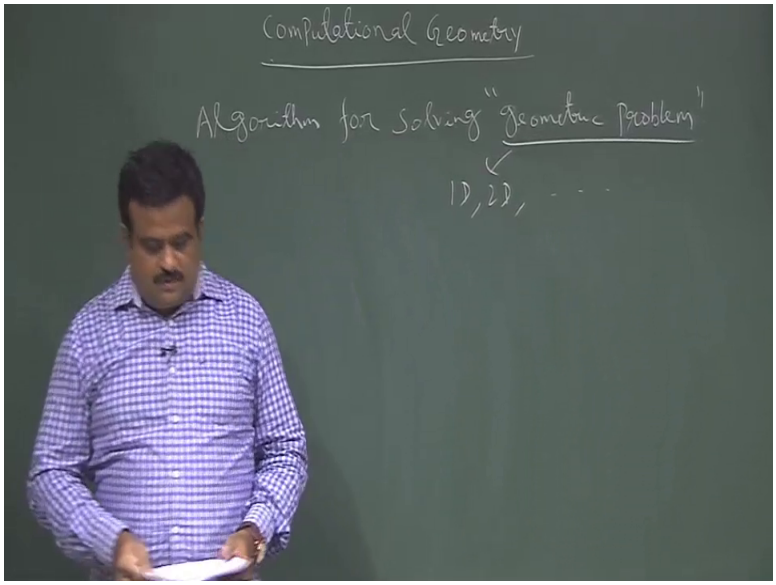
An Introduction to algorithms and analysis
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 34
Computational Geometry

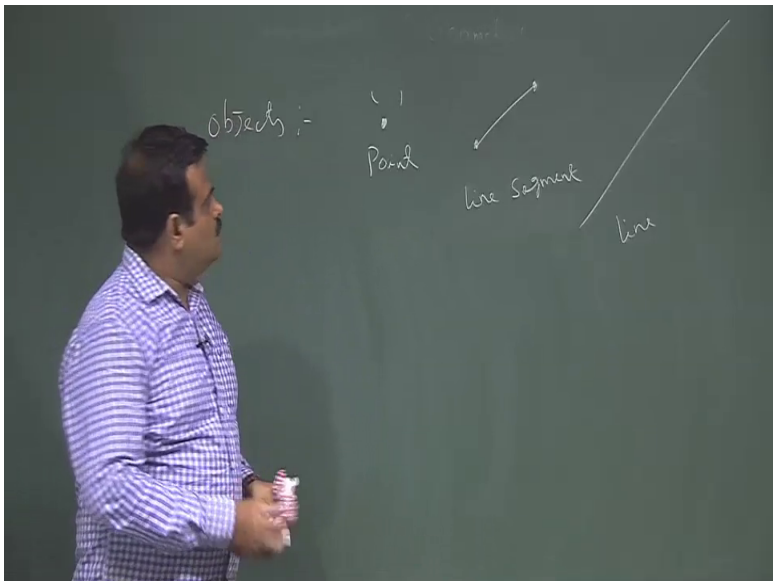
So we start the computational geometry. So, we will talk about the not much more more lecture on this area just maybe two lectures on this. So, this is the neat field for geometric problems.

(Refer Slide Time: 00:42)



So, this is the area the computational geometry is basically algorithm for solving the geometric problem, this al this is the area where we proposed the algorithm for solving any geometric problem solving geometric problem. So, any geometric problem if we want to solve then if you have to propose some algorithm for that, that is comes under this area which is called computation geometry. So, it could be geometric problem in any dimension it could be one d it could be 2D two dimensional or it could be some high dimensional also ok.

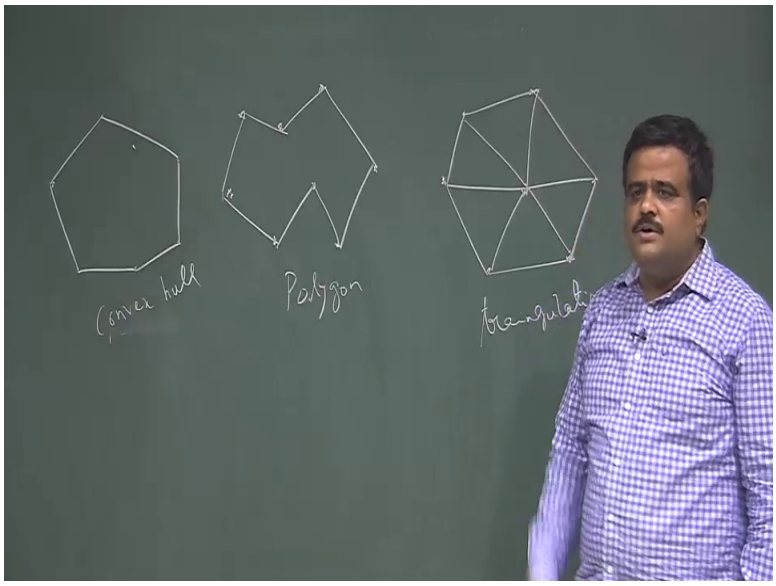
(Refer Slide Time: 01:41)



So. So, now, to start with let us talk about what are the fundamental objects in geometry. So, what are the objects fundamental objects in geometry. So, a point. So, point means it could be any dimensional pair point here if this point is on two d plane, but it could be in any higher dimensional point it could be a three d then we have three compound three dimension x x y z it could be d dimensional place. So, the point and a line segment this is a line segment and it could be a line. So, these are the six fundamental object in a in geometry a point a line segment and the line ok.

So, now suppose. So, what are the basic structures we can take suppose you have some geometric problem we can discuss suppose our our input is some points. So, here we have given some points.

(Refer Slide Time: 02:42)



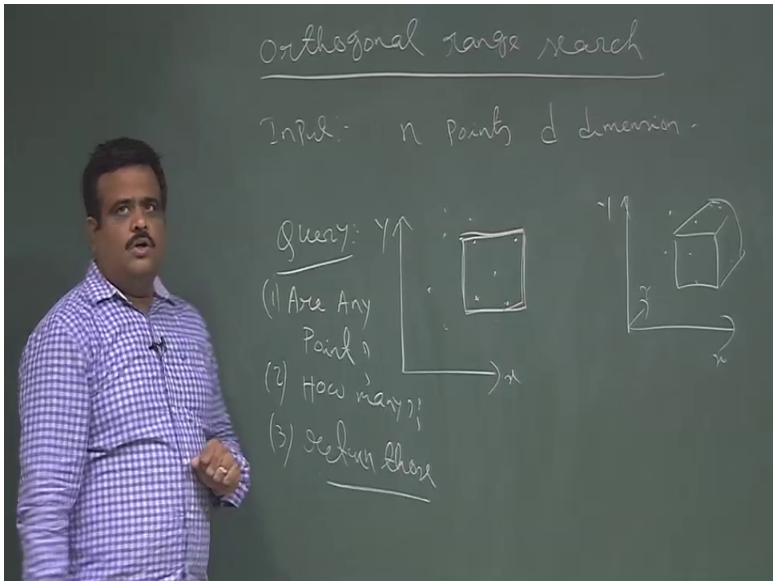
So, these are the points it could be any dimensional points. So, now, maybe we have to think for finding a say polygon using this point this is the smallest intersection which is covering all the points. So, this is basically polygon. So, given a point set. So, this is the point set we have this is a point set given a point set may be one structure may be we need to find out the smallest region covering all the points.

So, this is basically polygon or maybe we have to have a triangulation problem like we have these points we have to form the triangle triangle using these points. So, like this. So, minimum number of. So, triangle. So, this is the triangulation -[triangu]lation or maybe we have to find given a points we have to find the say convex hull. So, this is the convex hull. So, this is a closed region in which if we take any two points then the line should be inside the region. So, this is called convex hull. So, these are some geometric structure. So, these are some problems we have given the points we need to construct this. So, there are others structures as well also available.

So, basically we are if we have interest you can see this book this is mark mark mark book. So, this book contained a few more algebra a few more structure on this. So, this is the book for this todays lecture we will follow this books and tomorrows lecture the other next lecture will follow our textbook. So, let us start with the problem which is called orthogonal range search. So, this book contains some more applications on geometric problem. So, this is the very good book on computational geometry. So, if you

have interest you can go through the details on this book. So, our todays problem is orthogonal range search orthogonal range searching ok.

(Refer Slide Time: 05:31)



So, the idea is suppose we have the n points. So, we have given n points input is n points n points it could be any dimension say d dimension and we have given a range orthogonal range. So, the query is. So, so. So, if it is a two dimensional point. So, suppose you have some two d points is given. So, there we have two dimensional plane x axis y axis. So, we have some two d points and we have some range. So, we have some box. So, parallel to the axis. So, this is parallel to the axis. So, this is also given. So, suppose this this is also given.

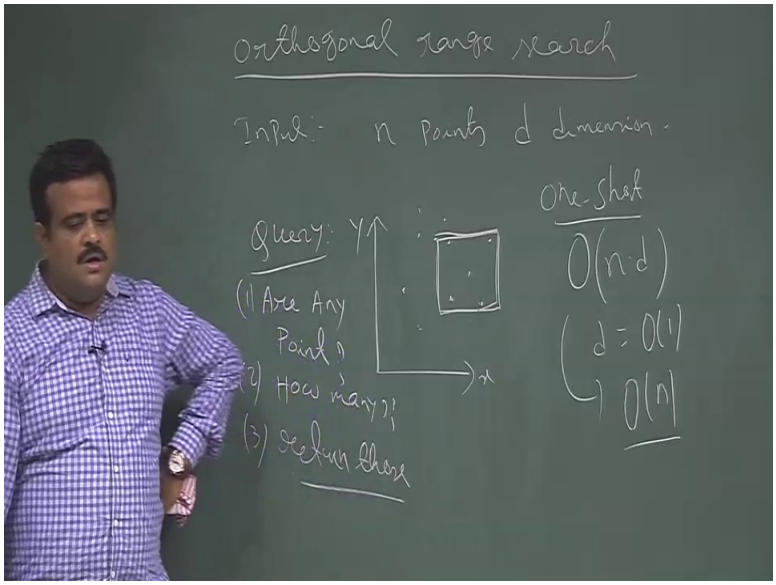
So, we have given a range range means the rectangle here in two d and in three d it will be like this. So, it will be like this. So, we have a box like this. So, so this is x y z axis. So, basically we have given a box parallel to the range parallel to the axis and it could be higher dimension also and we have to the our query is three types of query we can do is there any point in this range this is the first query if the answer is yes then how many are there this is the second query then then tell us the points give the points if there are k many we want k points which are in the range.

So, these are the three types of query one can get. So, is this problem clear this is this we have given a range orthogonal range; that means, it is parallel to the axis x and then then you have to find out the points which are in this range. So, this is called orthogonal range

search. So, how we can have a how we can handle this. So, so first query is are there any point are any points this is the one query then second one is called how many if the answer is yes then the return those return those like this ok.

So, now how we can handle this problem what is the net solution of this problem suppose if it is d dimension. So, how we can check if point is lies in this range or not. So, one sort. So, we take a point we just see the points dimen if it is two d points we check the x axis x coordinate y coordinate and we have a for two d points we have this rectangle then we check whether this is this x axis and x coordinate y coordinate is x coordinate by y coordinate fitted in this box or not lying in this rectangle or not. So, three d points we have three axis then x y z we check the three values if it is there.

(Refer Slide Time: 09:22)

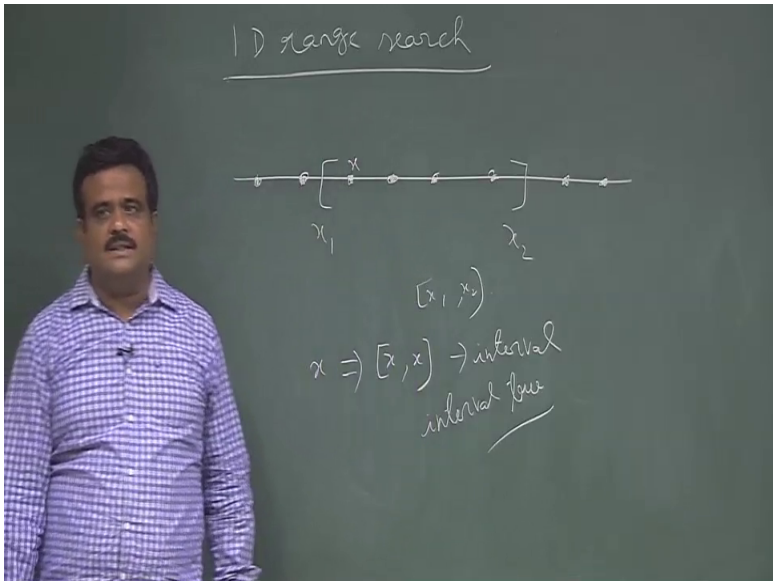


So, this is the one shot solution one shot. So, what is the time complexity for this it is basically order of we have n points order of n into d. So, d is the dimension if d is order of one if d is constant then this will be order of n. So, this is the order of n times algorithm ok.

So, now we want to do something better like we want to do some p processing because this safety statistic now we are not changing this. So, if we. So, this study. So, we we want to have a static data structure on this points n points. So, that we can do the search in faster way. So, that is the that is the todays lectures on this orthogonal range search.

So, we want to have a static data structure for this. So, let us start with one d range search.

(Refer Slide Time: 10:27)



Then we try to extend this for two d and high dimension. So, for one d what we have we have some points on the line one dimensional points.

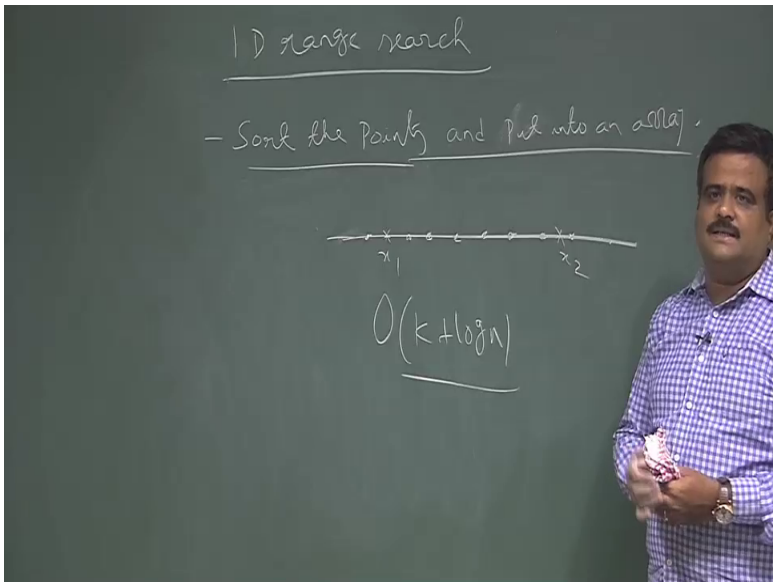
So, this is the real line. So, you have some points on this line. So, these are the points we have and on this points. So, this is the this is start this is a set n n points are there and this set is say suppose static set n points are there now suppose we have a interval. So, that is the range this is the range search only range search we have interval say x one to x two and now we want to find the points lies between here. So, that is the problem. So, how we can do that can you use the technique which we know the interval tree we we we learn this interval in the last i think last lecture yes. So, can you use that interval tree idea that is the augmentation of the data structure can you use that idea.

How we can think of interval research like how we can think this point has a interval then we can we have given interval x one x two then we can see how many intervals are overlapping with this that is the idea. So, how a point can be represent at interval. So, suppose this is x . So, x will be treated as x comma x. So, this is basically interval. So, basically problem is interval search interval tree. So, if we can find interval tree then it will be this problem will be solved when this will take how many times. So, if there are k answers then it will take order of k log n to get the all the points k points, but this this

idea cannot be extended for two d that is the problem. So, we want to have we want to extend this for higher dimension this cannot be extend for two d.

So, can you think other technique where we can do some p processing and then the query will be much more faster. So, what if we sort this point if we just sort this point then just to sort this point and store it into an array. So, this is the second idea. So, first idea is the interval tree and the second idea is the sorting

(Refer Slide Time: 13:19)



So, what if we sort the points because this point is static points we can easily sort the points and we can store into a and put into an array into an array. So, we have some points sorted points. So, these are the points. So, this now points are sorted.

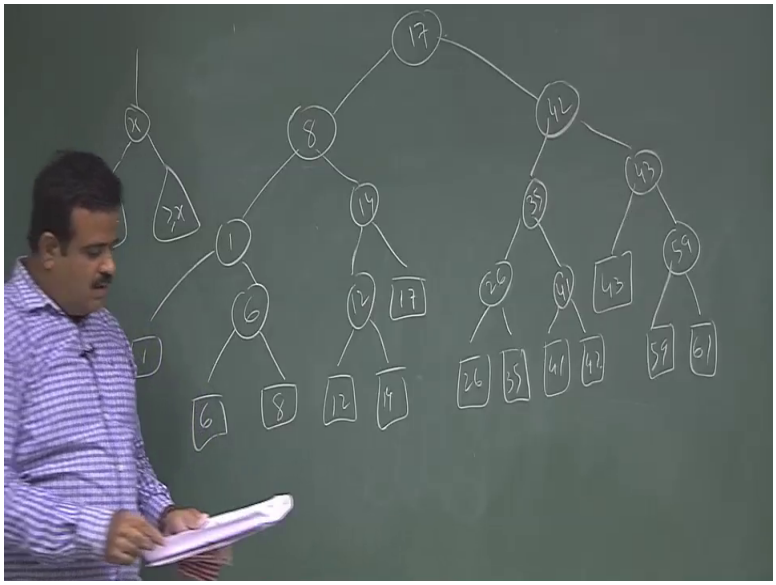
Now, what we will do now how to have this query the range search. So, we have given two points x one x two say we have given an interval. So, here range is the interval what we do excuse me. So, we do the binary search on this point and we do the binary search on this point then we got the position and then the all the intermediate points are basically our output. So, this is the idea. So, that will be basically, but the sorting will take the order of a log n, but each binary search will take log n time. So, it will be basically having two times binary search we are doing.

But we are checking this points. So, it is basically k plus log n where log n is the time for binary search this is, but the problem is this idea cannot be extended for two d because

for two d we have x axis y axis. So, based on which axis we will sort. So, that is the problem. So, this is cannot be extended for two d. So, now, we will think about some pre structure whether we can have a three we can have a statics data structure which is basically balanced tree then we can try to solve this query. So, that is the that is called one d range search tree. So, this is the third solution one d range search tree. So, this is static data structure. So, you have given n points how we can form from this tree. So, the idea is we put all the points instead of see in that tree what we do if we distribute all the points over the nodes what the nodes starting from the root to the leaf.

Now, here what we are doing instead of that we are putting all the points in the leaf at the leaf level then we form a tree that is the idea. So, how to do that let us have a picture

(Refer Slide Time: 16:07)



So, suppose these are our points one six some arbitrary points twelve fourteen seventeen then we have say twenty six thirty five forty one points we are storing into the leaf then here try to form that tree and this tree we want to be balanced and this tree we want to be balanced search tree.

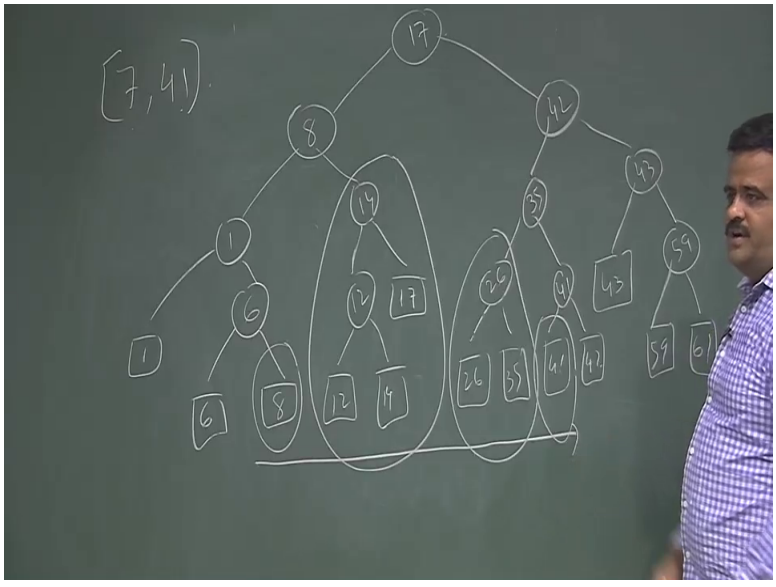
So, now we want to put this to have this then we want to merge this merge this then we want to merge this two similar way we will do like this you want to merge these two we want to merge this two then merge this we want to merge this two merge this and then we want to merge this and then finally, we want to have the root. So, this is the way we are going to form the tree so we are putting every inputs every n nodes in the leaf

level and you want this to be a binary search tree. So, for that what we need. So, we need to put a value over here such that it will be a binary search tree; that means, the it will be greater than six and must be less than eight.

So, we can put any value between eight and six, but we are going to put the maximum value of this node i mean like the idea is to put the maximum node value in the left subtree. So, that is basically we will put it sixteen this is one this is twelve this is fourteen maximum value in the left subtree is eight this is eight this is twenty six this is forty one this is thirty five this is fifty nine forty three maximum value in the left subtree is forty two and maximum value in left subtree is seventeen. So, seventeen. So, this is a binary search tree why this is a binary search tree if you can check any node if you take x.

Now, left subtree right subtree. So, this is less than equal to x this is greater than equal to x this property is satisfying. So, this is a binary search tree this is a balanced binary search tree. So, this is a good news. So, this is called one d range tree now we want to use this to have a. So, this is our people this is a static data structure. So, we want to use this for our query what are the intervals what are the points overlapping with a given interval suppose given interval is seven comma forty one suppose we have given seven comma forty one interval. So, this is x one this is x two.

(Refer Slide Time: 19:29)

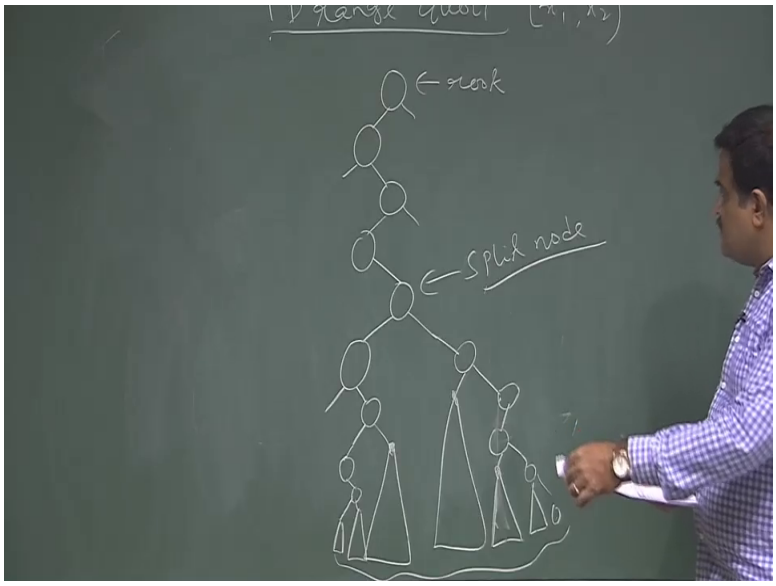


Now, we want to see what are the interval overlapping with this. So, seven comma forty one means. So, starting from here to forty one here so; that means, this one and this one and this one and this one. So, the leaf in all these roots all these subtrees. So, when you output if we if we just return this if we just return these roots and if we say that mine my result is all the elements all the leaf nodes in these root where roots are basically eight fourteen twenty six forty one. So, eight. So, the leaf of eight is eight leaf of fourteen is basically twelve fourteen seventeen leaf of twenty six is twenty six twenty thirty seven and leaf of forty one is basically forty one. So, this is the output. So, instead of returning all the elements separately we just tell that these are the roots these are the trees subtree is my answer.

So, what to get the elements the elements are in are intersection points of basically in the subtree all the leafs. So, you have to go to the leafs and we have to find the. So, how many such subtree will be occurring. So, is this clear. So, basically we are looking for the subtree which we can output and we can tell our answer is our basically points are leafs in this subtree. So, instead of returning all the leafs we just output the subtrees and then we go to the leafs of this subtrees to get our points which are intersecting with the intervals that is the idea. So, that that is what is called that is we are going to do in our algorithm which is called one d range search.

So, the general idea is. So, we are going to return this tree this tree this tree this tree thats it in our algorithm. So, upon after return after getting this tree we just take the leafs of the tree thats it. So, for the time being let us erase this. So, let us just. So, this is the one d this is the general one d range query. So, the we have given a interval x one x two.

(Refer Slide Time: 22:20)



So, the idea is we start with the root this is the root and then. So, we may find out there is nothing interesting in left. So, you may have to go right nobody is there interested in left

So, we go to the right and then from here we see there is nothing interesting in the left. So, there we go for the right. So, like this we continue. So, we see nothing interesting here like this we continue until we reach to a node where we found something is interested in left and something is interested in the right; that means, there are some nodes which are in the left subtree which are relevant and there are some nodes which are right subtree which are also ah relevant; that means, whose which are in the range. So, that node is called split node. So, this is called split node split node means we have. So, this is subtree this is another subtree so; that means, if both the subtree has some nodes of our interest; that means, both the subtree have some nodes which is in the range x one comma x two.

So, now we have to travel both the way now we start with this the left edge. So, we start with here say. So, you go to the left edge. So, now, again say we go to the again say we see there is nothing interesting in the ah left subtree we go to the right now suppose again we go to the left; that means, everything is interesting over here if we are going to the left; that means, all the all the subtree this is interesting. So, the then again from here suppose we go left and suppose here we go right then all the there will be interesting like

this. So, these are the subtree which are interesting similarly we have to go like this. So, suppose if we go right from here then all the subtree will be interesting here.

So, like this if we again go left then if we go right then all the subtree will be here interesting like this we continue. So, basically our our answer is this subtree we are going to return this subtree as the answer. So, once we return this subtree then we go to the individual subtree and we get the leaf nodes and that is our in the range. So, this is a general idea of range query and how many subtrees will be there basically how many. So, if there are k nodes which are overlapping with this. So, there will be this is there will be basically log k subtree or log n log k or log n intuitively it should be log n, but we can show that there will be log k subtree will be there anyway we will talk about that. So, this will be our basically output of the range subtree algorithm. So, this is the range subtree algorithm.

So, in the next class we will talk about the pseudo we we will see the pseudo code of this algorithm, but this is the general idea idea is to start we have a interval we see we match with this. So, if nobody is interested in the left right then we go to the left we will see the pseudo code in the next class then from here if you see nobody is interested in the right we go to the left we go to the right then this way we reach to a point where we will we will see every there is some some nodes which are in the range and there is some node in the right which are in the range so; that means, this is the node called split node so; that means, there is something interesting in the left path there is something interesting in the right path.

So, then we continue like this. So, we will talk about the we discuss the pseudo code in the next class.

Thank you.