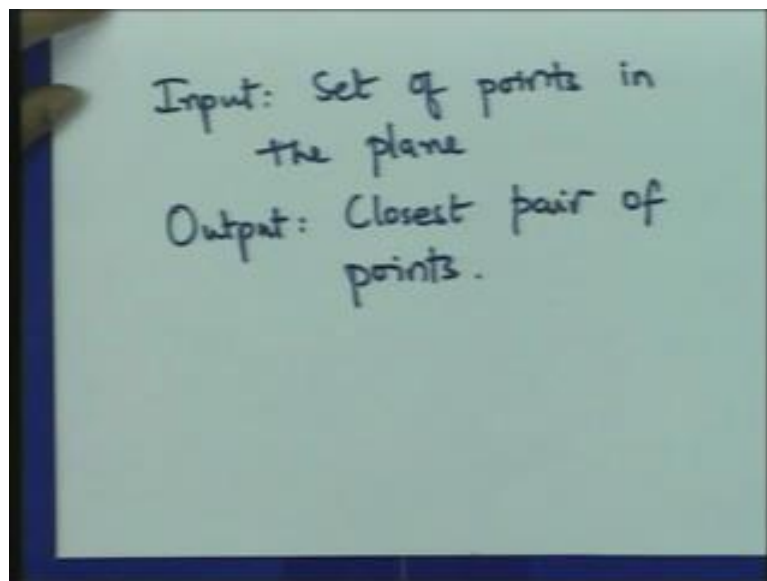


Design and Analysis of Algorithms
Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 9
Divide and Conquer - IV
Closest Pair

We will continue with our study of algorithm design techniques. We had looked at some examples or which we used this divide and conquer strategy. We also looked at sorting the strategy, applied to sorting to give a lower bound of $n \log n$ for certain sorting based algorithm. We continue along the same way. In fact we look at yet another problem which some meanable to divide and conquer. This problem comes from computational geometry. So, it has the basic is bit different, but the same strategies, algorithm design strategies applied even to field. So, the problem is this.

(Refer Slide Time: 01:54)



The input the set of points in the plane say the output I want the closest pair of points. So, given some points, I can compute the distance, we can compute the distance. And among these points, that you have given in the plane, you want to find the closest way. For instance, the points could be given with the x y coordinates, that what we will assume.

So, for each point, I give you the x and y coordinate. And actually there are n points and given these n points, with their x y coordinates. I want to find a pair of points, which are closest to each other. So, given any two points the distance; the distance is square root of $x_1 - x_2$ squared plus $y_1 - y_2$ square. So, the usual notion of distance that you have in the plane.

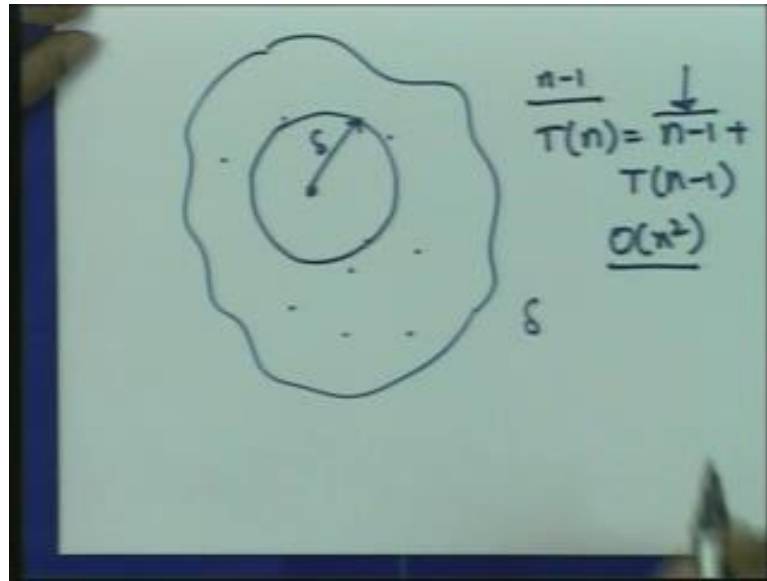
So, for each plane of points, there is a distance and among there are n choosable distances. So, between these pairs of points, if I have n points, there are n choosable distances. I want to pick the minimum distance. And the points, which actually witness this minimum distance. So, there is an obvious way to do it. It is to compute for each pair a distance, for each pair you compute the distance.

Now, you have n choose 2 distances and you find the minimum, so that takes n choose 2 time, can you do faster, can you do better. So, for instance, do you really have to compute all distances to find the closest pair. Well, that is the big question there, and the answer is no. If the answer is yes, then I guess I would not be discussing this problem here ((Refer Time: 04:30)). So, the answer is yes and in fact, we will do it much faster than, then n square.

So, somehow you do not really need to compute all pairs, distances of all pairs, that is the moral of the story. Let us see how we go about designing. Well, on the face of it, if you would think a bit. You start wonder, why should I need to compute every pair of distance. What can I mean, can I for instance, look at geometry of the points to certain distances, maybe there are many ways of thinking about it.

We should also be thinking of an inductive approach, which is... Supposing, I remove one point to compute a distance, shortest distance between the other points. Now, adding this point, can I get rid of some points, may be yes, may be no. I mean, so let us look at this approach. This will not be the approach finally follow, but this is something that I wanted to think about for every problem. So, that is the reason I wanted to go over this again. So, the classic sort of inductive approach to this any such problem is remove a point. The recurs some the rest, put the point back and see what you get, good.

(Refer Slide Time: 06:13)



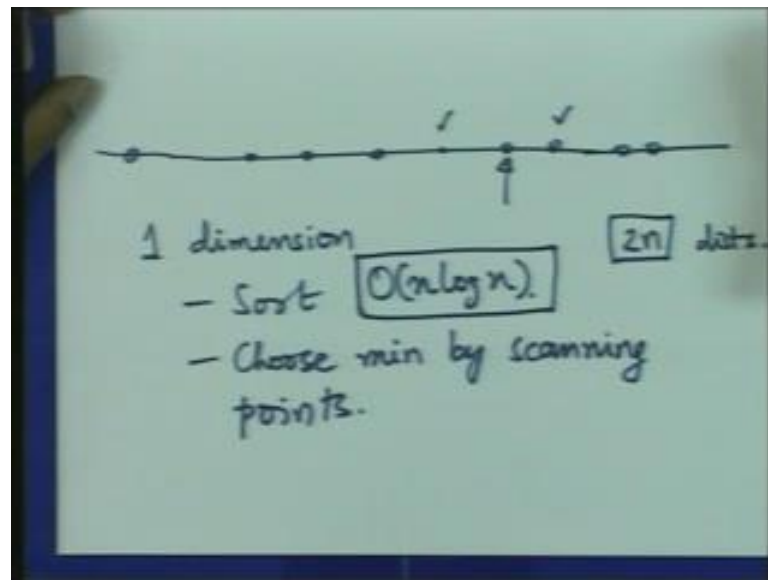
So, let us see this is the point I have to move, here the point set in this, let us say in this region. There are other points, this huge big star, this dot is what I have remove. Now, I find the smallest distance, now what can I do? I mean although I compute, what is distance. Well, it looks like, I mean the obvious thing is to compute distances from this to every other point. But, well that is not true way. You are back to the original to where you started.

Because, there are n minus 1, such other points and you will compute n minus 1 distances. And this recurrence you will see, will lead to in order n square solution, because your recurrence will be $T(n)$ is n minus 1 plus $T(n-1)$. This will lead to order n square, so you can check this. Well, can I get rid of some of these points now. If you could do the following, suppose in the minimum distance here, over the entire set, except this point this is δ .

The minimum distance, you have calculated recursively is δ . Then, you know that any point, the points that are actually in contention, are points in a circle of radius δ , around this point. Only points in this circle are in contention. The other points are well too far away. If you can quickly compute these points, then you are in business. I took n minus 1 times to extra time, but supposing I can do this much faster, somehow I do not know how.

But, if you can compute this much faster, then you are in good shape. Then, you can may be reduce this $n \log n$ somehow to something else. For instance, if you reduce this to $\log n$, your n really good shape. Then, maybe you can push this further, so your think is should be along these lines. How do I cut this $n \log n$.

(Refer Slide Time: 09:02)



So, before we go further, let us do this in one dimension. So, what is this? That mean, you are given a line and some points. You are given some points. And you want to find the closest pair along this line. You want to find the closest pair, how will you do this? Well, I think some of you should see that, you do not need to really calculate all the n chooseful distances.

There are n points, there are n choose to sort of distances this. May be you do not really need to do n chooseful, look at all n chooseful. So, how will you sort of do this, well let think to notice. In all these things, you need to notice something about the problem. Some property of the problem, that sort of pushes you up. That let us you do things faster.

So, in this case, well the thing to notice is that, supposing I am looking at this point, the only candidate points are the adjacent ones. These are the only candidate points. If for each of these points, I can identify these candidates. Then, I am done because, for each point I need only to check two distances. So, that leads to $2n$. For each point I need to check two distances. So, for n points I need to check $2n$ distances.

But, which 2 that is a big question. For this point, how do I find these two, which are neighbors in some sense. If I can do that, then just $2n$ distances, then well looks linear. Well, how do I find these two, may be just solved them, so in one dimension, sort. Well, once you sort them, look at them in increasing order of from left to right. And for each point look at the previous and then, the next one.

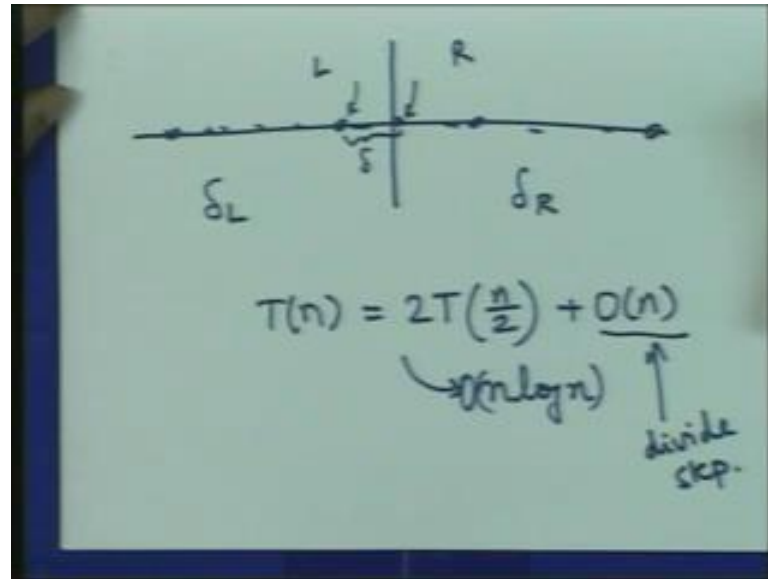
Compute these distances and choose the minimum. Then, you choose the minimum by scanning these points, sorting takes order $n \log n$. So, we started out with n chooseful distances, but really time taken is order $n \log n$. Sorting dominates this procedure and if I sort of first sort it and then, I do this then it takes the time taken is $n \log n$. I have reduced it from n^2 to $n \log n$. There are some morals in this story.

First one is you should always try a simpler problem. You are given this problem in two dimension. It always pays to first check out what the problem means in one dimension. You will learn a little bit, that is the first thing. And you at least now know that, may be it is possible to do it in less than n choosable time. Because, one dimension certainly you can do it in $n \log n$.

Question one asks now is about what about two dimension, can you do. Is there something like sorting that I can do, which will help me out. The answer is actually, yes and let us see how this is done, it is quite a smart algorithm. So, we will try and apply, divide and conquer approach, reasonably blindly. But as we proceed this fact we use some some trick that speed up ((Refer Time: 13:16)).

So, the first step is something you can think of, I want to divide the point into two parts. Find the minimum on the left part, find the minimum on the right part. Then, once you find the minimum, now I am going to use these minimums to compute the overall minimum. Will this help, will this not help, well will have to see. Before, we do this I think it is possibly illustrative to do this on the line. We first sorted in founded the minimum, that was $n \log n$. Let us see what this gives, this divide and conquer strategy gives on the line.

(Refer Slide Time: 14:06)



So, I am given these n points, can I get something, can I do something better. What would the divide and conquer strategy be, well divide the points into two parts. One on the left, one on the right and recurs on the both; and what? Once we get the minimum distance from this and that. Now, the only sort of thing that we have not check, is if the minimum distance between points on the left and point on the right.

I find the minimum here, I find the minimum there. And then, the only distance I need to now compute is the point on the left, the last point on the left. The first point on the right, find this distance and check this against the minimum, that I have computed. So, I found let us say, δ_L , δ_R and this is δ . So, I need to find this minimum. So, once I compute δ_L and δ_R and δ , I just check the minimum of these and return the minimum of these three.

I need to make sure, that I find this point on the left recursively. And I make find this point on the right recursively. So, again this, if I want to do by divide and conquer. Then, I need to strengthen in ((Refer Time: 15:39)), which is that not only do I find minimum distances. But, also to find the last point, right most point and the left most point. Once I do this, then I can sort of put these two things together to get a solution.

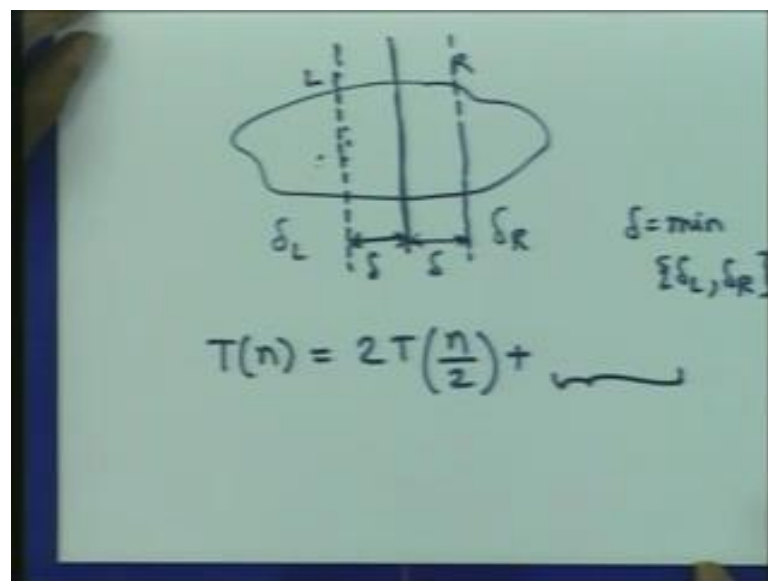
I do not want to get into with this in detail, because the problem here is again splitting the input into two parts. Now, how do you split this into two parts. Well, we need to split it into two equal halves, then you need two halves. Then, we need to find the median,

that takes linear time. So, if I look on the recurrence, it looks like T_n , this is $2T_{n/2}$ plus order n . And this we know is $n \log n$. The solution to this is order $n \log n$.

This is the time taken to find the median and divide, this is the divide step. So, this does not really use sorting in any sense, except to find the median. So, this is another way of doing this, using this plane, divided and conquered on these points on the line. Let us do this to points in the plane. You have given points on the plane and I am going to divide this into two parts. Let us say based on the x coordinate, the first $n/2$, the left most $n/2$ points, it will form one set.

The right most $n/2$ points will form the other set. I recursively find minimum distance, closest pair of points in the left, closest set of points in the right. And now I need to find, closest pair for the entire thing. And let see if this can be done fast. That is the goal of this lecture.

(Refer Slide Time: 17:50)



So, here let us see my points are points line this blob, divided into two parts. There are $n/2$ points here, so this is the left and that is the right. There are $n/2$ points, $n/2$ points here. Now, I find closest pair points in the left, closest pair of points from the right. Let us say, that δ_L is the closest, is the minimum distance on the left, δ_R minimum distance on the right. δ be minimum of δ_L and δ_R , this is the smaller of these two.

Now, I need to find closest pair. And I know that, if at all there are candidates or closest pair, it has to be that one point has to be on the left and one on the right. Any candidate pair has to be of this form. They cannot both be on the left and both on the right. One of them has to be on the left; and one of them has to be on the right. What else can be said, let us also sort of at the same time the recurrence in place. So, I am looking T_n , so T_n is the time. I have already done this plus what? Well, I would like this ideally to be.

If this is say order n , then this recurrence is $n \log n$, I mean good shape. So, this is the amount of work that I will need, to find the closest pair. One from this side and one from that side; where one point is from this side, one point is from that side. Suppose, I do it well the natural way is to just pick every point on the left, every point on the right. And sort of find the distances, for every pair. This could be as large as, well that could be n^2 .

There are n^2 points and n^2 points here, that gives rise to n^4 pairs. So, if I compute n^4 distances I am ((Refer Time: 19:58)), that is just too much. In fact, we will back to our usual n^2 . We will be, you can check that, you compute all distances, all pairs of distances, the n^2 chooseful distances. So, this is not what we want to do. So, how can we speed things ((Refer Time: 20:20)). This is the question, that you need to answer.

Now, at this point, it is left to your ingenuity. It is left to little bit of luck, but there are no clear. It is not as if at least I do not know of any thing that I can teach you, which you can use. This is left to just, your own intelligence and perseverance. You can try small problem, small examples etcetera, etcetera. But, the idea from this point onwards are completely new.

They are problem dependent, depends on the problem. And there are no general techniques, that I can tell you which you can use. There are no formulae's that you can apply. And this in fact, is the beauty of the speed, that it is not that I pump you full of ideas. And then, you are all ready to take on a problem. That is not the way to us. Each problem has a flavor of it is own.

And each candidate also has his own capability. And each person has his own capabilities. And is a good chance that, if you try this problem, you will come up with your own algorithm, your own passed algorithm. So, from this point onwards, I am going

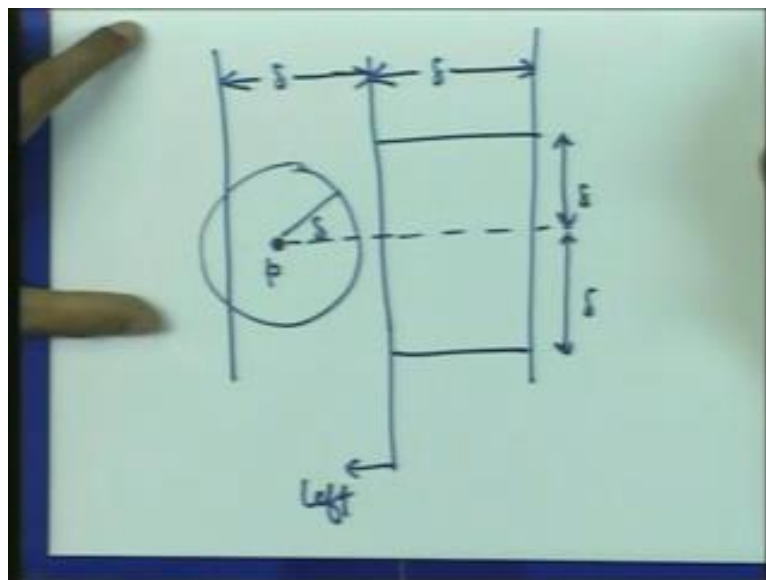
to tell you a few tricks by which we will reduce this. These observations are which I will make or we will find reasonably simple initially, but these we have to find on your own.

There is no teacher to teach you this, you have to learn it yourself. It just depends on your analytical skills, good. So, let us now get back to this problem. The first thing I am going to note is, where do these supposing I have these candidate points on the left. And where can they live, so let us look at this. So, this is my dividing line, dividing vertical line. Things are the left, where can they live, well they cannot, they all have to be, we draw a straight lines here.

They have to be within a distance δ of this middle point, similarly on this side. If a point lies here, then the distance between this point and any point on the right is greater than δ . If it lies on this side, this point and anything on the other side, this distance is greater than δ . So, I can forget about these points. So, the only points of interest to us are points in this small, on these small bands on either side of the middle line.

Each of these bands as width δ , these are the points of interest. That is the first observation, ((Refer Time: 23:19)) so far it is all fairly simple. Now, just this is not enough. The same problem could apply in the sense that, all n by 2 points on the left could lie in this band. And all n by 2 points on the right could lie, you know the other band. In which case we are again back to computing n^2 , roughly n^2 distance is the other smart thing. The intuition is this, so let us magnify this small band around the centre.

(Refer Slide Time: 24:00)



So, here my, this is the centre line and here are the two bands. This is δ and this is also δ . So, when I look to the left, if I take any point here, then I know that, if I draw a circle of radius δ , around this. There is no point inside this circle. The reason is this closest pair of points on the left, this is the left, this is δ . So, there is nothing closer than δ to this. There could be something at distance δ , that is fine. So, there is nothing closer than δ .

These points are actually spread, they cannot be a large number of points clustered together. They are all spread out nicely, distance δ . Similarly, the points on the right. We would like to use this fact, this is point number 1. That these points on both left and right are sort of spread apart. So, there cannot be too many points close by, somehow we need to use this. What else can be said, well let us look at this point on the left.

So, let us look at the point on the left, what about points on the right. Where can they live, let us draw this line. Well, I know that any candidates for this point will lie over this does not look like a square, but think of this as square. So, any candidates for p , must lie within either this square or this square. Please strain your imagination and imagine this to be a square. Then, any candidate for this point must lie, either in this square or that square.

They cannot be, if it is outside here, then the distance is greater than δ . In fact, distance to this line is greater than δ . So, this point is also greater than δ . So,

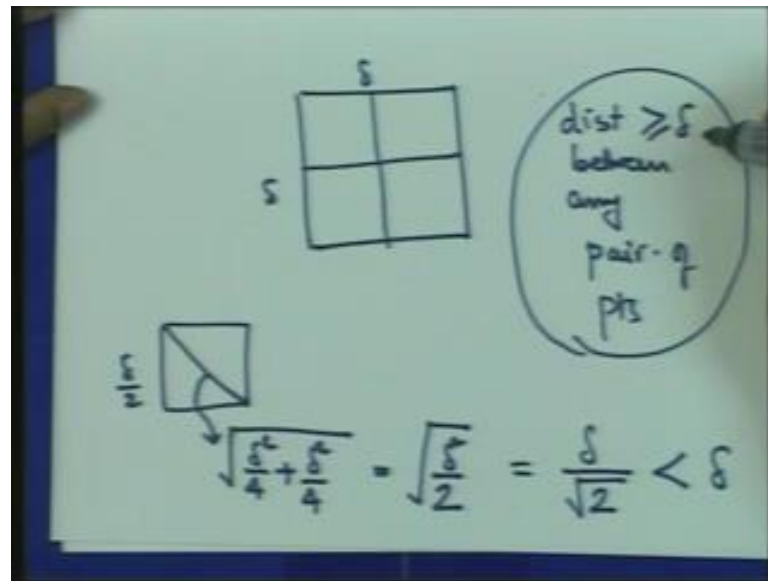
everything has to lie here, and that is the first. The second thing we know, that if I take any point here, then this precludes many points from appearing in region of size δ . If I have a point, I have a point in there.

Then, in a circle of radius δ no other point appears. So, there cannot be, I mean intuitively one feels that, there cannot be too many points inside each square. This is δ by δ square, if there are too many points inside this. There are bound to be two points which are close by. I cannot have every point as far away as δ . This is what we are going to use, but how do we use this.

The intuitively you feel that, there cannot be too many points inside the square. The reason is essentially this, that if I have a point, there is nothing within a distance δ , good. So, how do we put this intuition into practice. Well, we use a very simple principle, that you have heard many time. It seems very simple, but putting it to practice often requires some thought. The principle is called pigeon hole principle, well what does it say.

It says that, if there are $n + 1$ pigeons and there are n holes. And you stuff these pigeons into these holes, then at least one hole, must have two pigeons. There are more pigeons and holes. So, if you stuff these pigeons into holes; and at least one hole must contain two pigeons. Very simple principle, we all immediately understand the statement, but often very difficult to figure out where to apply and how to apply. Really nice problem, solutions based on this principle, which smart people have figured out. So, let us go back to this square problem.

(Refer Slide Time: 28:37)

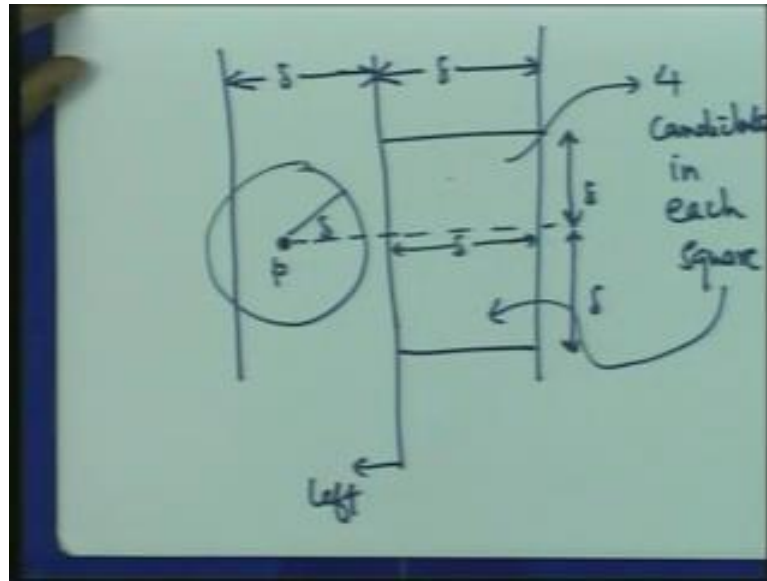


So, I have this square of size delta. And I know that, if I have any point in this square. Then, no other point is within a circle of radius; which means given any two points, the distance is at least delta between any pair. So, the distance between any pair of points in this square, is at least delta. How many points can I put in inside this square. Well, here is the trick, what you do is divide this into four parts. These will be our holes, you have to see where the pigeons are.

Well, what is the maximum distance inside the small square. Well, this small square as side length delta by 2. Maximum distance is along the diagonal, which is square root of delta square by 4 plus delta square by 4. So, this is the square root of delta square by 2. This twice this which is nothing but delta by square root 2, which is strictly less than delta which is what we want. So, if I take one of these small squares, then the maximum distance is delta, which means I cannot have two points sitting inside this small square.

If I take any two points, the distance between them is less than delta. So, at most one point can sit inside the small square, which means in the big square I cannot at most four points. If I say the distance between any pair of points is at least delta. Inside this big square, I can have at most four, this is the big trick, that we will use. So, this is the trick we will use, so let us get back to the original picture here.

(Refer Slide Time: 30:58)

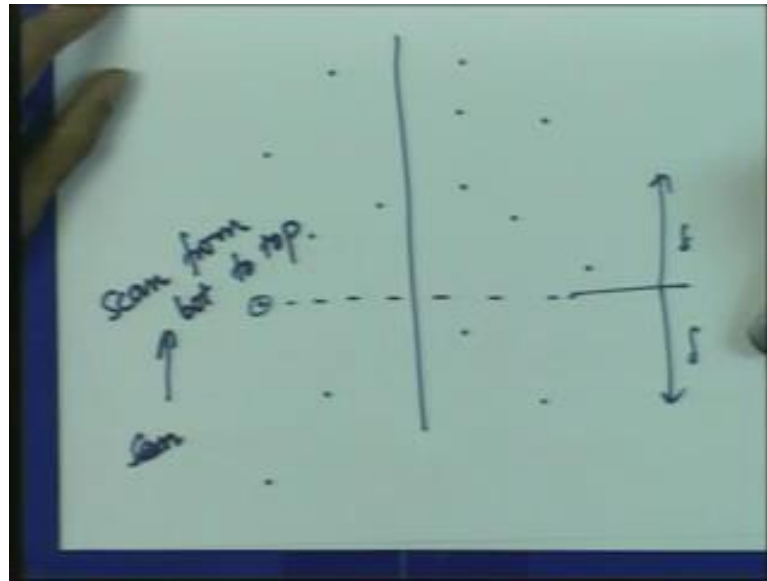


So, I have p and how many points on the right, do I have to really compare. Well, four inside this, there could be at most 4 candidates in each square, if at all. 4 or less than equal to 4, to be less. It looks like as of now, for each point I need to at most look at 8 other points and compute this much. And this certainly looks to be linear. For each point I need to compute at max, let us say, 8 other distances. Then 8, 10 distances is what I need to compute and my recurrence I will have $2Tn$ plus by 2 plus Tn ; and this is $n \log n$.

How will we do this, we still not done. I mean, we have all the ideas in place now, it is implementation. Smaller implementation details are we need to worry about. For instance, for each point on the left, how do you compute these 8 points, you have to do this fast remember. You cannot take a lot of time to. We look at all the points for each points, then your sort, because that is n^2 already. But, again looking at this picture, you should again get ideas.

So, let us go back to this picture. So, here is the point, the 8 points are somehow located in the neighborhood of this point. They are at distance δ from here δ from here δ . These are the points that I need to and this ((Refer Time: 32:44)) I need to pick up. So, let me put up another picture and after this picture hopefully the algorithm will be clear.

(Refer Slide Time: 32:54)



So, here is my centre line and here are these various points the left in the right. Here are the various points. If I take any point on this side, if I consider this. Then, I only need to look at a window size to delta, delta on this side, delta on that side. This is what I need to do? As I go up, if I look at this point this window shifts up, it somehow if I can scan them upwards, in this order, scan from bottom top.

If I can scan this points from bottom to top, somehow this window also keeps you think from bottom to top. And it again looks like I can do this efficiently, all we need is these things altered by y coordinates. If I have these points altered by y coordinates, then I can scan the left side from bottom to top, the right side from bottom to top. And I can maintain this window easily, this is the last trick in the algorithm. There is a detailed left. So, if I have these points sorted by the y coordinates. Then this is what I do? I start looking at the points on the left in the narrow band.

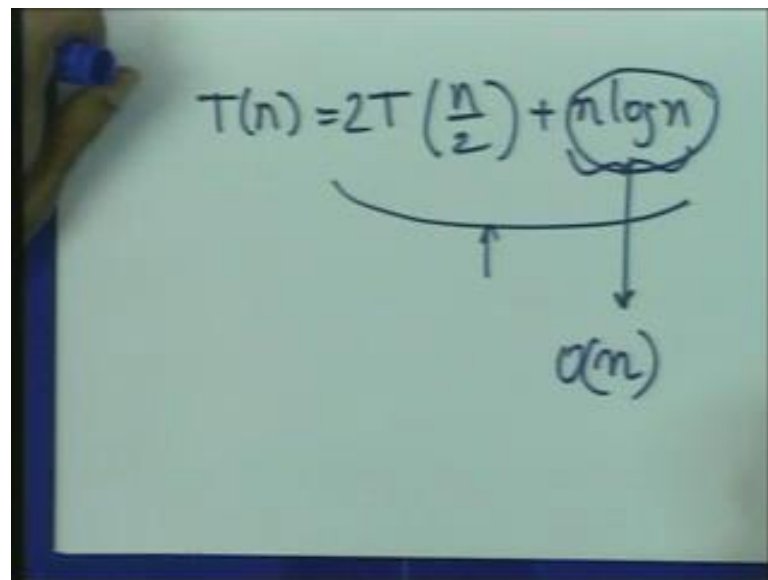
I look at these points, look at them in increasing y coordinates. And for each point I have I also maintain this band of this sort of window on the right. So, when I look at a point, I have a corresponding window on the right. When I move to the next point I correspondingly move them window up. This is all I do, so somehow I need to move this window up. And you can see that, it is not it should not be too difficult, to move the window up.

And the time that it takes is linear with and the very similar to the merge sort idea. The window moves up each time. On the left each time, you know a pointer moves up, on the right side the window could move up, but it can move up only n time. The window can move up only n time, because there are only n points. So, at least the movement, the number of times the pointer here moves or pointer here moves, that is linear.

And once you have fixed a window and a point on the left, I need to compute distances in this window. And I know, because of the geometry that there are at most eight points in this window, four on top four on bottom. There are at most eight points I need to compute these eight distances and choose the minimum that is it.

So, let us put all this together and see what we get? That is before we do that, I just want to point out one thing. So, we need the points sorted on to five coordinate. Now, if we do it on the fly, once we partitioned now if we look at points in the each band. And now we sort these points along the y coordinate you take $n \log n$ time sorting takes $n \log n$ time. And then, the recurrence looks like this.

(Refer Slide Time: 36:50)



A photograph of a whiteboard with a handwritten recurrence relation. The equation is $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$. The term $n \log n$ is circled. A curved line with an upward-pointing arrow is drawn under the entire equation. A downward-pointing arrow originates from the circled $n \log n$ term and points to the expression $\alpha(n)$ written below it.

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

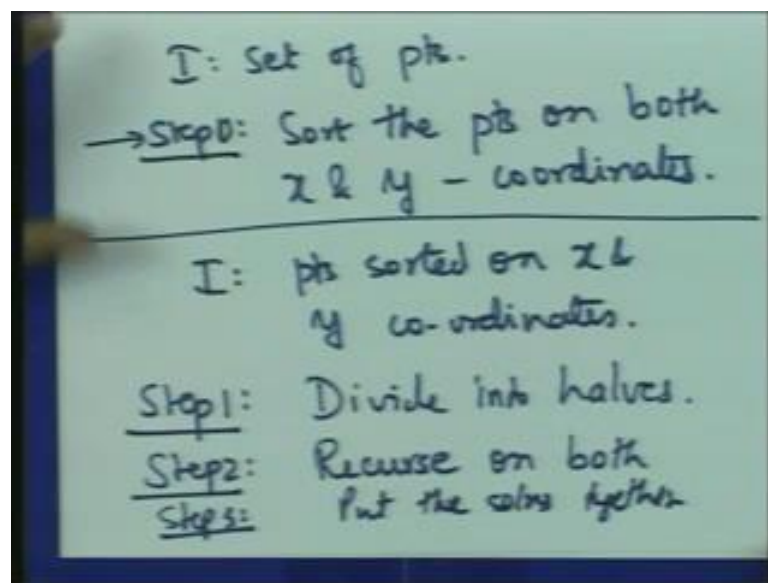
$\alpha(n)$

Your recurrence would look like $T(n)$ equals $T(n/2)$. Well, $2T(n/2)$ plus $n \log n$, because this sorting will dominate. Once, we sort we can go over them in linear time. But, this sorting will dominate and take $n \log n$ time. This the solution to this recurrence is still not $n \log n$, the solution you can check that the time to find what the solution recurrence is $n \log n$ square and it is not $n \log$, we would like to be positive.

In fact, you would like to get rid of this $n \log n$ and replace this with order n . So, that overall we get $n \log n$. And again the trick is we tree sort the points in the x and y coordinates. We do not do it at each recursive call. So, all I want is the point sorted on y coordinate. Suppose, I can do it once in for all initially. And use this effectively and that is what I want. To this $n \log n$ for sorting along both x and y coordinates I do right in the beginning, that is the one time cost I do not repeat this in each literature, it is a onetime cost I sort them I am done with.

And now I have this sorted order each time I make a recursive call. So, each time I make a recursive call I have this sorted order in place. In which case, perhaps I can put order n instead of $n \log n$ and you get order $n \log n$ in total time. So, let us look at this algorithm, now we have I have told you all the pieces, we just have to put them together and analyze the algorithm and see how much time it takes.

(Refer Slide Time: 38:52)



We first sort to, so input a set of points. First step 1, sort them on both x and y coordinates. At this point I would like to make an assumption, which is that each point has distinct every point has a distinct x coordinate. And every point has a distinct y coordinate, which means if I take any vertical line at most one point lies on it. If you take any horizontal line at most one point lie on that. This is not really a big assumption. It will not it affect us for instance what I could do is take the initial set of points.

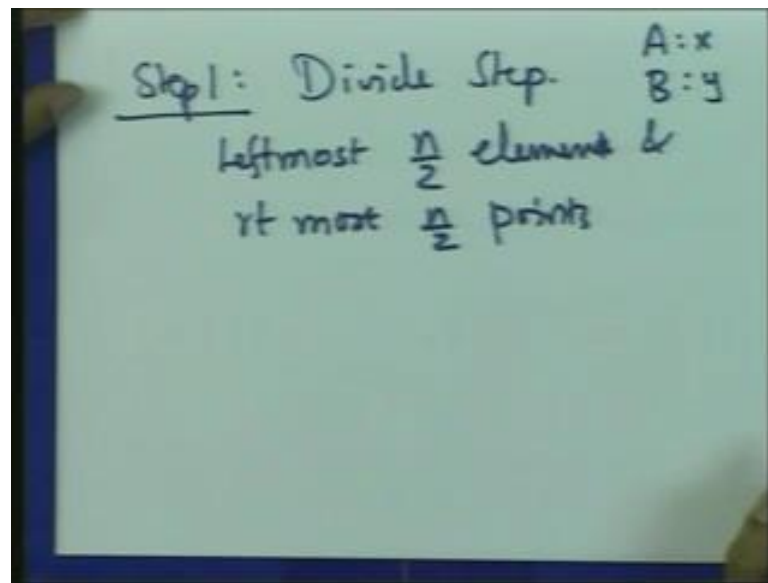
Now, perturb each point slightly, add a small Epsilon to each point it sort of move them around a little bit. So, that the minimum distance does not change, these perturbations are very, very small; these perturbations are very small. The minimum distance to change was a point does not change. The points participate in the minimum distance will that does not change.

While this perturbation, if I make sure that if I take any vertical line two points do not lie on this vertical line. Actually the algorithm I give right now, will assume that you know no two points have the same x coordinate or y coordinate. You can, then try and modify this algorithm to work even in the case. There are points on with the same ((Refer Time: 40:36)). So, I sort the points on both x and y co-ordinates.

So, I have two arrays the first array where points are sorted on x coordinates. Second where points are sorted on y coordinates. This is done right at the beginning I am not going to do this recursively. Now, the actual procedure starts. So, the input to this procedure, the input is points sorted on x and y co-ordinates. Points sorted on x coordinates, same points sorted on y coordinates. So, I have two sorted arrays. This is the input, what I do the first step, let me call this step 0. This is the initialization phase.

Step 1 here is divide, step 2 recurse on both, divide into halves. And step 3 put them back. Look at these two solutions and now look at these points and find the minimum and ((Refer Time: 42:06)) broadly these are the three steps. So, let us expand on all these steps, here is step 1.

(Refer Slide Time: 42:19)

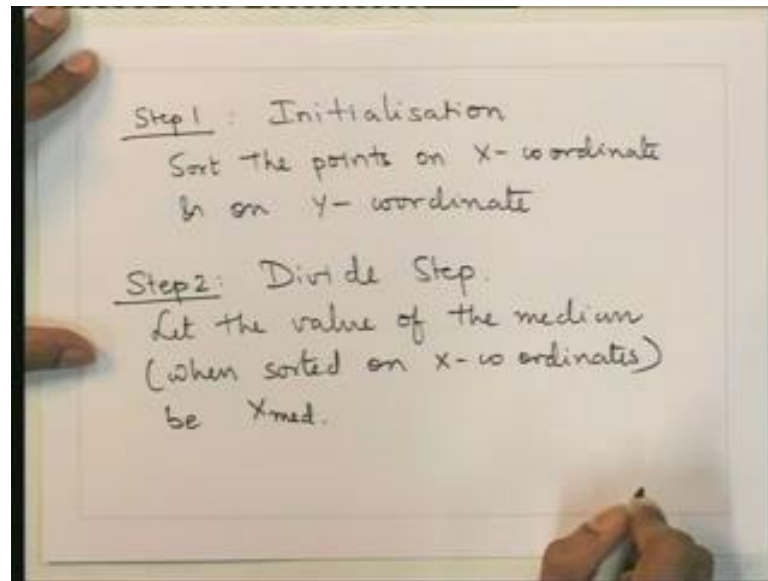


Step 1 is the divide step, this is the divide step. Well, I have the point sorted on x coordinates. So, I can find the left most, so since points are sorted on x coordinates, find the left most n by 2 elements and right most n by 2 points, these are points. This sort this give you the two points, the two sets. Now, I need to do this even for the points sorted on y coordinates. So, I have two arrays remember.

So, let me call them, let us say A and B. So, A is sorted on x coordinate, B is sorted on Y coordinate splitting A is easy, leftmost n by 2 elements and right most n by 2 points ((Refer Time: 43:22)). So, this algorithm that we have just seen is perhaps as you realize is more complicated, than what we have seen so far. And we seen all the pieces actually I have given you all the details.

What they are going to now is summarize the whole thing, put them together. And then, right out the recurrence, which will not be difficult once we put them on the steps. And once we do that, we will see that this actually runs in $n \log n$ time. So, let us summarize and write down this algorithm in one piece.

(Refer Slide Time: 44:06)



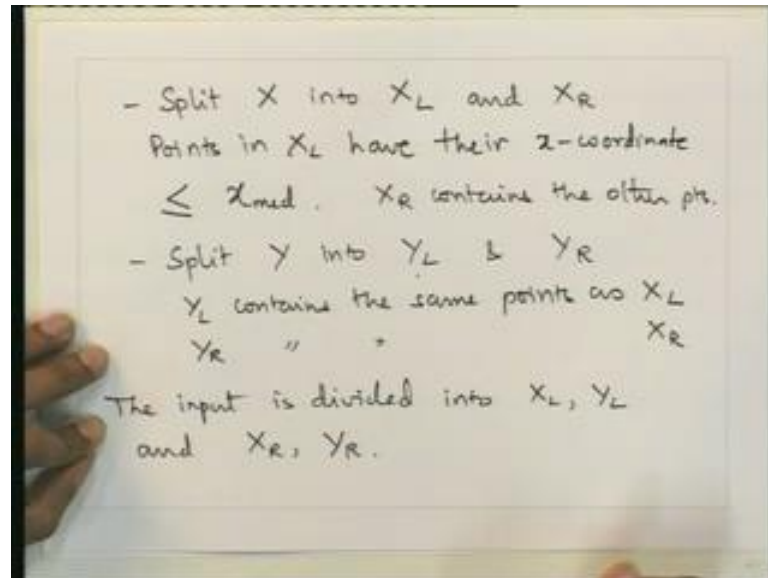
The first step is initialization, it is initialization. And here we sort the points on x coordinates and on y coordinate. So, given the set of input points we form two lists. So, we have two arrays. One array which will store the points in increasing order of x coordinates. The other array will store the points in increasing order of y coordinates. And this will be the input to the procedure.

This initialization is done before we start the algorithm. And each recursive step will pass down this information. So, the next step is the divide step, so it is step 2, it is divide step. So, here we first look at the median. So, let the value of the median, when sorted on x coordinates, let us say x median. So, now we split the input into two parts, points to the left of x median and points to the right of x median. And then, we would like to recurse on these two parts.

To do this, we have to split both the arrays into two parts, both the arrays which sorted on x coordinates and the array which is sorted on y coordinates. The x coordinate business is easy, because you know where the median is right it is n half the element. So, you just split the array into two parts in a very natural rate. For the y coordinates what you need to do is go through the entire array. Look at each point, look at it is x coordinate and then put it into an appropriate array.

So, now let us say these two arrays, where it is points are x coordinate. So, these two arrays are called, let us say capital X and capital Y. So, split let me write this in a new paper.

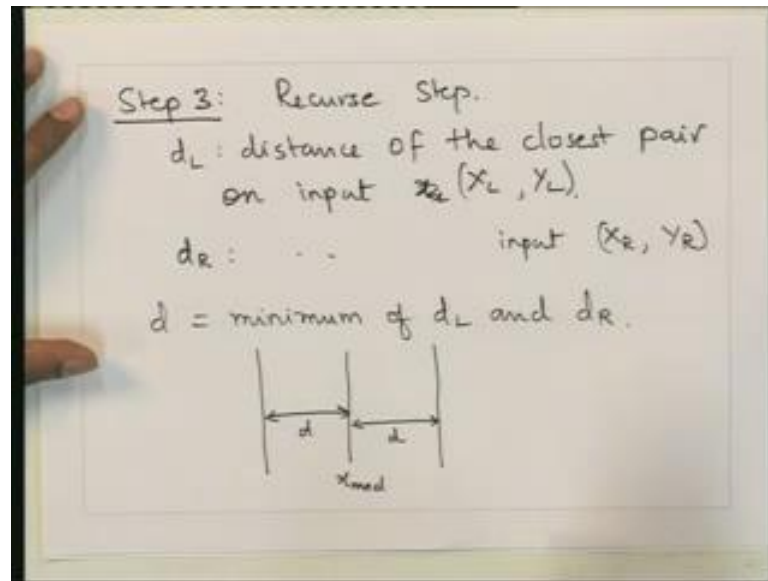
(Refer Slide Time: 47:18)



So, split X into X_L for X left and X_R points in X_L have their x coordinate less than equal to x_{median} , the median value. And the others are in X_R , the other points X_R contains the other points. Similarly, split Y into Y_L and Y_R . Remember, Y sorted on Y coordinates, Y_L and Y_R will also be sorted on Y coordinates. Y_L contains the same points as X_L only sorted in Y coordinates. Similarly, Y_R . So, Y_R contains the same points as X_R

So, this actually divides the input into two parts. Initially you had this sorted arrays X and Y. And now you have created X L, X R, Y L, Y R. The next step is the recurse step. So, let us write that, so pass. So, the input is divided into two parts X L, Y L and X R, Y R. These are the same set of points, in X L they sorted in increasing X coordinates and Y L they sorted in increasing Y coordinates. So, here is the recurse step. So, you recurse on these two, when you get the shortest the distance of the shortest the closest pair on both sides, on either sides of x median.

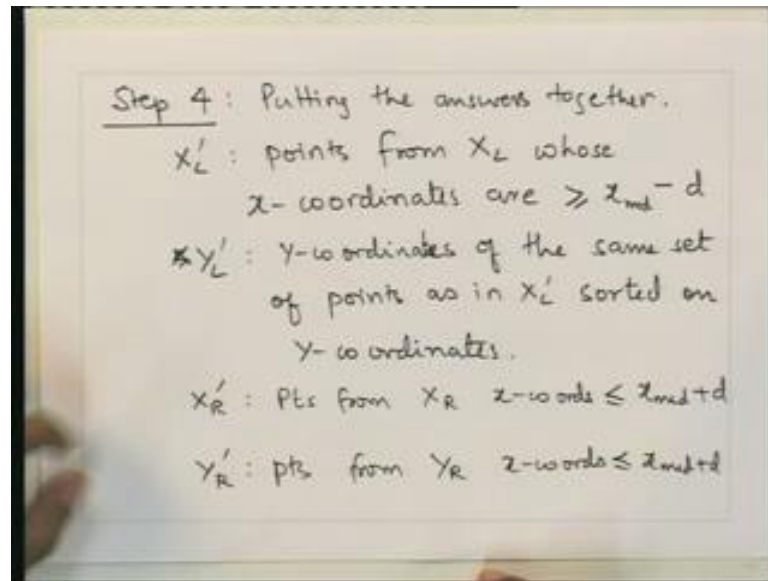
(Refer Slide Time: 49:56)



So, this is the recurse step. So, step 3 is the recurse step. So, what you get is let us say d_L is the distance of the closest pair on input capital X_L capital Y_L . Similarly, d_R is the distance of the closest pair on the input X_R, Y_R . On input X_R, Y_R the distance of the closest pair on the right hand side is d_R . The one on the left hand side is d_L . So, now you look at these two distances d_L and d_R and d compute d as the minimum of d_L and d_R . So, that is the next step, where you compute the minimum of d_L and d_R .

Now, let me quickly review what we have to do next. Around x_{median} , we look at a band of size d on both sides. So, let me draw a picture. So, here is x_{median} around this on both sides at a distance of d , we look at the band. This is the on the left side, that is on the right side. Now, we look at points on these two sides. And we know that, if you need to find points which are closer than d . Then, one point should come from the left. One point should come from the right and they must be in this band. So, let us first prove X_L, Y_L, X_R, Y_R to only points in these bands.

(Refer Slide Time: 52:27)

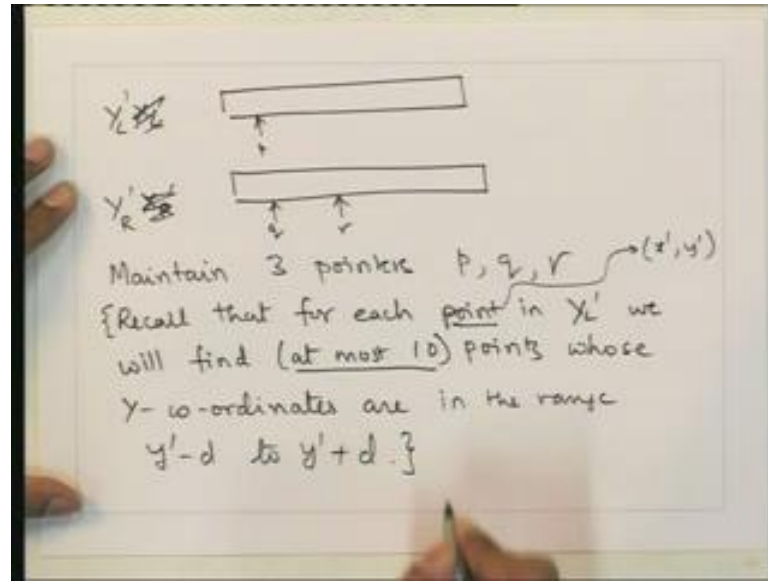


So, step 4 is putting them together, is putting the answers together, so that is the next step. So, X_L prime will be points from X_L whose x coordinates are, well greater than or equal to x median minus d . So, just look at this figure ((Refer Time: 53:13)). This is x median, we need points in this range. So, any point will have it is x coordinate at most x median minus d , so that is step 4. So, similarly Y_L prime, so the Y_L prime same y coordinates of the same set of points, as in X_L prime sorted on y coordinates.

So, you get these by pruning X_L and Y_L . So, you look at X_L and Y_L and only look at these points in the band and you prove, remove some of those points. So, you are left with X_L prime and Y_L prime. Do a similar thing for right hand side, so you get X_R prime and Y_R prime. So, X_R prime will be points from X_R whose x coordinates are. So, these are points from X_R with x coordinates, which are less than equal to x median plus d . These are points from Y_R with x coordinates less than equal to x median plus d .

So, these are the same set of points, only X_R prime is sorted on x coordinates; Y_R prime is sorted on y coordinates. Our focus from now will be on Y_L prime and Y_R prime. We will only look at these points from increasing y coordinates. So, we are still into step 4. So, let me recall what we do now. So, here are the points let us say we have X_R prime and Y_R prime, X_L prime and Y_L prime.

(Refer Slide Time: 55:28)

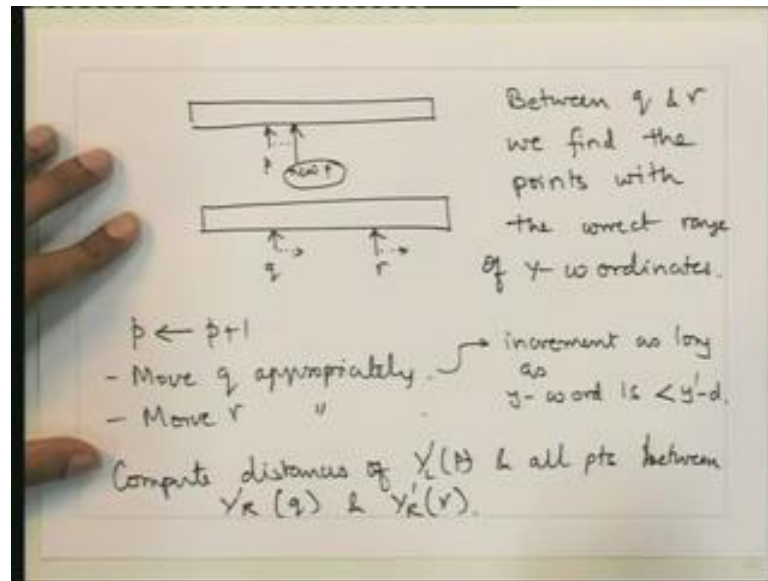


So, I look at, so here is my array Y_L' and here is my array, let us say Y_R' . These arrays could have different sizes, because we have proved points from Y_L' and Y_R' . This should be Y_L' prime, because you want to look at the points and increasing y coordinates. So, Y_L' prime and Y_R' prime are what we are concerned with. So, maintain three pointers, let us say p, q and r .

p will point to something there, that is p, q and r will be pointers to Y_R' prime. So, let me quickly recall what we want to do. For each point here, so here is what we are going to do. So, recall that, for each point in Y_L' prime, we will find at most 10 points. This at most 10 follows from a discussion that we did earlier. That in a square with 5 points, there will be two who are close as half the diagonal.

So, we will find at most 10 points whose y coordinates are, so supposing this point is, so for each point, let us say this point is some x' prime, Y' coordinates are in the range $y' - d$ to $y' + d$. So, we find these points and then, compute the distance from x' prime y' prime to these points. And this will add 10 more distances. We do it for every point in this array. Once you do that among all these distances, 10 times the size of number of points in Y_L' prime, we compute the minimum. That is our goal.

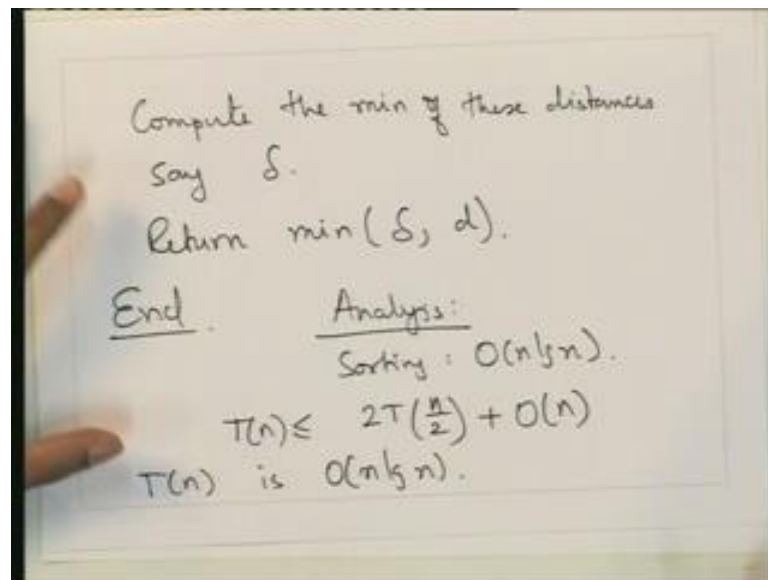
(Refer Slide Time: 58:17)



So, a generic step is supposing, so a generic step is this, so I have... Let us say I have computed up to this point p is done, here is the letters q and r . So, between q and r , we find the points with the correct range of y coordinates. So, supposing we have done up to this. Now, in a generic step, I will raise p by 1, p is p plus 1, so this moves one step, now this is your new p . So, now correspondingly q will move and r will move. So, the next step is move q appropriately, then move r appropriately.

What do I mean by appropriately, you increment q till, if this value is not in the range y prime minus d to y prime plus d , which means it is smaller than y prime minus d . So, increment as long as y coordinate is less than y prime minus d . And similarly you move r , as long as it is less than y prime minus d . Finally, when you do this, then you will have these two values for the next value of d . And now you compute the distances. Then, compute the distances of $Y'_L(p)$ and all points between $Y'_R(q)$ and $Y'_R(r)$ small r . So, then this is the generic step in the loop, put this in a loop. And once you are done, you found these distances, compute the minimum. So, the last step is compute the minimum of these distances.

(Refer Slide Time: 61:08)



So, say it is some delta, then return min of delta and d. And that is, ends the algorithm. Now, if you look at the analysis, then there is an initial. So analysis, so the sorting initially is order $n \log n$. There are two recursive calls, so that is $2T(n/2)$. And then, if you look at the whole analysis, the total time is $O(n)$. So, we can quickly do this, while splitting does not take time. The only problem is this pointer manipulation. ((Refer Time: 62:13)) In this pointer manipulation, how many times this p, q and r incremented.

Well p can be incremented at most n times. Q can be incremented at most n times. R can be incremented at most n times, so this is order n. The total number of distances we know, we calculate is at most order n. So, that is why this is order n and $T(n)$ satisfies this recurrence. And the solution is $T(n)$ is $O(n \log n)$. So, this is the analysis of this algorithm, it is pretty simple, and this ends our discussion on divide and conquer.

Thank you.