Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Module No. # 03

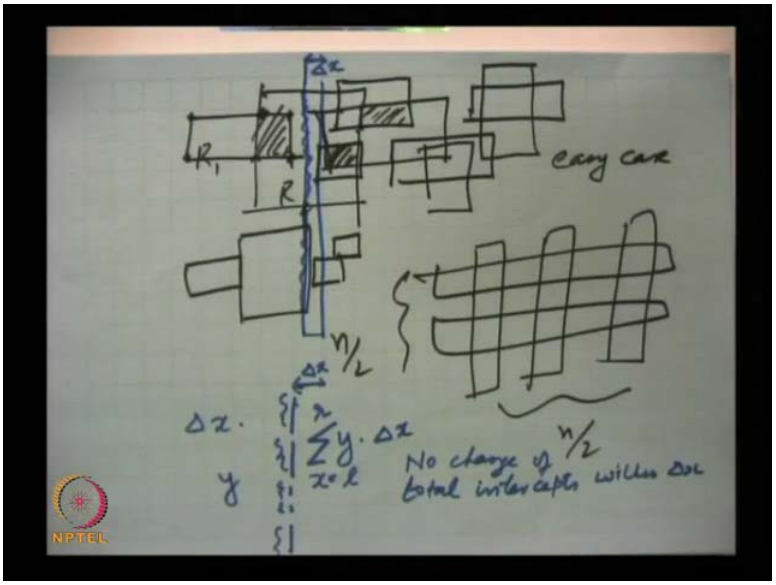
The Plane Sweep Technique and applications

Lecture No. # 03 ((6))

Line Sweep (Contd.) Area of Union of Rectangles.

So, welcome to lecture six of computational geometry. So, we will continue with our example that we started last time on computing the area of union of rectangles, isothetic rectangles that is axis parallel rectangles and again the technique that we are using the main technique that we are using is the line sweep.

(Refer Slide Time: 00:49)



You look at this example where the number of overlapping rectangles, this situation of course is more complicated than the situation where the rectangles are non-overlapping in which case, we simply just add compute the area of individual rectangles and add them up. As we had observed last time, we can try to reduce this situation where we are overlapping rectangles to the situation where its non-overlapping, we are actually trying

to compute all the intersections. So, if you can compute all the intersections, then we can think about them.

So, these intersecting rectangles can become the smallest intersecting rectangles which do not have any further intersections they become your disjoint set of rectangles and they there by again you can compute the total area by just summing up the areas of these rectangles. Unfortunately if you try to do that then we will have to compute these intersections and there can be lots of intersections namely you know if is this is the case, they are close to what n square intersections so, then you chances of a sub quadratic algorithm is no longer there right. So, we would ideally like compute this faster than you know something like n square times, where that is we do not we want to avoid computing this intersections.

So, what is the basic idea that we started pursuing? So, again if you want to use line sweep, and as we start from the left most rectangle and as we sweep across at any point of time, we maintain the intersection of the rectangles with this sweep line and if we look at this you know delta x. So, this infinitesimal that I am sweeping, whatever is the total area that is being swept by this line, we can write this as you know summation of.

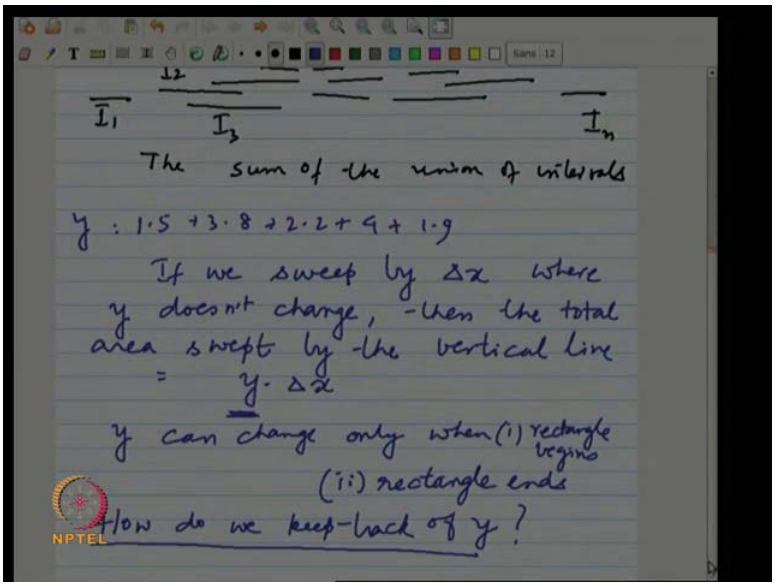
So, y is the total intercept of the rectangles with this vertical line and therefore this infinitesimal area the rectangular area would be y times delta x and so, if you sum it of x is equal to you know the left most rectangle; let us call it r and r let us call it l and r left and right most, then you know it is it is basically the measure of the union of the rectangles.

So, this y that we are saying multiplied by delta x, we can write this only if this y which is intercept of the rectangle. So, here in this line that intercept is essentially this plus this. So, as we sweep past we do not want this y to change. So, our stopping point so the event points we will those where we will be those y x coordinates such that there is no change in the intercept and then we can write this equation and thereby we can compute the total area.

So, this actually introduces a problem which is actually you know 1 dimensional in nature. It is the total intercept of the rectangles with this vertical line. In other words, so, I will just sort of rotate the axis. So, what we are now trying to do is given a set of

intervals, they are actually they all along the x axis I just spread them apart so that you can they are visible.

(Refer Slide Time: 04:23)



So, I have a set of intervals and I want to compute, I will just change this example little bit, create more gaps actually. So, we have this set of intervals, let us call them i 1, i 2, i 3 etcetera up to some i 7. So, we are given this intervals along x axis of course, in the actual problem these intervals around along y axis and we want to compute the sum of the union of intervals namely in this particular case, the sum would be where this is covered. So, I will just use another color. So, this part is covered and this is let us say sum some number you know say 1.5 for something and then this entire thing is covered by at least 1 interval and there is this again some gap. Then again we have this and this and this. So, if I had actually not drawn this interval separately in the y direction, this is what it would have look like and what I am interested in is a sum.

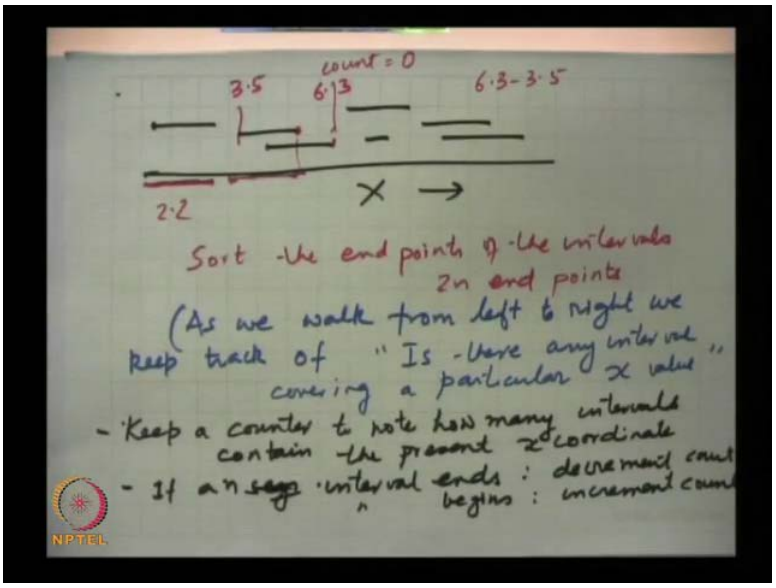
So, may be this is 1.5 and this is a 3.8 and this is 2.2 you know something like 4 may be something like you know 1.9. So, then I simply add this up these are disjoint intervals and whatever is a sum that is what I mean by the sum of the union of the intervals. So, when we are talking about computing the area of the rectangles using this plane sweep method, we have now actually reduce the dimension to a 1 dimension problem because we are only looking at the intercept of the rectangles with the with a line.

So, now, we can only look at these intervals. These intervals are nothing but you can think about them like the projection of the rectangles on the y axis. In this case of course, you know I gave turned the coordinates around it is x axis fine. So, once we know this and this is essentially is our y you know this 1.5 plus 3.8 plus 2.2 plus 4 plus 4.9; is what is y is equal to. And if we sweep by the amount delta x, if we sweep by delta x where y does not change, then the total area swept by the horizontal by the vertical line equals you know whatever that whatever the sum is y times delta x. That is all I am saying.

So, we have now reduce the problem of computing the area of rectangles to one that where we want to compute this keep track of this y where y do not change and y only changes, y will change or let us say y can change (( )) y can change only when one rectangle begins and two rectangle ends. And unlike our segment intersection case, both these events are known to us right from the beginning because rectangles are given to us we are not trying to compute anything that is related to the intersection of the rectangles.

We are only looking at the beginning that is the left boundary of the rectangle and the right boundary of the rectangle that is all. These are well known to us in advance. So, the actual the work or actual algorithm focuses around computing this y. So, how do we keep track of y. This is essentially what it comes down to know which is as a just pointed before, you know it is a it is set of intervals and what is the sum of the union of the intervals. So, how do we keep track of this? What would be the simplest way of doing it and given a set of intervals, given a set of intervals something like this.

(Refer Slide Time: 11:11)



So, how should you keep track of the sum of these intervals given that two intervals can overlap. (( )) Yeah I think someone split that. So, we can use another sweep kind of technique which is see when I started drawing these intervals, I was actually drawing it in kind of sorted fashion and left to right. I took the left the left most sorry the left most end point, do this interval and I shifted my pen and drew the next interval which means that the next left most point and I drew it in that order and actually this is also the method by which I can actually keep track of the of the sum of the total length covered by these intervals.

So, this could be something. So, as we start sweeping from again if I start sweeping from this is the 1 dimensional problem, but you know actually we just start walking along the x axis (( )) x axis we can simply start walking. So, as long as we pick the left most interval. So, as we go on. So, we keep track of the fact that we picked up some interval.

And if we first solve what do we do we first solve the end point of the intervals. So, let us say we sort the end points of the intervals. So, there are two end points to be sorted 2 n end points. I do that and then I start walking from the left most left most interval, the left most end point of the interval and as we walk you know I have to keep track of the fact the at any given point of time, is there any interval covering that x axis. So, as we walk along this x axis. So, I keep walking because this interval has not ended.

So, we also keep track of use another color as we walk from left to right, we keep track of “is there any interval covering a particular x value.” So, as I walk across so, we walk across so I can continue because that I have not encountered the right end, but it means that there must be some interval covering every bit of this journey, whatever this movement from left to right. Now of course, what this segment ends. The movement it ends here there is a gap. So, there is nothing that is covering it. So, whatever I have covered till now I can make a note of that, maybe it is there is say some 1 point either 2.2 or something that. So, some where you know I am going to count that I have swept 2.2 and then at this juncture, there is no interval covering this.

So, I have to now jump to the next end point which is here again I start walking great I note down this whatever this left end point whatever that x coordinate is. So, again I start walking of course, we do not walk continuously, you only walk to the next event point which must be the end point of some segment. So, here it happens to be the staring end point of another segment great what it means is that. So, as we walk we not only keep track of fact that there is an interval covering it, we may also want to keep track of the fact that so why I have encountered this, but then there is still another segment covering as I walk fast that. So, what should be also keep track off.

(( ))

Number of intervals covering any given x coordinate. So that we always keep a count. (( )) whenever we whenever we encounter the end point of any interval before we do something we encounter an end point.

See stack is a very you know it is a data structure is only in the last in first out. These intervals may not have the last in first out kind of property. In intervals two intervals, let it can be any kind of relation.

(( )) when whenever you encounter given point, if it is a begin of some segment. So, what is that you are going to push and pop. If the stack is empty you do not walk I mean. No, I am not talking about the stack. I am just talking about counter, a simple counter we will do or else changes keep a counter to note how many intervals contain the present x coordinate. So, then if a segment ends sorry an interval (( )) word interval sorry and an interval ends, decrement count interval begins: increment count. So, this is simple thing about count and we must also keep another kind of a global counter where we are going

to add how much x coordinate, what is the measure of the x coordinate that is being covered. So, only when the count reaches 0…

So, here for instance we have started from this left end point. So, we make a note of this x coordinate suppose it is some 3.5. So, we know that the last end point that you started with for you know when things actually covered by some interval to wherever this ends, the count also becomes 0, the count is 0 at here. So, count equal to 0. So, when the count equal to 0, we also note what is the x coordinate of the (( )) 0 may be it is in 6.3 which means that we have swept 6.3 minus 3.5 and this must be added to the total measure that we have encountered so far.

Any doubts? This is the most straight forward way that you can prove it right. So, what is the overall sort complexity that we are encountering in the process? So, end of the whole thing you should give the correct measure of the union of the intervals.

(( )) Well order n you know without counting the sorting the sorting cost right. So, if things are sorted, then you know its ordering we will not doing anything any work, every event point we are just increment or decrement the counter checking that the count is 0, if the count is 0 then you know you must do the subtraction and keep it a side, if count is not 0 we just continue. So, this is nothing but actually the 1 dimensional version of the area problem. Here we are computing the measure of the union of the intervals where we are actually computing the actually problem was compute the measure of the area of the union of the rectangles.

So, using the line sweep method, we have you know somehow managed to you know reduce the dimension which is actually, let us say made the problem at least more easier or more straight forward and then we put these things together that you know. So, we know how to actually at any in the original problem, let me go back to the original problem.

In the original problem actually we are sweeping a vertical line, at any x coordinate just by the previous approach, we can compute the total intercept of the rectangles with this line and then we can keep sweeping till we encounter the next event point which is either beginning of the rectangle or the end of the rectangle. And then multiply whatever distance we have travelled that multiplied by the total intercept which is constant because

it cannot change between two event points and then we eventually sum of everything and we should get the right area.

(( )) So, what is being observed that the total complexity should be n square. Why is the total complexity n square? So, let me just note what we noted essentially is that, to compute y, we solve the 1 dimensional problem i is the sum of the union of intervals. Again by you know some kind of, by line sweep method let us say. So, we noted that this state's, you know if we if we if we looking at all these instances as independent problems to compute this intercept with the y axis, this could take about order n log n times if you sort and then there are 2 n event points for corresponding to the left and right vertical boundaries of the n rectangles.

So, at every event points, if you do it naively or mean in the in the straight forward way you just did it. So, that is n log n. At every event point, we may have to update this essentially. So, the straight if you just do it naively and then you will get this just thing about order n log n times 2 n, that is some of n square will pretty close to n square let us say you know n square fine. So, we are unfortunately back to the same complexity if you were to actually compute these intersections and treat them as disjoint rectangles, but you wanted to avoid all that.

However note that all this is done assuming that these every time we have to compute, we encounter an event point correspond to the beginning of the end of the rectangle, we are computing the some of the intervals from scratch. But the reality is that, when we encounter the left or the right boundary of a rectangle exactly and let us again assume that you know no two left or no two right boundaries are the same vertical line. Then actually exactly there is 1 interval that is going to get deleted here.

So, this rectangle ends. So, this gets deleted or corresponding to here, this rectangle this interval corresponding this rectangle gets added. So, at every event point, the change is very incremental and trying to you know sort of run this entire 1 dimensional algorithm, what I mention is something that hopefully can avoided because you have a lot of information you know corresponding to the remaining intervals is just that one interval we have to take care either we have to so, either we subtract that one interval or we have to add that one interval.

Now, that may or may not make difference to the intersect, but all that is essentially again with you know just 1 interval that we need to somehow handle. So, it is some kind of a dynamic situation. It is not fully dynamic, I am saying it is not fully dynamic because these intervals are known to us in advance; the rectangles are given to us in advance. So, it is not that this 1 dimensional problem, we have just arbitrarily introducing some interval like this. But this interval is known to us because it has to be either the right boundary of a rectangle or left boundary of the rectangle and it is available as the initial input. So, it is not a completely arbitrary interval.

So, we have fair amount of information available and we should be somehow able to exploit that.

So, (( ))

yeah. So, I mean the idea is that. So, what area. So, now, the question is that if we keep track of some of that you know this part of the x axis is covered by this interval and if this interval disappears, then we should update something, but it is not such a easy thing to do. I will give you a very bad example it is not easy thing to do because we to now keep track of the first of all it is a continues information. I cannot keep this information for every x coordinate ok.

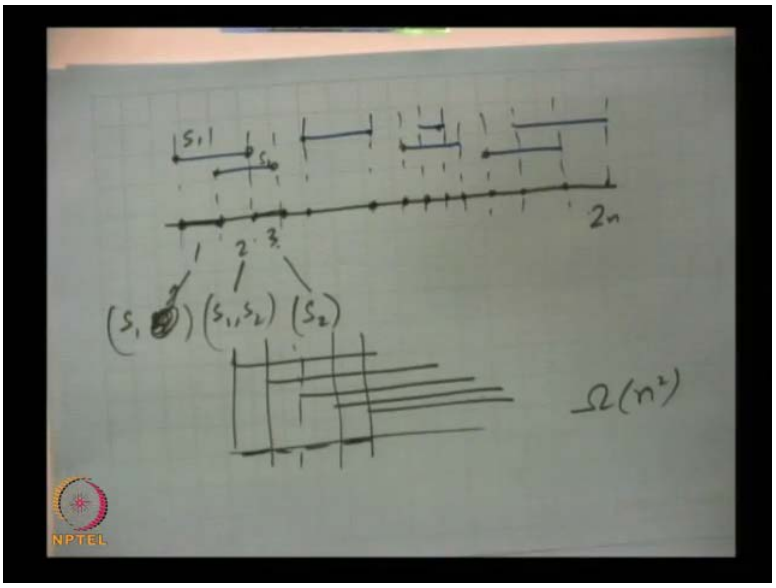
But these (( )) we only access those (( )) intersection right.

yeah. So, here this x axis this x coordinate is covered by these two intervals. The thing that I am trying to point out is that which of these x coordinate should I keep track of because there are infinite number of x coordinates.

(( ))

Right. So, the information remains unchanged let us say in this whole thing and what is observation here, that it must remain unchanged between two event points two consecutive event points. So, we have to deal with. So, now, let me expand this figure.

(Refer Slide Time: 27:57)



So, if we have lots of intervals, we have to deal with relevant ones are corresponding to this end points and so, if I take this to (( )) x axis. So, these event points corresponding to end points. So, between any two consecutive event points, the information is fixed.

You know that these are exactly the two or three or four; whatever segments intervals covering that x coordinate or that this intervals, but if you keep track of this information also, suppose I am in a situation like this. So, then what means it means I have some intervals I have too many intervals, I have 1, 2 all the way up to 2 n intervals. So, for each interval I keep track of which segments covered it.

So, interval 1 is covered by you know segment 1 and segment 2. So, 1 is covered by segment 1 and segment 2. 2 is covered by sorry 1 is only covered by segment 1 and sorry 2 is covered by segment 1 and segment 2. 3 is covered by segment interval 2 or segment 2. So, this is the kind of information that we had in mind. The bad thing about this, so, therefore, you know if there is something that is need to be updated, if I delete or add a segment, we know exactly you know which of these intervals are going to get effected, but then large number of intervals could be effected and not only that, this information its self has a bad you know you have a situation like this, suppose what it means is that...

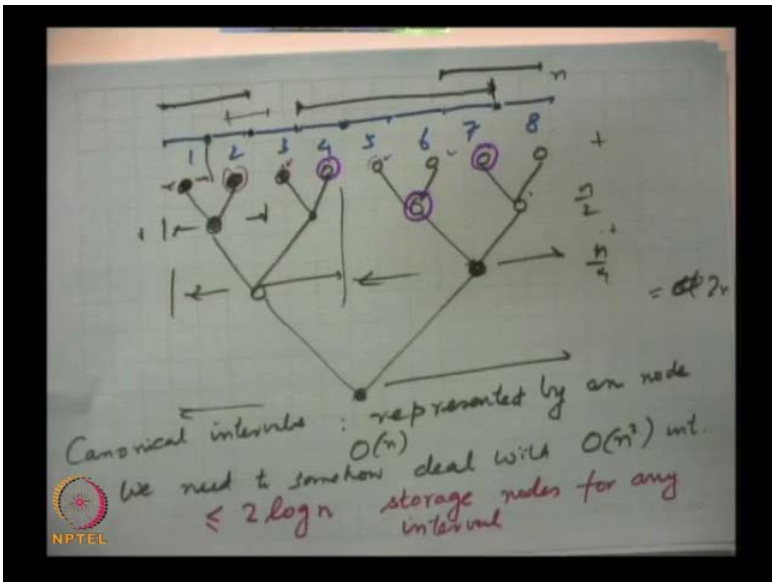
So, for each interval, there could be a large number of segments covering it which is in fact, if you do this will be at least about n square kind of information that you need to store. So, if I need to square the square information again I do not have much hope about

an efficient algorithm. So, although this kind of information can be maintained in this way that you know which and these intervals are known to us, we know that which we know the rectangles we know the boundaries and therefore, we know all these intervals in advance, these 2 n intervals in advance, but still updating that could be quite expensive because a single segment or single interval can cover a lot of them.

So, if I have to keep it updated whenever encounter the right boundary or the left boundary, it could still mean a lot of work and it could be let us say about order n work per update where I have given you this kind of an example and therefore, again you are back to this n even points each update takes order n time you are back to the order n square situation. So, we have to be much more clever about this data structuring part and how do we actually update do these updates. So, how do we do these updates? So, for that, we will actually make a few observations. So, there these intervals. So, there are these intervals; 1 to 2 n. I will try to define a kind of... So, 1 is that, let me number those intervals. So, well I have done it here 1, 2, 3 up to 2 n.

Yeah keep a fresh page.

(Refer Slide Time: 32:24)



So, I have some kind of intervals 1, 2, 3, 4, 5, 6, 7, 8; suppose I have only eight intervals just for an example. I will define a kind of it you know its tree kind of its structure associate with these intervals. So, I will treat these eight intervals as leaves of the trees. Let us say, I have this leaf is associated with this interval. This leaf represents this

interval and this leaf represents this interval and so and so forth. Now, what the actual intervals are the lengths etcetera, that information I can store, but now you know you still be more of a combinatorial representation. So, what the lengths are weather they are equal or non-equal, it will not make any difference to the structure of this tree. So, these intervals sorry this interval corresponds to this node, and then I will define the parent of these two leaf nodes which will be the sum of the intervals that the two children represents.

So, this one will represent essentially you know the entire interval from here to here. This represents only this much; similarly for this things. Now this 1, the parent of these two nodes will define and represents this entire interval. So, this one represents the entire intervals and of course, there is no the root of the tree you know represents this entire tree.

So, I have a certain number of intervals, just for again for simplicity, we are assuming it is a power of 2 that I can define this tree very easily, even if it is not a power of 2, you could just either have some empty things you know are you can make it in a almost balance that everything all the leafs nodes will be within 1 1 level with each other. Now. So, this yeah

(( ))

So, that we will come to that. So, right now, these are water called canonical intervals. What is a canonical interval? The canonical intervals are exactly those represented by the leaf nodes. (( )) not leaf nodes, by any node. So, this is a tree which has how many about fifteen nodes, eight plus seven. So, these fifteen nodes represent you know fifteen intervals which I am calling canonical intervals namely so, the leaf nodes represents these if you call it n the n nodes, the once about the leaf leafs the parents of them represent let us say n over 2 and there are again n over 4 and you know etcetera. So, essentially this tree structure is able to represent only a linear number of intervals.

Alright. Any node of the tree corresponds to some interval and the total number of representable intervals is in these nodes is only n plus n over 2 plus n over 4. So, some order n, let us say 2 n or something. We have still you know fairly restricted and the question that came was what about the arbitrary intervals. Suppose I want to represent this interval, there is no node in the tree that represents that interval.

How many intervals can be there? Possible intervals according to this structure? About n choosed right. I mean I can choose any two end points among these end points. So, there are n choose to possible intervals. These are not arbitrary intervals you know these have to whatever intervals we considered must be flush with one of the end points. You know I cannot I am not thinking about representing something like this from middle of this and middle of that, I am only thinking about representing intervals that span you know two such points of this because these points are known to us. If you think about original problem, these points are known to us.

Any interval will span they are not arbitrary intervals, it has to span you know two such event points or end points whatever you called them right. So, if you restrict your intervals to be those kinds of intervals, then I can choose any two points and that represents an intervals and that is basically n choose to. So, we have only order n canonical intervals and we need to represent, need to somehow deal with order n square intervals. How do you do that? If you could do that, then I clean that our problem, Well I mean that that is basically what we are trying to do and have a nice way of representing these intervals so that we can do this deletions and insertions of the intervals easily. So, let us look at this one interval that I just I pointed out you know this interval alright.

So, clearly it cannot be represented by one node, but we should be able to some are represented by some union of some nodes perhaps. So, what do you think which one should be used to store this interval? (( )) Yes that is what I am saying precisely. So, let me use another color. So, if I want to represent this interval as you saying that I should store it here of this node and which are the nodes. (( ))

I stood well I have a choice of this and this also. I could I said I could do this, this and this. I am done right. If these are, but these are all elementary intervals essentially those are the intervals of the leaf nodes, but then I if use leaf nodes there are too many leaf nodes. So, especially if I both leaf nodes or the both two nodes that represent, they are used for the storage I might as well substitute them by the parent, then I save something. See if both these nodes are required, then I am not going to use that, but would rather use other parent. So, I should use this. So, here also I could I have I cannot use this and this. So, I can use only this. So, I will store this and then of course, I do not have anything else. So, I will store it like this.

Suppose it well the case this was all to used, then I wouldn't have used this and I would have store the whole thing here. So, if both the children are storing, then I will not use them, but I will use the parent. So, therefore, you can see, the storage structure of an interval is such that you are not going to store them in two nodes you know which are ancestors to each other.

(( ))

Yeah. So, you know. So, the answer comes very quickly, then it is a total number of nodes where that is required to store any arbitrary intervals is about order log n and so, we will have to justify more carefully, but if this is true then it is great because now you see that an interval if you can use if it can be stored using only order log n intervals as a nodes of this tree, then I have to when I am adding or when I am deleting, I have to at most update order log n nodes of this tree and what do I mean by update. So, eventually I would like to I have some intervals. I have some intervals not just one interval. I have some intervals; may be this interval this interval, this interval etcetera and I want to find the total measure of these intervals. I am going to find I am going to store them in some nodes.

I want to compute the result in the root node somehow and that is how I am going basically keep track of the total measure of the union of the rectangles and how is this done? If there is some interval that is stored here in the root node, this is by the way an actually tree, I mean our computer science trees are usually like this, I could have written like this is actually growing out of the ground right. So, anyway. So, this is a real root. So, if there is any interval that is stored here that is one of is this is also colored which means that this interval, there is some interval corresponding which is stored in this node. It means that the measure of the sum must this entire team.

You know all the way of this eight whatever it is from left end point to the right end point of course, each of this end points has an coordinate actually. So, whatever it is, that entire length would be my measure because there is some something is stored here, if there is something stored here right, but it may not be the case. If it is not the case is then there nothing stored here then what we do? Then it must be the sum of what is stored here and what is stored here because that is disjoint. So, whatever is the total measure stored here and the total measure stored here, it will be the sum of that.

So, if we basically just you know go through this tree and we have to visit every node, you can easily find out you know what the total measure is of this union of the union of these intervals. So, that will be order n, if you have to compute this, but then you do not have to compute the entire thing. We only has to make updates. In our in our context, what are we doing we are only either adding an interval or deleting an interval. So, we do not have to visit all the nodes to for doing that. We only have to visit those nodes which are either added because of this thing or deleted because this when this gets deleted you know this things this storage basically is thrown again. So, then and this is the we are dealing it only order log n nodes, we can do this updates in time proportional to order log n order log n and the height of the tree.

(( ))

Yeah So, I will come to that. I am just giving you a I am just giving you a summary of you know I am giving you a feel of why you know all this is good and then we will do the precise reasoning. So, if what I am saying is correct, essentially you have an algorithm where every update will cause to order log n corresponding to each event point and there 2 n event points corresponding to the left and right boundaries of the rectangles. So, therefore, the whole thing will work out in log n time, that will be absolutely great. So, now, let us take these things one after other.

Why is it true, why do we claim that there are only order log n order log n nodes that will be where n interval can be stored because there can be at least two nodes at each. Because if they are (( ))

Correct. So, that that is a fairly you know accurate reasoning. So, what is being said is that, it look at any level of this tree corresponding to any interval because it is a continuous interval, you can have at most two nodes that stores some part of the intervals some sub interval of that.

I cannot have the situation where I am storing something here. So, one is that if you if you have two consequent things it must move up to this thing. So, and if we cannot have the situation where we have something and gaps and something like this because it is a continuous interval.

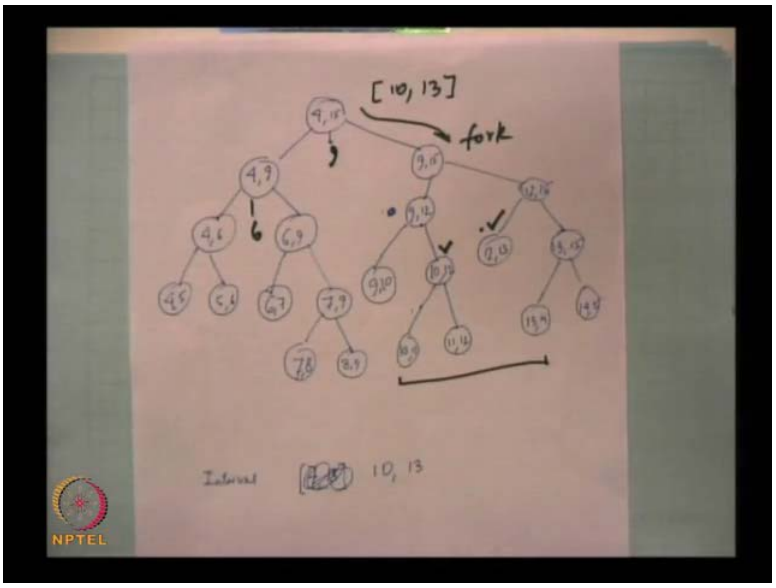
It cannot have there cannot be a gap because there must be some storage here also, if there is a storage here then they should two should combined and move up. So, there can be at most two storage nodes corresponded to interval at each level and there are only log n levels. So, the at most 2 log n. What you argued is that, there is less than or equal to 2 log n storage nodes for any interval. So, that is the situation, then y is the update order log n. does not immediately imply the update order log n? Not really. See if you think about it here is a node when we want to update we have to basically walk along this node to the root and update the counts.

Each node could have one or more intervals as long as it has an interval. It means it covers the entire interval corresponding to this canonical interval. If the count is 0, then it must be the sum of these two.

So, whenever we have a new storage node or whenever we are deleting something from a node we must actually start walking towards the root updating all the counts. So, if there are 2 log n storage nodes, what does it implies. It only implies about log square n because for each node, for each node you have log n updates. So, for log n nodes you have order log square n updates.

This would be the nice reason, but actually there is a more structure to it which makes it actually an order log n update rather than as a log square n update. It cannot be clear of exactly from this example. So, I will give you another example may be from there it will become clear. So, this is something that I drew before I came to the class. So, let us look at this.

(Refer Slide Time: 49:00)



Is it visible or not? So, shall I write or let me draw it in the case because it may be not visible in the camera.

(( ))

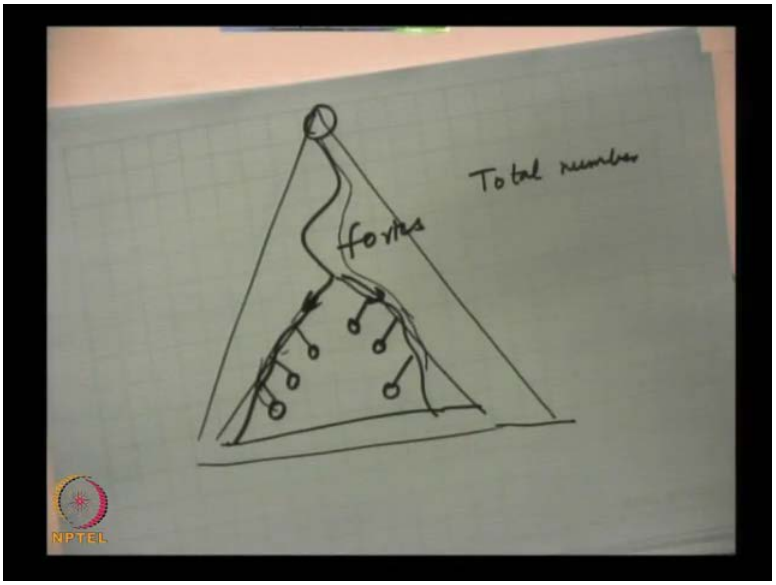
So, I have drawn something corresponding to these intervals 4, 5, 5, 6; this are my elementary intervals, 6, 7, 7, 8, 8, 9; all the way to the 14 ,15 and I have a tree like this and you look at this interval 10 comma 13. So, this is I have drawn it in the proper way that tree root is at the top. So, if you actually now. So, we argued that there are about 2 log n's 2 edge nodes for each intervals. Now we will actually able to argued to something about the structure of this, they are not arbitrary log n nodes. So, if you look at this interval 10, 13; it kind of spans this. It kind of spans this and how do we actually you know if you have given this tree and I have to find out where we should allocate where are the allocation nodes, I take this 10, 3 10, 13 interval and look at the... So, this is the root node which corresponds to the entire interval from 4 to 15.

So, 10, 13 you know and we see that actually this 10, 13 correspond to… So, this entire interval will lie to the right on the right sub tree because the left sub tree only goes up to the 9. So, for every node, I will actually store a kind of a divider dividing point. Here it is 9, here it is 6 and so on so forth. Now this entire interval 10, 13 lies to the on the right sub tree. So, we actually basically come here.

So, when we come here and if this for bigger example, it could be that this entire interval lies to one side, that could the case. So, for a while may be this entire interval basically travels down for a few levels of the tree before it hit some node where some part of it is at left some part of it is at right.

In this case it happens that at this node itself you know which we call this a forking node it fork basically. It forks and after it forks you know things are very regular. So, this is… So, we have 10, 13 and if you look at it, this is an allocation node and this is an allocation node So because it covers 14 12 and 11 12, of course, they have to be get the same level. So, the structure actually again if I had a bigger example that would become clear, but now I am I just abandon that and just show you what is basically happening.

(Refer Slide Time: 52:20)



So, if you have any tree, this interval that we are looking for, that entire interval travels for a while together and then it forks and after it forks, you can only have one kind of pattern, the worst case pattern or whatever; one kind of pattern is that you will have something like… So, there will be some path.

So, you look at the left path and the right path. So, each interval let me come back here. So, you look at the left end point of the interval and the right end point of the interval. So, 10 you just locate, you just try to find where 10 lies and you know 10 will take you basically like this and 13 will take you something like this well.

Well. So, these are the actually the two end points will trace this kind of path in this tree. So, which side the point falls on. So, after all, we are talking about you know whether the points for every node we have a dividing point. So, the point goes left the path goes left.

Going adjacent to (( )) right in to the.

13 was.

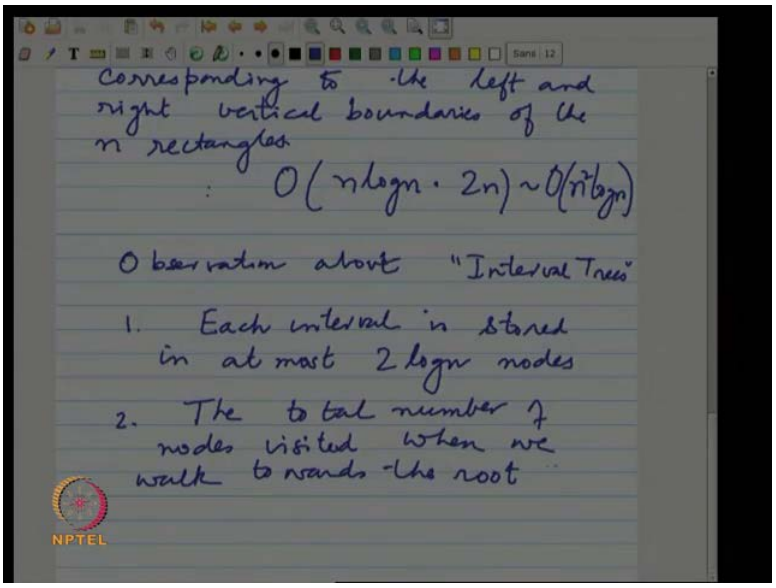
(( )) This one.

(( )). This one

Yeah you could have (( )) also right

yeah. So, actually strictly speaking you know this 13 you know, this is actually an open intervals. So, you know the intervals are actually something like I should have be pointed out. It is actually something like a to b alright. This is strictly speaking it is like this. So, we actually trace we can trace out two paths for the left end point and the right end point of the intervals and this will path like this and so, this some path and there is some path and you can actually you know this is something I will leave for you to you know convince yourself your allocation nodes will be exactly this; the right child of the left path and the left child of the right path and because this is the case so, when you actually walk up from there, you know this paths actually merge. So, you are not dealing with you know order log n distinct paths, but they are basically you know I have a very nice patterns. So, I am going to only deal I am only going to visit order log n nodes on my way up from each of these nodes. The total number nodes that I visit will be order log n. So, total number; maybe I should write it here.

(Refer Slide Time: 55:32)



So, one observation is that about. So, this structure I should say is about it is called interval trees, observation about interval trees. So, whatever I drew corresponding to the interval is called interval tree. So, one is each interval is stored in at most 2 log n nodes and two: the total number of nodes visited when we walk towards the root from the allocated nodes is again less than 2 log n. So, this is what actually makes the update, not even log square n, but log n.

Now you can put everything together. So, what is. So, let me just summaries before I end. So, what we did today was we took this rectangle problem and did the line sweep and when we did the line sweep, the problem came down to how do we keep the intercepts updated when we sweep between two consecutive event points defined by the left and the right boundaries of the rectangles.

Now, for performing this update of the intercept, we looked at the 1 dimensional problem will becomes essentially we given a set of intervals now if I ask add an interval or delete an interval, how do we update the measure and for that, we defined this interval trees where the elementary intervals are consecutive points. So, leaves correspond to elementary intervals and every interval node corresponds to union of its children and that is how we defined the tree and if there is.

So, these the nodes corresponding to the trees correspond to about order n canonical intervals known to represents an arbitrary interval, we figured out which are the storage

nodes for that particular interval and then we observed that there are at most 2 log n allocation nodes for each interval and moreover not only that, these allocation nodes for each interval you know I have a very nice structure. So, that you know this is basically data structure corresponds to this that when you walk off from these nodes towards the root, we are not going to visit more than again about 2 log n towards the nodes. So, this entire thing therefore, gives us a order log n update time for adding or deleting in a interval, but beware that this is not a completely dynamic process because this intervals are known to us in advance.

I cannot deal with an interval whose end points do not coincide with this end points, but in the context of the rectangle problem, we are know we know all these things and therefore, the entire thing about the line sweep on the rectangles, then computing the area will work in n times each update will take you order log n for each event point and the whole thing will be order n log n. So, we can compute the area of the rectangle in order n log n. So, we should end here today.