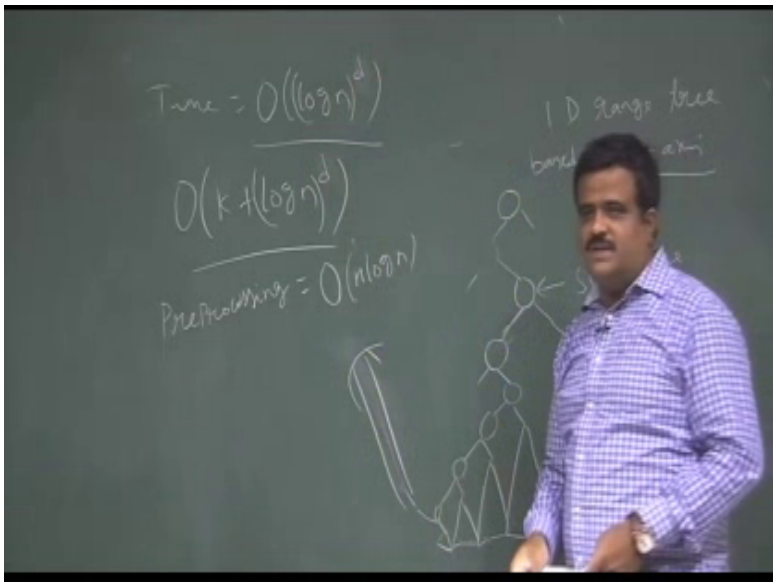
So, these points will be just, this points will be like this like this. So, these are based on the y coordinate. So, we have the similar same size tree, but these are based on the y coordinate we have like this. So, this is the range this is the part of the range tree in the y coordinate because we, so their size is same because this number is same. So, now, what we do now for each of these we do the range 1 D range search query. So, we now we have a y 1 y 2 we have a y a y interval. So, we do the range search you start with this root we go here, go here until it is clear speed node. So, this is the speed node then we continue if we go right then all this available like this we continue. So, these are the points where y axis is also matching. So, these are the points we are looking for. So, these are the points where y axis is also matching. So, that is the idea.

So, basically what we are doing we have 2 range. So, basically we are having two data structure 2 range tree one is based on primary range tree one is based on the x axis we store in a tree and then the second one is based on the y axis we store energy for all the points because we do not know which points will come into this. So, lots of data structure is going on.

So, then we first apply our 1 D range search query on this x axis we got some subtree and we have similar. So, this is the similar data structure we have the similar data structure with the same nodes here based on the y axis and then in that tree we have to do the 1 D range search based on the y coordinate and that will give us basically the matching of the y coordinate also. So, that will give us the points which are in the rectangle. So, that is the idea. So, this can extend for further dimension. So, on this if we have x y z axis we have another primary range tree and there we have to search. So, this will match x axis y axis z axis. So, this is the idea.

So, what is the time complexity for this? So, for this we are basically log n square because this is the log n and here again if we need to go for log n square for each suppose there are log n tree subtree. So, time is basically log n square.

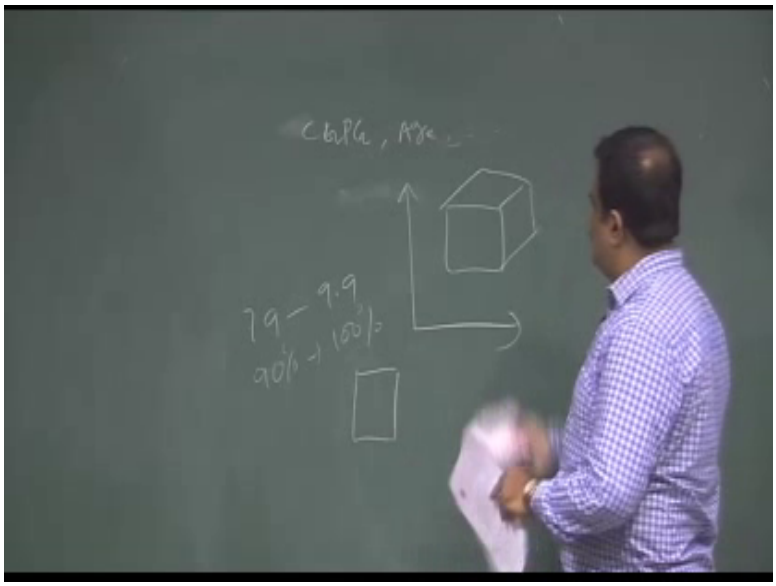
(Refer Slide Time: 18:13)



So, after knowing the subtree if we have to report all the elements then it will be k plus log n square k is the this is the output sensitive if k is the number of points which are lying in that rectangle then it will be k plus log n square and pre processing time is same a pre processing time is basically same as earlier it is basically n log n and the space is also. So, now, if you want to generalize this for D dimensional then query time will be taking 2 to the D this to the power D because we have D many of such dimension and each dimension we have to search it.

So, we will just quickly look at another problem geometric problem and this has lots of application in the database system because basically we are; what we are. So, this range such tree has lot of application what is those because suppose we have a n dimensional D dimensional data n data which is D dimensional. Suppose student record student records each student has many attributes, so D dimensional say marks in or CGPA age like this. So, there are many dimension and the marks in algorithm marks in some mechanics something. So, now, these are the dimensions. So, based on this we store it.

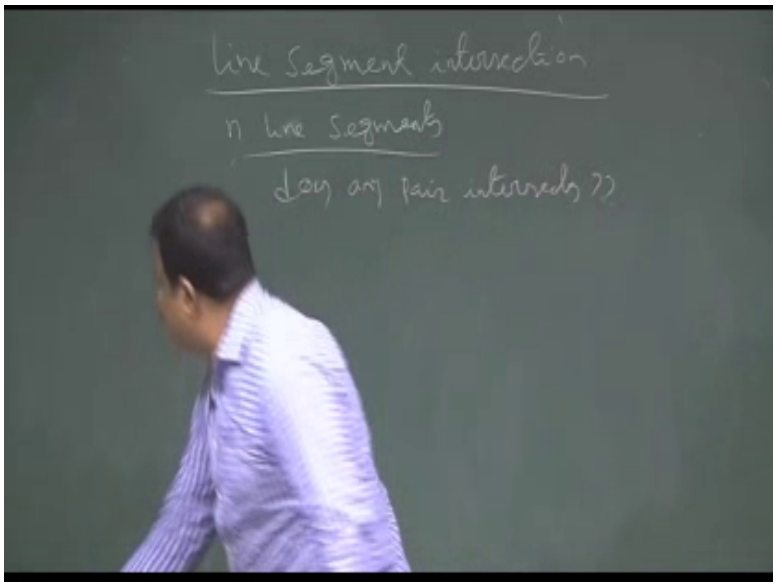
(Refer Slide Time: 19:50)



Now suppose we have a query like this we just have a points like this x 1 x 2 then, suppose it is say 3 D then we have a range like this and we want to see how many points are belongs to this so that means, we want to we want to fix that. So, Microsoft came for the campus in. So, Microsoft wants database of the all students name of the students whose CGPA is lies between 7.9 to 9.9 and whose algorithm marks is basically lies between 90 percent to 100 percent is the 2 dimensional range search.

So, we have to get all the students who is matching with this range. So, this is a rectangular search 2 D search. So, then there are lots of application of range search tree. Anyway let us have a quick another problem which is called a line segment intersection problem.

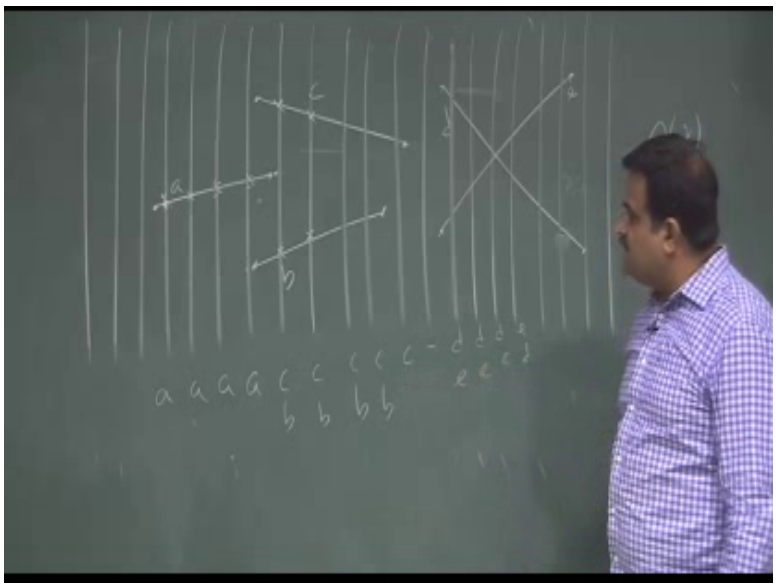
(Refer Slide Time: 21:17)



So, the problem is basically this is this is also static a data structure. So, we have some lines, we have few line segments say we have a line segments line segments and we need to find out any 2 segments are intersect or not. So, does any pair intersect that is the public.

So, for example, if we have a line segment say this is the line a and say we have a line like this like this suppose this is a b c d line this is line segment not line, line has no starting point and ending point, but segment has starting point this is e line.

(Refer Slide Time: 22:08)



So, suppose these are our inputs there are n line segments now we want to know is any pair of this segment are intersect. So, that is the idea. So, what is the near approach? So, we can take any 2 segments and we can just check by coordinates wise whether they are intersecting or not. So, that will take order of n square because there are n c 2 pair. So, just take any 2 pair. So, we will check, but we want to do see something better way that is called sweep line algorithm, sweep line algorithm. So, basically what we are doing in sweep line algorithm we are having a vertical line. So, we just move this vertical line this way. So, we have a vertical line, we move this vertical line like this like this. So, this should be continuous, but you cannot have a continuous. So, this is the movement of vertical line. So, we move until we end to this now.

So, while moving what we observe. So, we observe the ordering of this number ordering of this intersect. So, this is the intersect point or the line this is the intersect, intersect, intersect. So, here, it is studying from here intersect intersect intersect intersect. So, we are interesting in these intersecting points and if we maintain a data structure dynamic data structure with this intersecting points and this we are putting in the order of the value of the x axis y axis. So, this is the value of the y axis ordering.

So, if you do that, this is basically the ordering is basically like this, this is basically a a a a a is finished, this is c b, this is also c b, this is also c b, this is also c b, now these phase this is only c this is nobody is there. So, this is basically d e and this is also d e, this is also d e this is basically. So, sorry this is d this is e this is basically d. So, we maintain a dynamic set ace for this storing these numbers this y value. Now, so when this value will change in the 3 condition this value will change either when a new line is enter new value is enter; that means, a line segment has joined so that way and when in line is finished then it will change.

Otherwise it will change when there is a intersection because here it has change, here there is a change because b has finished, here there is a change because c and b are joined. So, either this change will occur either a new segment is joining or a segment is leaving or there they are crossing they are intersecting. So, we have the interest here. So, how to check and whenever a change is occurring we will see the neighbouring node we will see the neighbouring lines and we will check whether they are intersecting or not. So, for this dynamic set this set is we need to maintain a data structure. So, what we can do we can use the (Refer Time: 26:31) tree which is balanced and that will be done in

order of a n log n time. Anyway this is the rough idea of this sweep line algorithm. So, if you have interest you can look at the textbook for the details of the pseudocode.

Thank you.