Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Module No. # 11

Range Searching

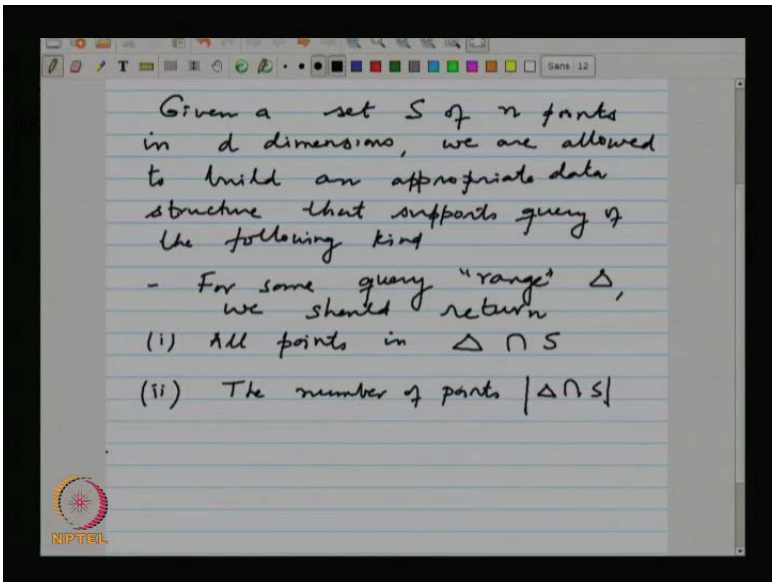
Lecture No. # 01

Range Searching

So, today's topic is range searching which is probably one of the most important areas of computational geometry and also application is extended to almost every area of computer science, namely database searching and more closely, probably you know geographical databases.

So, what is it? So, there is a very general definition, but let me not get into too much generality, but just let me give you the reasonable definition of what it is.

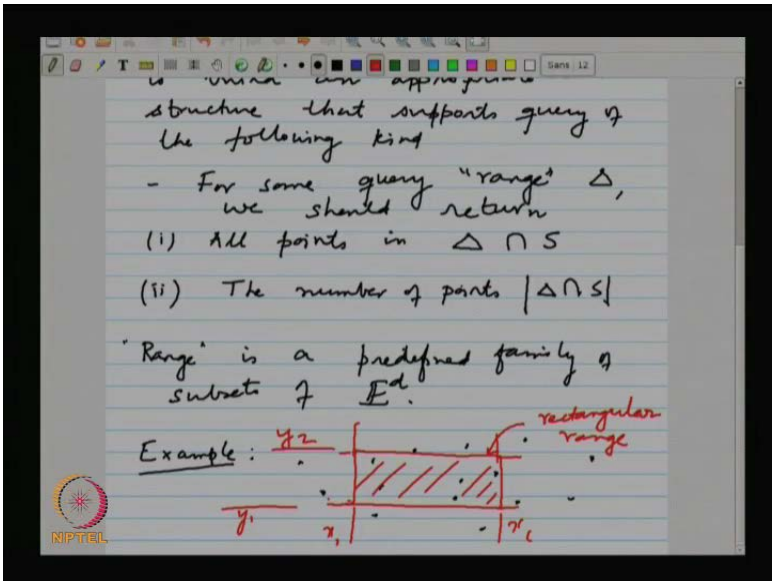
(Refer Slide Time: 01:10)



So, a given set of points, and let us say in a plane, more generally in d dimensions. Given a set S of n points in d dimensions, we are allowed to build an appropriate data structure, that supports query of the following kind. For some range, I will just define what I mean

by range is, some given range is delta, we should return with the two kinds of queries. One is reporting query, where we must return all points in delta intersection s or the cardinality acp number of points right.

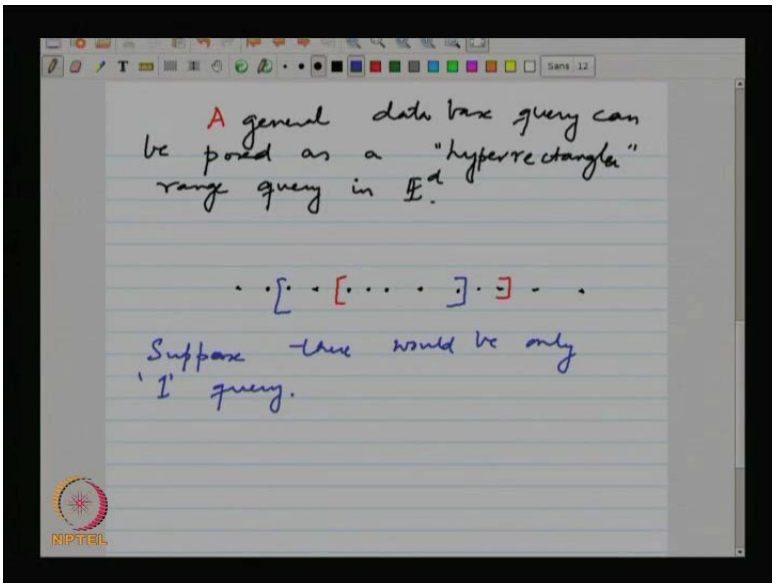
(Refer Slide Time: 03:21)



What is a range? A range is a family, it is a predefined family subsets of Euclid and d dimension right. So, simplest case or let say very natural case would be example, points on plane and my range could be let us say rectangle right. So, this is a natural query for database kind of problems. So, your query range is a Cartesian product of two intervals right. So, you could specify that give me all the points that have x co-ordinates between x 1 and x 2, and simultaneously between y co-ordinates between y 1 and y 2.

So, we have Cartesian product of two one dimensional intervals and most common database queries can be posed in this way that we have points in some dimensions which corresponds to the attributes some high dimension spaces. If I have ten attributes, then I will work in ten dimensions. Then you have a query that give me all those points or data that have x 1 co -ordinates between this, and this x 2 coordinates between this and this x 3 co- ordinates between this and this.

(Refer Slide Time: 06:00)

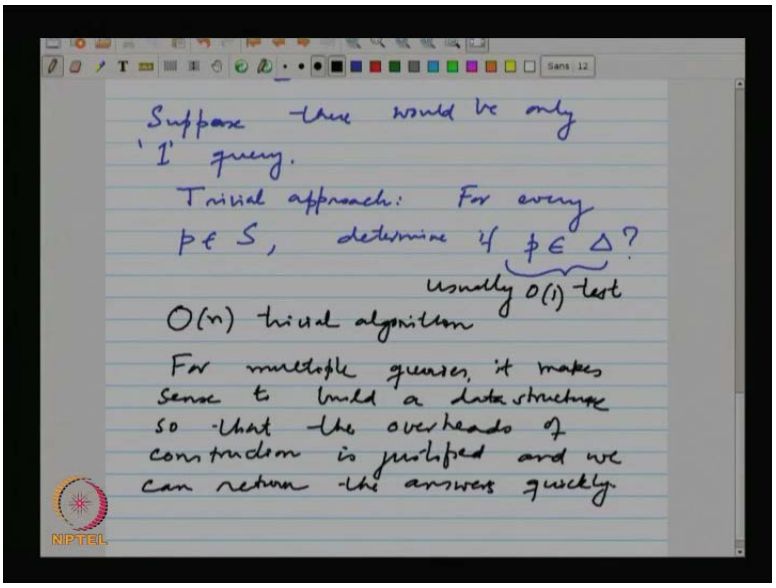


That would be hyper rectangular. So, a general query. So, a general database query can be posed as a hyper rectangular range query in E d, and most trivial example of this thing would be one-dimensional right. So, I have points on a line and my intervals are closed intervals, sorry my range is a closed interval.

So, either this or it could be this etcetera. Now, the important one of the things that I have actually mentioned the definitions, but it strictly speaking it need not be a part of the definition. I am already referring to data structure that builds in appropriate data structure that supports a query of the following kind. This is under the assumption that there is not one query that you want to handle, but you know the whole sequence of family of queries.

If just one query you know suppose, we were interested in only one query. Suppose, there would be only one query, then clearly the overheads of building the data structure of something that you do not want to bare and something that not worth it. So, what you going to do is what is most trivial way of answering a range query.

(Refer Slide Time: 08:15)



So, I am going to basically determine for every point. The trivial algorithm will be for every point. So, trivial approach for every point in the set determines if p belongs to the range and assuming, this is a simple constant time test. Usually, your ranges are not going be too complicated. I mean although I have not put any restriction in the definition, but mostly this test usually, it is a constant time test.

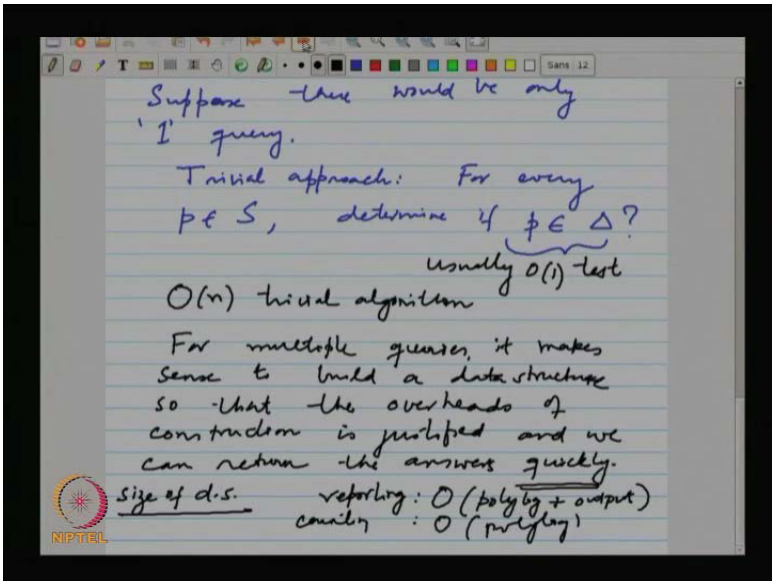
So, for any kind of natural range query problem, you can provide the answer in order n time, just check out every point with respect to that given range whether or not belong to range right. So, order n trivial algorithm and it does not really require you to build any databases. Sorry any data structures, but the problem becomes interesting once we go beyond this one query or a few queries things. We know there could be a large number of queries. You know perhaps millions of queries or it just would be a continuous process.

Where you get these queries? So, for multiple queries, it makes sense to build a data structure, so that the overhead of construction is justified and we can return the answers quickly. The meaning not like order n time something fast. Hopefully, something for a logarithm, but there is a catch. Then, you know we are used to this model now. So, there are two kinds of queries right. So, one is the reporting query. This is the reporting query where I am supposed to report or produce as an output all those points that lie in this range. The other one is a counting query, and by now you are used to the fact that in the

reporting query, the minimum time should be at least, the size of the answer or the output.

In the case of number of points that becomes much simpler, it is some number between 1 and n that is much simpler, but then in the type one kind of query which is the usual kind of the more common kind of query, there would be some kind of additive things. So, it is not that ideally we would like to report it in time proportional to the number of points in the range, but then you do encounter some fixed term and that fixed term is usually polylogarithmic term.

(Refer Slide Time: 11:54)

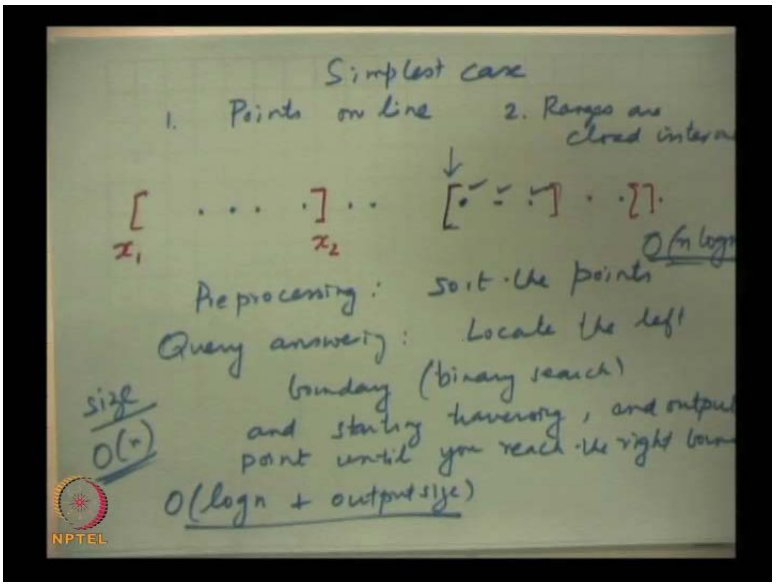


So, ideal kind of bound for range queries, so this is quickly as for reporting, we mean let us say about order some poly log plus output size. In the counting, we would expect some poly log. So, this is basically objective when we build a data structure and also almost equally important what other parameter is. The size of data structure correct the other thing is the size of the data structure and somewhat less important is the preprocessing time.

So, with this definition in place, let us see what we can do with this. I already do this picture. So, for points on a line and the ranges are intervals let me shift to the other one. So, the simplest case probably points on line and ranges are closed intervals. This is my given set of points and any point of time, I could be given what is report all the points

between x1 to x2. It could be any kind of interval. This one or could be this one. So, how would you solve this problem?

(Refer Slide Time: 13:07)

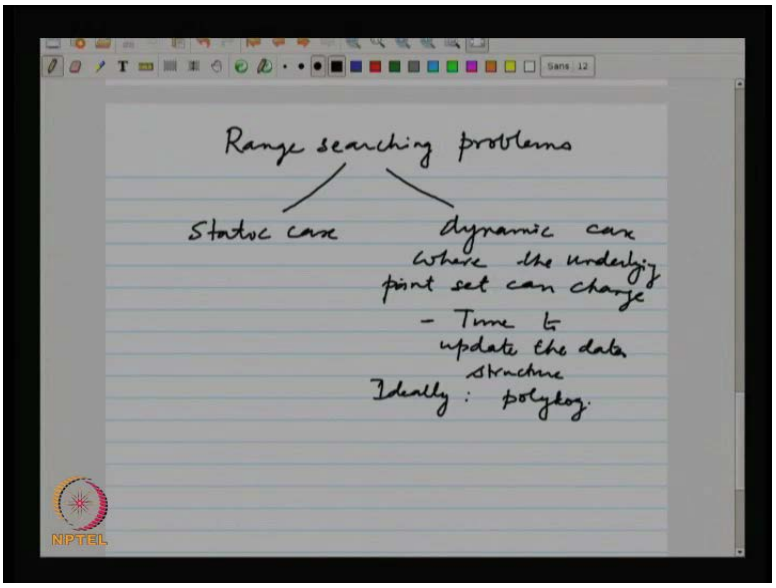


It just looks rather simple. So, you must basically locate your left boundary of the interval and if you have kept this point sorted. So, essentially what you are doing is sorting the preprocessing sort pre-points right, and query answering locate the left boundary using binary search and start walking. Basically, start reversing and outputting until you reach the boundary.

So, it is this one. I locate this one and using a binary search and then output, this output, this output and I find that I have reached the boundary and that is the end of it. So, what is the query processing time here? So, log n for the binary search plus the output size right. So, that is your ideal bound of log n plus output size sorting the points. Let say if you use the comparison sort n log n, and you have data structure size is nothing but a sorted list ordered n right. So, all these figures are exactly what we wanted to be.

So, one dimensional problem usually is much simpler. So, time to graduate to the next dimension. So, this may be one more variation I should just mention. So, all is fine till now, but what happens if we also have this provision that points can be deleted or inserted, so in the one dimensional case, I will just write it here further variation or natural variation on these. So, range searching problem may have this following variation. It is a very natural variation.

(Refer Slide Time: 17:07)



One is the static case and the other one is a dynamic case where the underlying point set can change. In the case of one dimension, even if they change as someone suggested instead of a sorted array or something, you just go and do your some favorite in the balanced tree structures which supports insertions and deletions. That is it.

So, all your bounds remain unchanged asymptotically. You still have order and space. You still have order log n plus output size query time and in addition, the other parameters are the time to insert and modify the data structure right. So, what is the movement to have this? Then, the additional parameter is, basically another important parameter is time to update the data structure.

So, this one also should be ideally some poly log. This is what it should be for each point inserted deleted. I do not want to incur too much cost in modifying the underlying data structure and in case of one dimension again, this is order log n point insert point delete point everything is order log n. So, in one dimension, we have everything ideal whatever we can hope for.

Does the same thing happen when we go to higher dimension? The higher ones we move to higher dimension that is two or more. There are actually two things that happen. One is in the case of one dimension what are the other kinds of ranges we can think of? I mean, the kind of ranges we dealt with our interval ranges right. The subsets that are

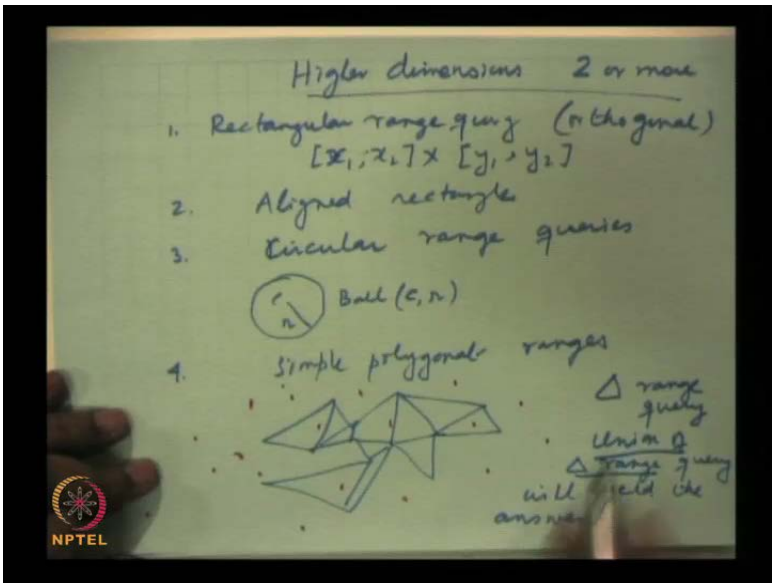
queries and if I mean, I did not say it is explicitly, but clearly the data structure that you have, we have designed should depend on the kind of ranges you are supposed to handle.

So, you build this data structure that whatever sorted set or a balanced tree assuring that my query ranges are going to be closed intervals. In one dimension, you cannot really think of much variations from there right. You can probably think of union of intervals something say that, but they are not very national kind of queries. Even, if it is kind of intervals we know exactly how to merge them etcetera. You know those things you can handle easily. Rather the movement you go to high demonization. Yeah, it shows those cantle set something say that sort of way what people would like to query.

I mean the querying sets see here we are trying to specify those are implicit kind of queries. Here, we are talking about explicit ranges. You know, I am actually presenting the description of the range sum way explicitly. So, in that context you do not whole lot of variation you can think of or I mean, national problems not going to served out deviate much out sidle that. You know what kind of maybe you can think about open intervals or close intervals, something like that maybe union of some finite number of intervals perhaps, but not too much beyond that.

Anything beyond that you know just becomes someone said you know it can be described mathematically, but probably not that useful in Indian life, but the movement you go to high demonization, your ranges could take arbitrarily shapes and forms. So, the generalization of the interval range query will be the Cartesian product right. So, that is a rectangle range query. So, one is that rectangular range query and not only rectangular, but orthogonal right.

(Refer Slide Time: 21:26)



It is basically Cartesian product of one dimension, but then it can have any other kinds of shapes. So, one rectangular range query need not to orthogonal. You know it may be aligned. So, aligned rectangles other kind of natural queries you know circular range queries and that is very natural. What is it? Basically, I am presenting a ball. There is a ball of radius r centered at you know from c. What are the points inside the ball and that is something very natural. You know I am located somewhere and I want to know exactly how many points occur within distance r from there. So, that is your circular range query. If you want to handle arbitrary shapes ok, well, at least if you keep it linear boundaries, then it could be a simple polygonal query right.

You know someone is interested for some reason to know how many points are inside this. Say how many points are outside this? Now, this actually does not go well with earlier model where you are saying that we could test whether a point is inside or outside range in constant time. Now, when if I give you a simple polygon to test whether a point is inside or outside, the simple polygon you know it may take you more than constant time. So, suppose it is a simple polygon and you want to test to the point which is inside or outside and allowed to build a data structure.

How quickly can you answer it? Constant time does not seem to be because it is completely dependent on the size of the polygon that is unnatural and unlikely answer. So, what is the minimum time? You can think of it. So, it is kind of a ray shooting query.

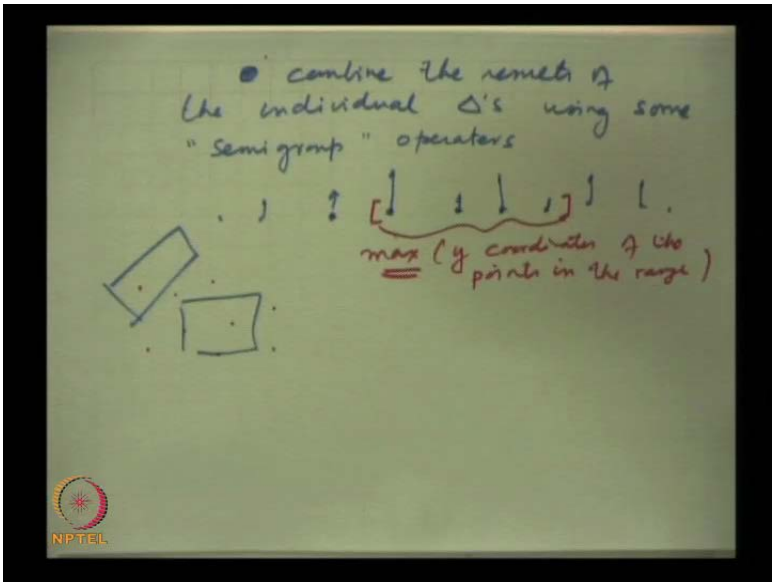
So, if I build a ray shooting and data structure on a simple polygon and that will be log n. So, it is not constant time. You may actually want to modify this polygonal range query in terms of if I want to actually decompose this simple polygon into union of triangles or something.

So, you can think about this polygon. As you know I triangulate this polygon and if I can actually answer a triangular range query, then I can just union these triangles and get the final answer right. So, the union of triangular range query will yield answer again. There are some mathematical possibilities where this is simple union, and this is sometimes also where actually I am going to only add up sometimes.

Simple addition does not work depending upon what kind of problem we are handling. So, here we are handling points and you know simple polygons of ranges. So, you can actually generalize the whole ranges query problems you know not we just point could any kind of objects. Then, this simple kind of additional unions is not going to work. You know there is a natural generalization which uses some kind of operator to combine the answers of these triangles, but I do not want to get into that right now.

Let us not get in there too much of generalization and these operators are usually, so I will just mention this and this. Perhaps you know little later in the course. I think Professor Aggarwal will talk about what is called non-orthogonal range queries. So, we are going to mostly focus on orthogonal range queries.

(Refer Slide Time: 26:18)



So, there are some ways by which you can combine the results of the individual triangles using some usually this operators are called semi-group operators. This is to be just one example. Suppose, you know even in the simple one dimensional case, this point had kind of it is not a single dimension problem really. So, these are points and they have some attributes may be which is like some kind of height. So, this is some of height, this is some kind of height so on and so forth. May be function basically some discrete function in I gave you a query where it says interval query. Tell me the highest y axis in these intervals. So, of course, in this intervals there only these points. So, that is first job to find out which points are we talking, which points are in consecrations.

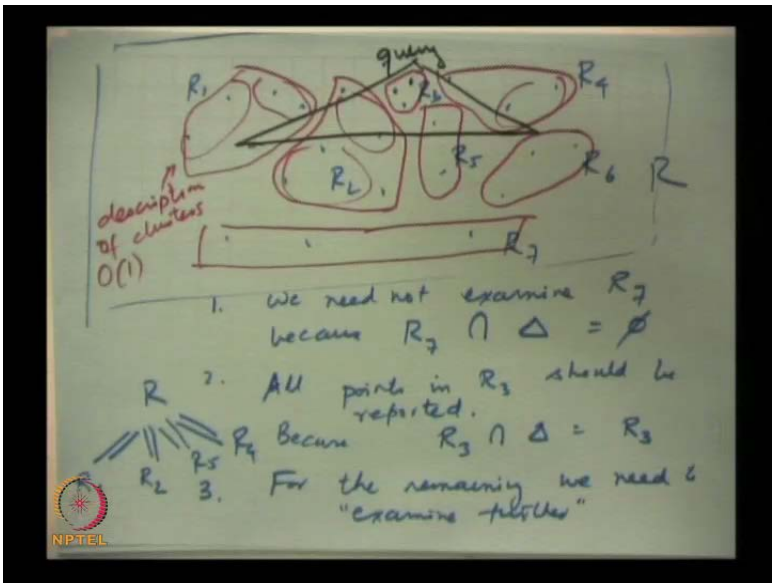
After that I am going to do kind of max of the y coordinates of the points in the range. So, this max is an operator. This is not a two dimensional problem where I am saying that there are two, these are the points and give me all the points that occur in this rectangle this problem. These problems, they are not the same. So, this is actually a one dimension. Range query problem where there is an underlying operator which is a max operator and semi-group, basically means associative operator.

So, it is this operator will also act on the results of the range query and that is how we will get the final answer. So, this is also a range query problem, but it is a more complex range query problem and you can really sort of build very complex range query problems for more and more general definitions. I will just leave it at this and let us proceed with the two dimensional case that is the immediate problem at hand.

The axis is parallel to the axis. A line would be you know it may not be parallel. Well, I just called it back. So, I am calling it orthogonal. I am calling it you know aligned it any angle basically. So, let us look at the two dimensional problem and the first solution that we will examine in the two-dimensional case, you know actually comes from a very general principle. So, let me first talk about the general principle. Then, we will see that it kind of falls out immediately from a general principle.

So, when you have these points, one way that we can do some pre-processing is kind of clustering these points. So, clustering is a very great term. I have deliberately want to keep it way.

(Refer Slide Time: 30:14)



Some way, we are going to sort of patrician or cluster this points. So, I am just drawing some arbitrary clusters. May this is one of these clusters are not too bad, in the sense that they are fairly sort of convex kind of regions, but it did not need to be. So, in some way we have grouped out clusters at this point and then my range query, the kind of subsets that I am dealing with let us say also has some kind of arbitrary shape. Perhaps, it is a triangle. Let us say because triangle is a very high could be it can be aligned in any way. So, suppose here is my query range. There is one query.

So, I have built data structure where I have a description of these clusters. So, I have some description of this clusters and usually, again the description will be, let us say some order one kind of description either like, some may be some rectangular clusters or some circular clusters whatever, some groupings which can be defined easily. It is not a very complicated shape and which will require, have a large descriptive complexity. So, they have some simple description, these sets of points and then I have given this query triangle.

So, some simple observation is the following that let me have another set of points here. So, let me make this observation rather simple, but you know will be quite useful later on. So, let us call this region 1, region 2, region 3, region 4, region 5, 6, 7. So, what I can claim is the following that we need not examine seven. Can you tell me why?

So, this region or the description of the region that I have, suppose I have a way to figure outdoes this query triangle intersects with R 7. I am not individually all the points. I am asking the question about R 7 and not the points inside R 7 right. So, R 7 can contain many points. I could simply ask is, any of the points in R 7 contain in this query triangle and I can test them individually, but I do not have to. I can simply find out because this R 7, this entire region R 7 does not intersect with the query triangle.

I can conclude that I did not prove any further because R7 intersection the range is empty. So, there can be many such things R 6 is also another such region that you know it does not have any intersection with the query triangle. The second kind of course, is the other direction right. All points in R 3 should be reported. Why? It completely lies inside this range, because R 3 intersection, this is R 3. So, these are the two extreme cases and what can this be about the rest.

We really do not know. I mean the query triangle intersects R 1, R 2, R 5 and R 4, but I have no idea how many points inside R 4, R 2 or R 1, because I only know about the description of this region. I do not know how the points are distributed inside. I have not kept track of that and it is very difficult to…

So, based on this information, I have to look further or examine how many points in R 1 and R 5 and R 2 and R 4 are contained inside the query triangle. For the remaining, we need to examine further. Whatever it means and what is the natural way of sort of doing this, taking it further. So, again R 1 is a set of points, R 2 is a set of points and I am going to again do the same thing within these regions right. So, maybe I will just have some more clusters here.

So, now I am going to my final answer. It is going to be the union of the range query of this triangle with R 1, again R 2, R 5 and R 4. So, I have branched out now right. Unfortunately, now this whole thing, suppose I call this entire thing the set of points as R. So, the range query R is the regions that completely lie inside, that is taken care of the first level itself.

The regions that do not intersect, that is also taken care of, but recursively we need to handle all the regions that partially, have some partial overlap with this. So, it becomes union of R 1. In this particular case, R 1, R 2, R 5, and R 4 which actually is not good

news because if you want to write any kind of recurrence relation for the query time or whatever, then from a single point, you could have four way branching and (( )) that.

If you have four way branching or five way branching who knows, so the whole thing may not turn out to be very efficient. After all, what do we want? Our goal is to report in time proportional to a fixed logarithm term or a poly logarithmic term plus the number of output points but any kind of reference where you have so many branches is unlikely to be very efficient. So, that is why we have to be careful when we define this partitioning right.

So, this is a very general approach to range query. That given set of points we would like to partition them in a certain way and then the query will be answered as union of which partitions this query triangle or whatever the query is, the range has non-trivial intersection way, if it fully overlaps, no problem. It does not overlap at all, no problem but partial overlaps are the one which have to be recursively sort of handled.

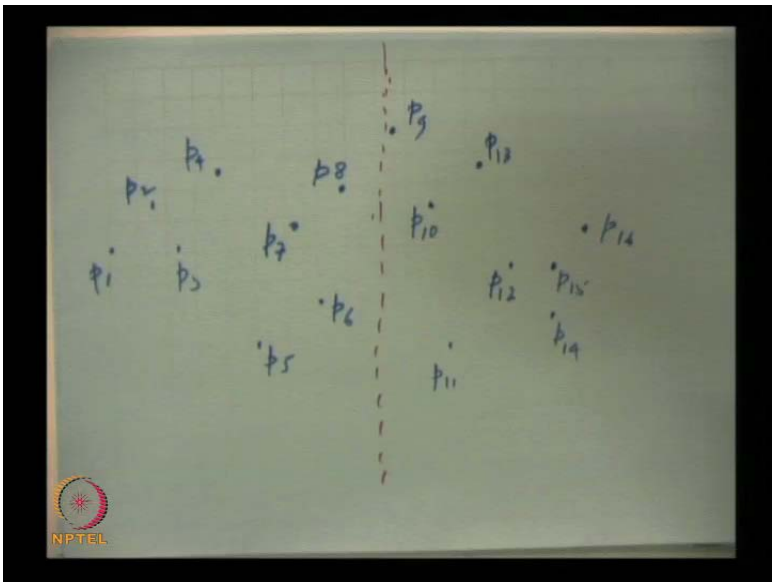
So, we will basically in two dimensions, we will look at a special way of defining this partition. So, this is very arbitrary partition. In this way, where I could have a four way, five ways. Who knows how many ways I have to recursively look forward? So, I want to somehow bound, you know the number of branches that I am going to have.

So, any suggestions or any ideas as how should we branching or how should we partition these points? We are handling orthogonal range queries. We are not arbitrary line rectangles. You know this solution fine. So, thanks for confessing, but you know, if no one else answers, I will let you answer. It will be some kind of binary search. Binary search is a very you know what you say ubiquitous term. You know there are many ways that you can do binary search. So, what are you partitioning on that? Half-half, in what direction? Half-half is a good idea. Half-half basically means that you are data structure; the depth of the data structure is going to be sort of logarithmic. Half-half is a good thing.

So, how would you partition these points? Yeah, hierarchically fine, but that is what we said. We already discussed that. We are going to recursively partition further let say at the top level how you are going to partition? We are handling only rectangle range queries on. What is query triangle? No I am not handling query triangles. So, I mean they gave the example as a triangle because it works for any kind of ranges, but we are going

to look at only for schemes, for rectangle range query, orthogonal range query. Take the median along y or x. Does not matter, let us take it first. So, we have, I had a example. Is this visible? You cannot. No, I will draw it. Then, the basic idea is clear. So, I am going to, it is visible now. Better, there is a gap fine.

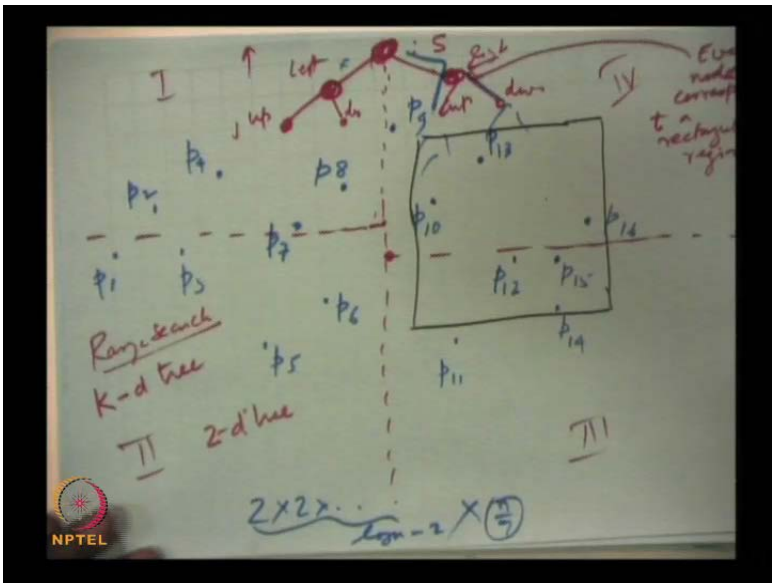
(Refer Slide Time: 41:18)



Just being glazy. I need this exact point set, but I draw just point sp 15, p 16. So, there are 16 point sets. Thus, all of you agree. We should try to find median, their 16 points. The numbering of the points was such that, so, my first level of portioning is this. I draw the median along with x coordinate. What should be your next? Someone already saying you know, let us not be partial towards one coordinate. Let also treat the other coordinate really. So, this is x coordinate. Now, how we patrician should we just draw the couple of way. So, I have these 16 points and could have draw ay, so that you know I balance out the number of points above and below.

Want to do in that way, that is one option. So, I keep this separately. So, this set of points and these set of points will be treated separately because this is one patrician, this is another patrician. Now, I recursively partition this and recursively partition this, but then now I use the y to partition the median, but then they are handled separately. So the 8 points here. So, on the left side, 123, I will know from 4, may be this one. So, this looks like a separator in the y direction for the left sub problem and for the right sub problem, may be this one.

(Refer Slide Time: 43:45)



So, this you can see that these two lines, you know they are not the same lines. They are not flush. They are necessary flush because we actually partition them separately. Now, what do we do after this? Now, we feed. So, now I have this one set of points, second set of points, third set of points, fourth set of points and we are back to the original problem. Again, within this each of this partition, I can do the xy partition area. I basically recurse like this.

So, what happens when I get query rectangle? When I get a query rectangle, well I mean, I could be lucky and I could get rectangle like which means that. So, how is the search tree build? Let us first look at that. How the search tree is build? Initially, there should be some node of the tree that corresponds to the entire space, the entire point sets. So, this corresponds to the entire point set. Then, we partitioned along the x axis. So, I create two nodes corresponding to the left half and the right half. So, this represents the left half point. This could be the right of the points. Then, within each of these halves, left and right halves, I have up and down right.

So, this is again the up down and at this point, we are basically, back to the original problem, but again we can repeat the same thing. So, in two levels, after two levels basically, again we do the left right, up down, left right, up down. So, alternately we do the x and y axis and eventually, when we have only one point, we do not need to do anything. Every time we are partitioning or we branch to a left right child. We are using

the number of points and this process cannot continue for too long because you are having the number of points. The depth of this tree is only logarithmic and finally, each leaf node is going to be associated with only point.

Now, what do we do for these intermediate nodes that we will see later, whether we are going to store all the points here or we do not need to. That is something we will find out later, but we did certainly do is that with each of these nodes, we need to store the description of region necessary will be that. So, the initial node must, you know you can enclose all the points in the rectangle hue rectangle. So, it will contain the description of that rectangle, this and this will contain the restriction of left half, the left rectangle and the right rectangle which is a constant time discrete where a constant size description. So, each node is associated with some rectangular region, because we are actually made and we have enclosed everything in a box. Every node will basically be a bounded rectangle region right.

So, every node of the tree corresponds to a rectangular region and that is what makes life easy for us that any time, we get a rectangular range query, I can quickly find out whether or not it has any partial overlap, no overlap or complete overlap because these are all my rectangle of a constant description. My nodes are also constant description. So, you can quickly determine whether it is a full overlap, partial overlap or no overlap. So, I can make those tests are constant time tests. Eventually, in the leaf node, we will have these points stored. We will see whether or not. We need to also store these points in holes. May or may not depend on what the algorithm is going to be like at this point we get a rectangle. So, I compare the rectangle with the region of the left. Well, it does not overlap. So, I do not need to proceed along this at all ok.

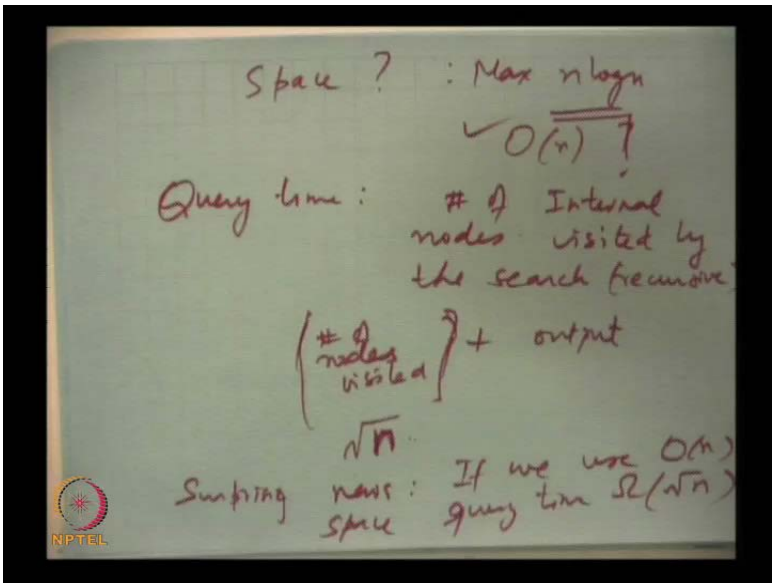
So, this rectangle does not proceed along this at all. It completely lies to the right of the median line. So, the search path will take is here next level. Unfortunately, we are not so lucky because it intersects both the up and down. So, we need to the search paths is like this gone. May be I have lost all the advantage. So, what happens if you keep branching on both sides? Eventually, you know that at the leaf level, we will have to deal with about order n nodes order. Let say every internal node, we need to a constant time check whether it is partial overlap, full overlap or low overlap right. Now, if this kind of branching continues. Let us say up to all the available, almost liquid the levels. Then, what you have? Suppose, if it has branching, you know all the way up to leaf level minus

1, then what happens? Every time it is a branch of 2, so every time when it is 2 time, let us say about log n minus 2 times. How much is this? It is almost in my 4 which means that we have lost whatever we are aiming for in terms of saving query time. I does not you know gone.

These are all claims that you are making. I am not convinced. We will have to analyze it. What I am saying is, well I mean, interval trees are interval trees. This is not an interval tree. This is by the way called a range search tree. It is a range search tree and it is a very special kind of a range search tree which is called as a K-d tree.

Why is it called a K-d tree? Here, the d stands for dimension and the K stands for the number of dimensions. So, in this particular case, what I have drawn is essentially a 2d tree in K dimensions. We are going to cycle through these coordinates. So, all the K coordinates. So, here we cycle through first patrician on x and then, y. When you have this K coordinate, it is unable to cycle through all the K coordinates. Then, go back to the original problem. Again, partition along the first coordinate, then second coordinate. All the K coordinates. So, it is called the K-d tree. In general, it is a very common and powerful data structure, but this is the basis of essentially that you are partitioning the points. You know you have a range that is going to build in search tree based on that and you go into 4 or follow certain search path depending on which of these nodes rectangle intersects with.

(Refer Slide Time: 51:26)



So, the real question here to analyze is, of course, what is the space each node occupies? Only constant phase because the region description is constant, but depending on whether or not I store the points also. I do not know if I store the points also. Then, every level I will store a point, then it will be n log n. So, I would say maximum n log n depending on what kind of, so a scheme you used, but then it may also be order n. You know if you are lucky with that, we will have to figure out this is a space.

More challenging is the query time. What is the query time really? The query time is the number of internal nodes visited by the search, the recursive search that is precisely the time of query. Finally, when you leave, reach the leaf node, we just either report the point or do not report the point. So, in terms of the output size, it is either going to report or not report. So, we are going to charge one for the reporting and not charged for not reporting. So, the plus output size kind of false immediately from here, but number of nodes visited is the fixed term.

So, how large is this term? What is your guess? Tell me. First, we will first look you know if you can guess it and then we will try to, if you have guessed it correctly, we will try to analyze it, log star 2 log n, wow. Any more guesses? More complicated functions, less complicated functions. No one thinks it is constant. Good because at least the search path will be taking at least one path. So, that is minimum of log n. Anyone willing to guess? Log star n. Now, log star n is 2 and it is too small, 2 to the power log star n. So, since we are running out of time, what I will do is, I will give you the answer. Let you think about it over night and tomorrow, I will complete the proof.

Some of you should know because you have done some course with me where I think, I may have discussed this, the last chance, log 3 by 2. Still stuck with logs n. Yeah n is certainly an upper bound for sure, cannot be more than n nodes n log n. No n log n is way. You will be finalized for that answer. I heard it. Yes root n, got it. So, you may think that root n is, I mean the reason may be, you did not try to guess root n because it is not consistence with our goals or poly log, but there is a cache that I will explain tomorrow.

So, it turns out that the space is order n that I do not need to store points in any of the intermediate nodes. I will be able to analyze where I will show that the number of nodes visited is no more than square root of n by any rectangle. This is almost the best possible.

What is the surprising part, surprising news is that if we use order n space varying time, one can show the lower bound is a very, you know rather unexpected result actually. It is very difficult to prove, we will even attempt. We will not attempt to prove it, but this is known. So, if we restrict ourselves to linear space, we are bound to encounter in the worst case is about square root of n.