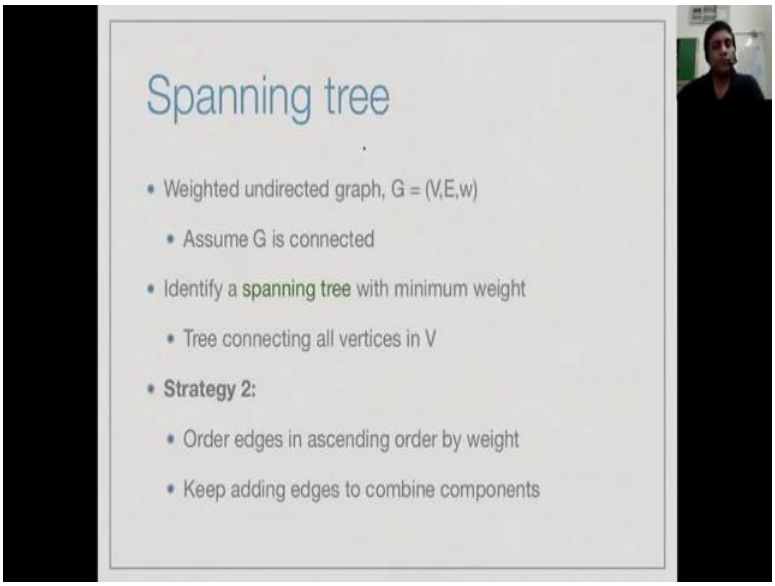
Design and Analysis of Algorithms
Prof. Madhvan Mukund
Chennai Mathematical Institute

Module – 07
Lecture - 31
Spanning Trees: Kruskal's Algorithm

We have seen one algorithm for the minimum cost spanning tree namely prim's algorithm. Now, let us look at the other strategy we talked about namely Kruskal's algorithm.

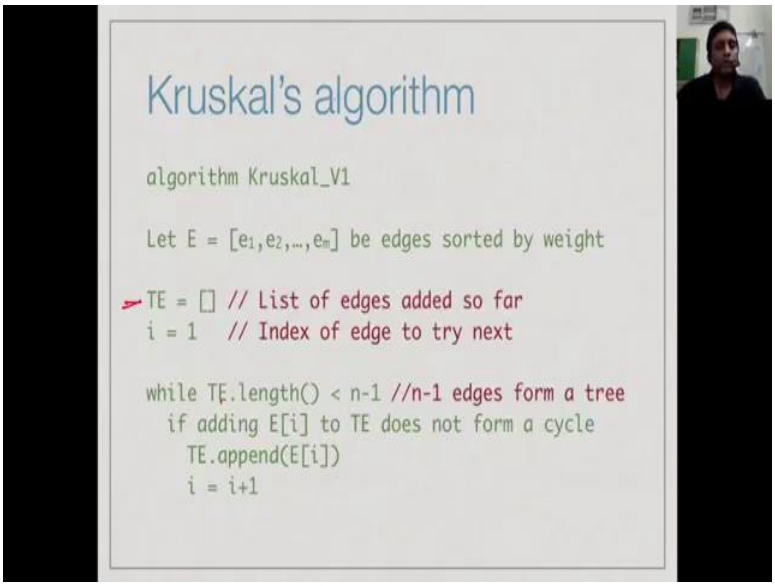
(Refer Slide Time: 00:10)



So, we are looking for a minimum cost spanning tree, in a weighted undirected graph. So, prim's algorithm starts with some edge and gradually expands the edge into a tree, whereas Kruskal's algorithm follows the other strategy, which is to order all the edges in ascending order of weight. So, it keeps trying edges from smallest to largest and for every edge if it can add the edge without violating a tree property, it adds it.

Now, in the process of adding the edge, it may not actually construct a tree, all it make sure, it does not violate the tree property, then the it does not produce a cycle. So, if we keep adding edges, so long as we do not produce a cycles and at the end the claim is we get a minimum cost spanning tree.

(Refer Slide Time: 00:52)



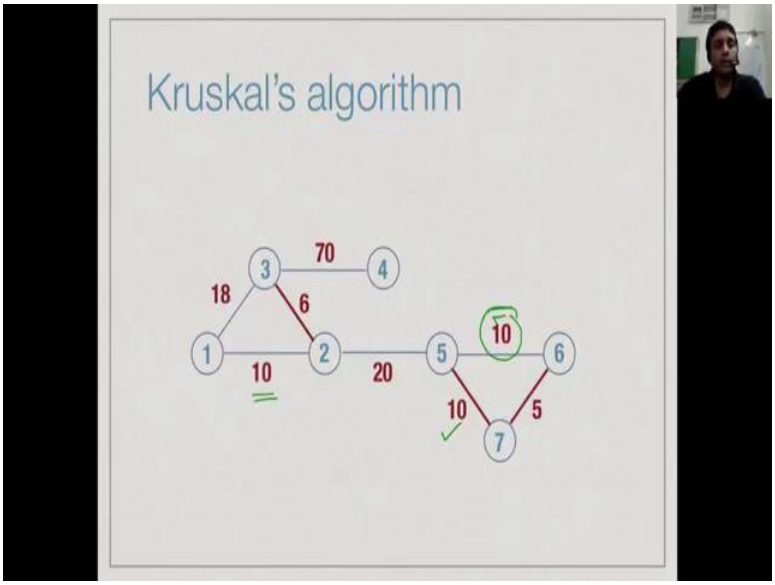
So, here is a kind of high level view of the algorithm, so let the edges be sorted in order e 1 to e m. So, we start with an empty tree, so again we will keep the tree as the list of edges and now we are going to scan this in order, 1 to m. So, let i be the index of the edge to be try next. So, long as we are not yet added n minus 1 edges. Remember that if you add n minus 1 edges and we have a connected graph, it must be a tree. This is one of the various characterization that we said about trees.

Trees are n minus 1 edges, but any connected graph with n minus 1 must be it. So, once we have n minus 1 edges we can stop. So, so long as length of the tree in terms of number of edges is not n minus 1, we have to add some more edges. So, we look at the next E i, if E i does not form a cycle when added to TE we append, then we look at i plus 1. If it does form a cycle, we just discarded and go ahead, this is all Kruskal's algorithms ((Refer Time: 01:49)). Scanning through all the edges and until we have found a tree, it is just keep looking at each edge in turn in ascending order the weight and every time if it can added to without forming a cycle it will added, it cannot added without forming a cycle it will dropped.

(Refer Slide Time: 02:03)

So, let us look at the same example that we say of a prim's algorithms. So, here this smallest edge weights are remember that we have sorted it neither. So, 5 is the smallest then 6, then we have 3 of weight 10, then we have 18, then we have 20 and then we have 70. So, now if we pick among 3 smallest one, we start with edge 5 we first add the edge 5 to our tree.

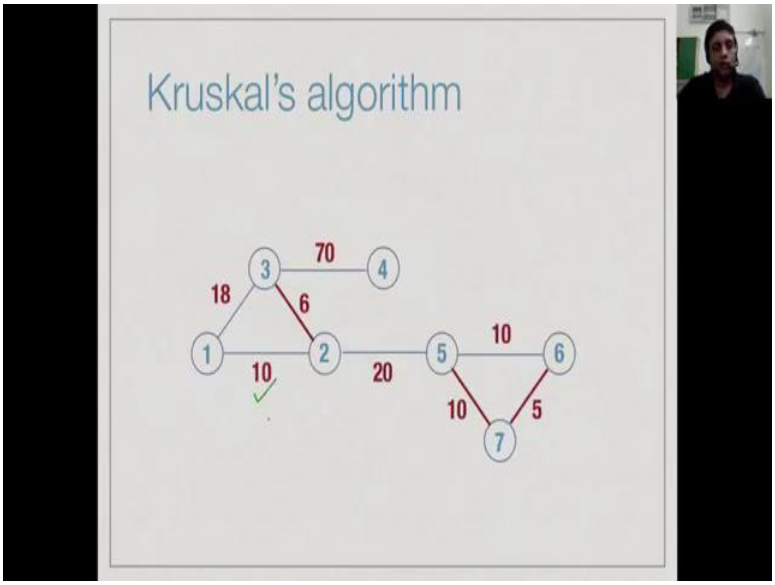
(Refer Slide Time: 02:28)



Now, the next one is 6, so we add 6 to our tree. Now, we have to pick one of the edges labelled 10, we can pick any one, so let us assume that we pick this one. Now, we want

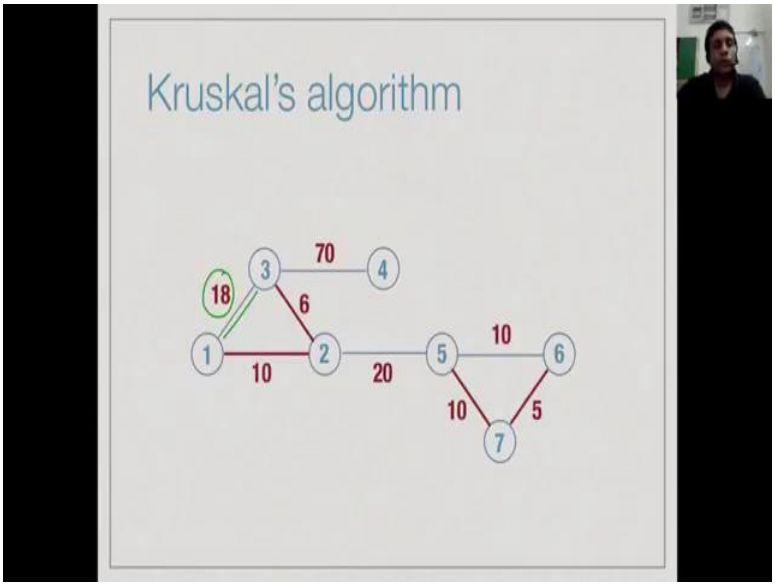
to pick another edge label 10, because the next two edges are again label 10. So, supposing we pick this edge label 10, then this edge label 10, form this cycle.

(Refer Slide Time: 02:47)



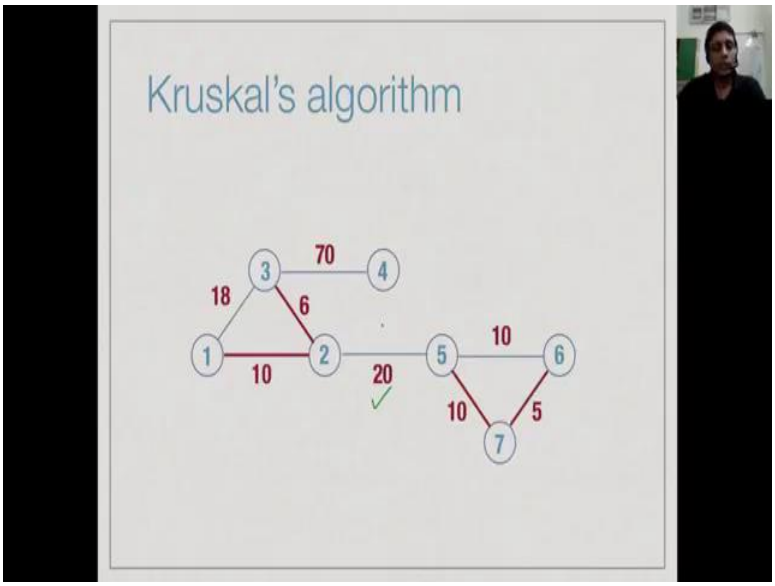
So, we cannot do it, so we discarded it, we discord it and go to the next one. So, now we pick this one, so we through this away and now we will go to this one.

(Refer Slide Time: 03:00)



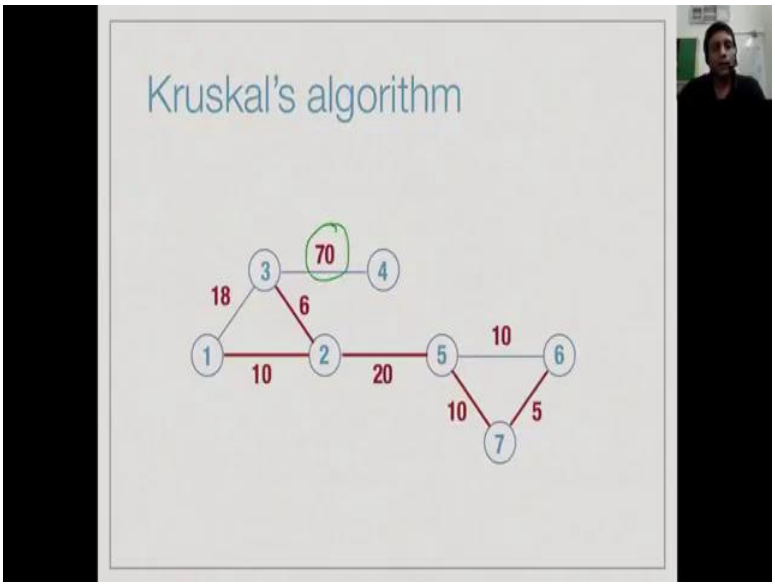
And now this one make an order, so now we have finished the tens, so the next one is 18, but again added 18, we going to form a cycle.

(Refer Slide Time: 03:08)



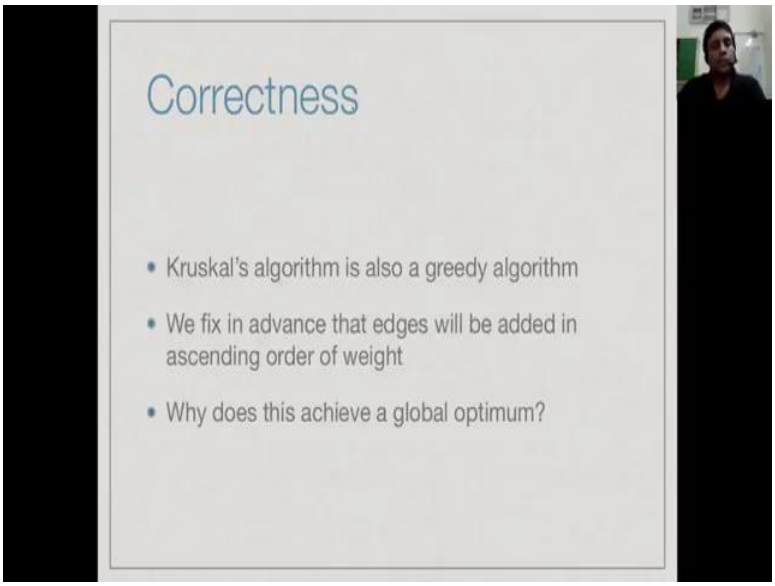
So, we discard it and now we go to the next one which is this way.

(Refer Slide Time: 03:16)



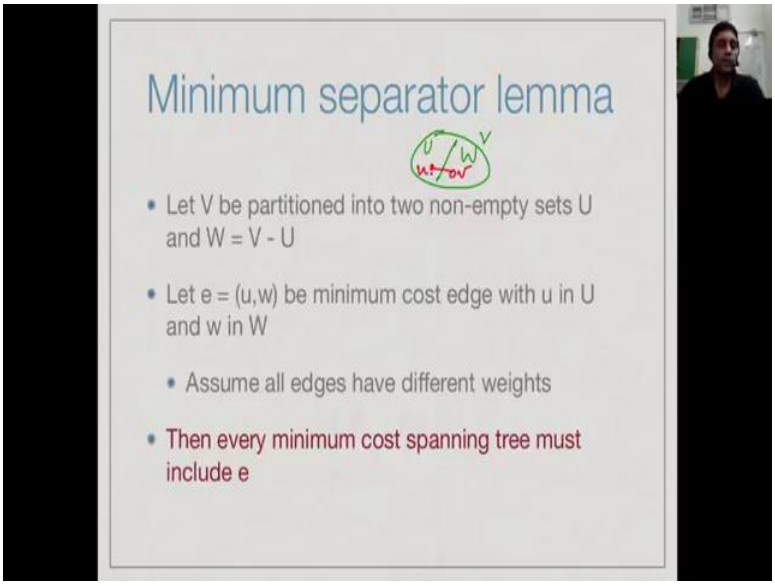
Now, having added 20, then we still not I have done, because we have added we have skill to add enough edges we have 1, 2, 3, 4, 5 edges and we need 6, because we have 7 vertices quickly, we get n minus 1 edges. So, the only edge left is this 70, so we have that in a way. So, this is the spanning tree form of Kruskal's algorithm, we actually scan through all the edges in this particular case and on the way we dropped the coupler.

(Refer Slide Time: 03:43)



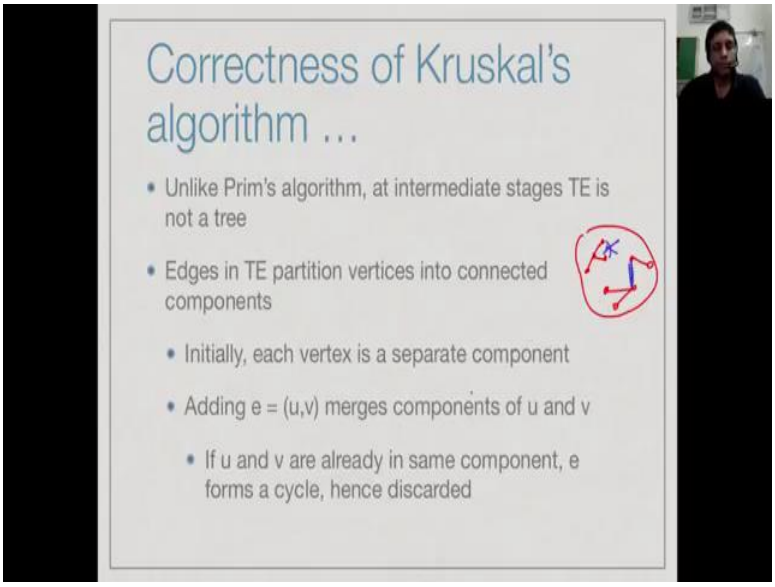
So, Kruskal's algorithms is, also a greedy algorithm, in this case we do not make a... So, in prim's algorithms more as Dijkstra's algorithms, we make incremental choices, local choices at every point, we will look at what we do next, based on what we know now. Here, we make a choice well in advance, we say right at the beginning let us sort all the edges and do it in that order. And again it is not clear by doing it in this particular order which decided right at the beginning gives us overall an optimum solution.

(Refer Slide Time: 04:13)

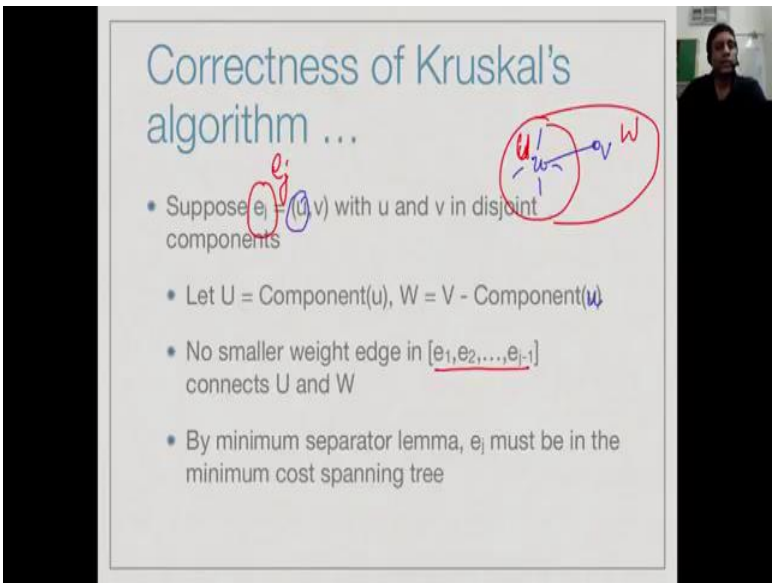


So, once again we will use the same result that we used for prim's algorithm, this minimum separator lemma. So, recall what the lemma said, it said if you take a set of vertices and separate out into two non empty groups U and W and if we take the smallest edge, now the graph is connected. So, there must be an edge connecting these tool, we take the smallest edges which connects the two parts together, then this edge must be there at every spanning tree, every minimum cost spanning tree, this is what the lemma said.

(Refer Slide Time: 04:44)



(Refer Slide Time: 05:29)



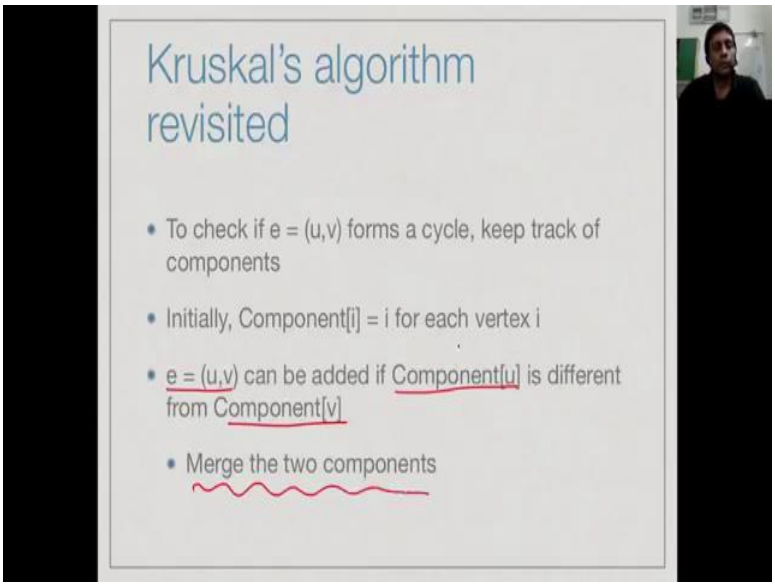
Now, what we...Therefore, no edge whenever we add an edge, it is 2 n points are in disjoint components. So, in terms we are using the lemma, let us call capital U to be the component containing one n point of the edge and capital W to be the rest, whatever is outside this, so this is whatever is outside component. So, we have U and whatever it is connected to. So, this is my set U and then I have the rest of the vertices, so U is connected at this point to something in a different component, but that an all the other components is what I called that W.

So, now what we know this is the first time that we are trying to connect U to V, U to V have not been connected, so far. But, we have been looking at edges an ascending order of weight, so if you look at all the edges. So, this current edge we are called e j, so if you look at all the edges you not to e j minus 1 none of them connected U to V are the component connecting U.

Because, had it component containing U to the component containing V then when could adding this edge, this edge form a cycle. So, this is the first stage that we can use connects in two moves that no edge we have seen, so for a connects these too. So, putting all this to together, this is the smallest edge which connects capital U to something outside capital V.

And since is this smallest stages, the lemma tells us this must be in the tree and therefore, Kruskal's idea of including in this tree is sound, this is an edge it must be input tree. So, every edge the Kruskal's algorithm adds is actually a validate justify at the minimum separative.

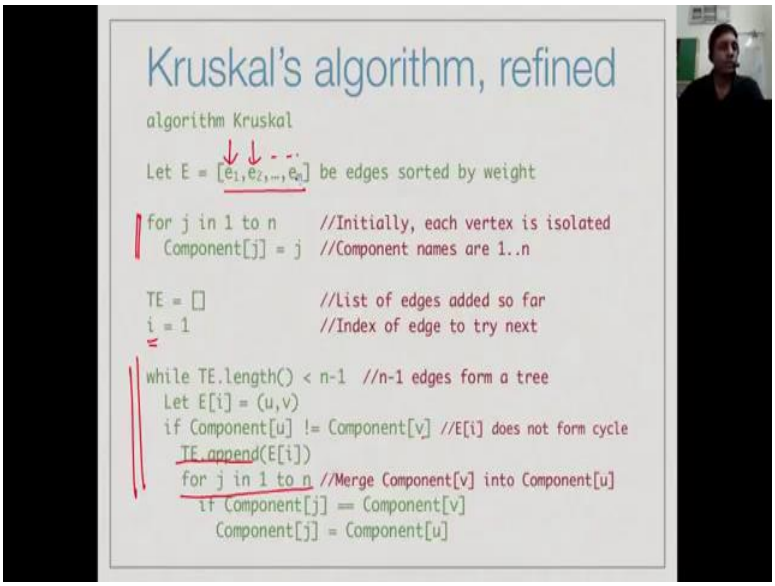
(Refer Slide Time: 07:11)



So, now we have to just decide how to keep track of this property of forming a cycle. Remember that whenever we add an edge, you must first check it forms cycle and if it does not form a cycle you must include. So, the easiest way to check, but the edge does not form a cycle is to keep track of the components, if the edge is inserted on two different components that is one n point of the edges and one components and one n point of the edges in other component. Then, inductively each components is a tree and therefore, these two trees are disjoint and therefore, the new edge does not form a cycle. If the two n points are in the same component then it will form a cycle. So, keeping track whether are not a new edge forms a cycle is equivalent to keeping track of components. So, how do we keep track of components, well, we can basically label the components by numbers and it since we have n vertices and initially there all n independent vertices with no edges in the initial starting point of Kruskal's algorithms, everything is disconnected.

So, initially we are n vertices n components, the most added a thing it to do say, but each vertex i belongs to a components part i. Now, I can add an edge u, v if the component of u is different from the component of v. And at the component u different from components of v what happens after this is the two components become the same. So, add some on merge the two components, in other the at depend the same lay.

(Refer Slide Time: 08:37)



So, this is now detailed explanation of the algorithm, so as before we let e 1 to e m being the edges sorted by weight, initially we said let every vertex 1 t on is in its own components for every j among to n components of j is j. So, 1 is the components 1 and 2 is components 2 and so on, now we start off with an empty set of edges and now we are going to run through as we set before these edges in ascending order. So, we have the index pointing to the first edge in may sorted array, so this is my index.

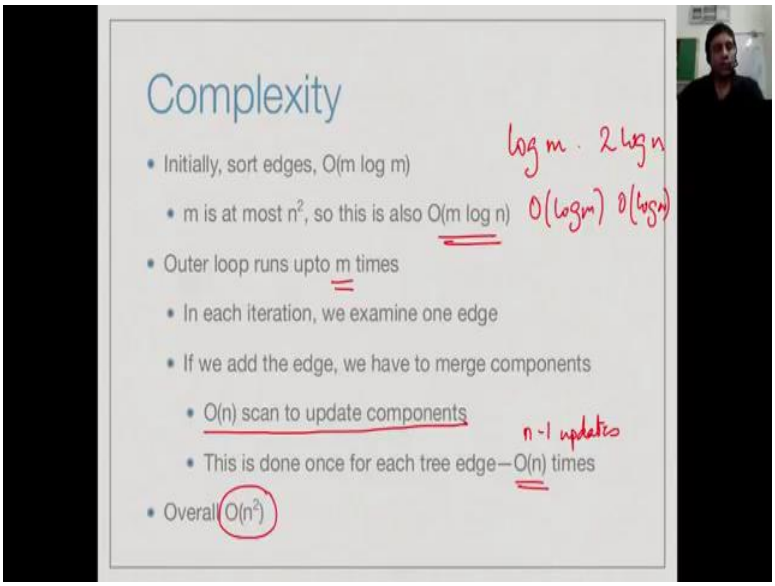
So, so long as we are not add it n minus edges, we look at the i th edge in this sorted order, it is of sum u to v. If the current component of u is different in the current component of v, then we are sure that the edge add does not form a cycle. So, we added to a tree and now we have to do this merging of components, now what is the component of the vertex is just a number.

So, for every j in our set now the problem is that these components could have been merged earlier. So, components of u could contain other vertices component to v could other vertices and all of them must now we given the same component number. So, that they are all new same number, so I have no choice, but to scan every j in my set of vertices and whenever such a j has the same components as v I reset which component to u.

So, after this everything which has the same components number as v as got the same components numbers u. So, therefore, effectively the two components have been merged

into one components all label by the value components of u. So, this is Kruskal's algorithms, just says keep considering edges and if they connect different components add them and merge the components, if they do not connect different components it will form a look, so just through it away.

(Refer Slide Time: 10:26)



So, what is the complexity well a first step in Kruskal's algorithms requires it to sort edges. So, we know that we can sort n merges and m log m time, but n is at most n square, so log in will be 2 times log n. So, order of log n is the same as order of log m. So, I can also write this is m log n, so this will be just another way of writing. So, this sorting text m log m time.

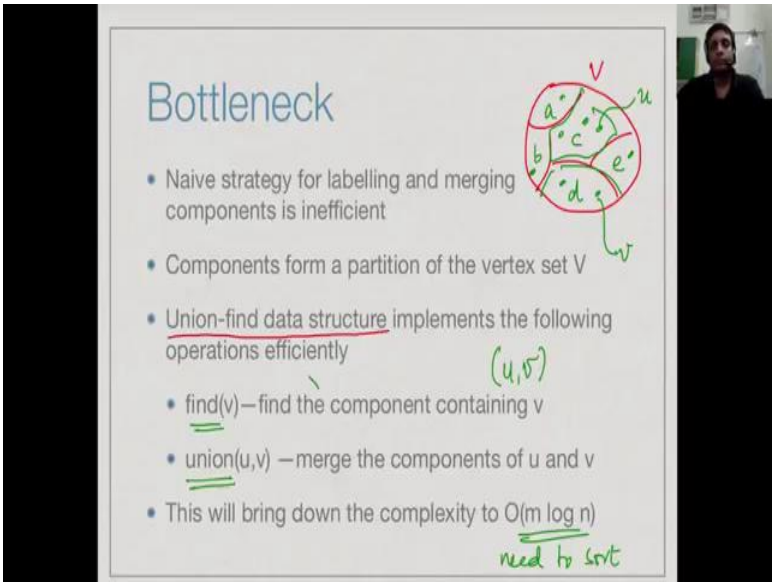
Now, we have an outer loop ((Refer Time: 11:00)), so this loop which runs here, in general runs through all the edges in this list. Because, it could very well that for some reason after go all the way the last one found the tree. Rather, even in the example we saw of the largest edge has to be include in the tree, because it is the only one that connects us some particular vertices.

So, we have an outer loop it is an m text, now among these m times we will update the components for each edge we add that will happen only n minus 1 times. But, when we update the component, that is why we have a problem, because we have to scan through all the vertices in order to change the components number. So, each update of the

component, each merging components that it is order n time and this itself happens n minus 1 times.

Because, we have to keep doing each time we add an edge to at tree, there are going to be exactly n minus 1. So, we have to n minus 1 updates and each update text order n times. So, we have an overall complexity at order n square. Now, remember that prince algorithm n square was the worst case and then we said with the better data structure we could bring it down to n plus m log n.

(Refer Slide Time: 12:12)



So, what can we do here? So, let us look at the problem, the problem is basically this labeling strategy. Now, when we label the components and do this linear scan we are spending order n time in order to merge components. So, we will find just like we can use the heap for Dijkstra's algorithms and prince algorithms to find them minimum vertex to be added quicker and to update the distances, here we will find that what we need to do is maintain this component structure.

So, assume that we want to deal with a set v which is broken up into different components. So, it has components at any given times they call a, b, c, d, e and then I have individual elements which belong to this components. So, sum of them are many, sum of the one and so on. So, one of the things you want to do is given the name of vertex v, you want to ask which components does not belong. So, this is called find, given the find the component, but n the other thing is given to vertex.

So, suppose I am given a u and v it will say merge these two, so take these two components and get then the same name. So, either call them all c or call them all d, so this is called union, union does not take the components it takes two representatives, one representative one components and one representative from the other components and themselves merge everything which is in the components of the u with everything in component v.

And these are the two operation; obviously, that we do is Kruskal's algorithms, find v is what we if have an edge u, v to check at the same component we find the component of u write the component of v with they are the same. That means, the insert component they are different and different components and we can add this edge. Now, if you add the edge then we have to do union v, so these are precisely a two operations and we will find that the there is this data structure which is called union find because of these two operations, which can actually do this efficiently.

And efficiently means what, it means it can eliminate this n square thing and bring everything down to the cost of m log n or remember this we need this sort. Our first step was a sort the edges for Kruskal's algorithms sorts by sorting in the edges. So, m log n is something which we need to m equal to sort the edges. In addition, these essentially this updates as in a heap, the updates the union find reverse the actually log n. And remember that they are going to be a these updates are happening in this iterative loop. So, we actually get something similar to this now prim's algorithm complexity.