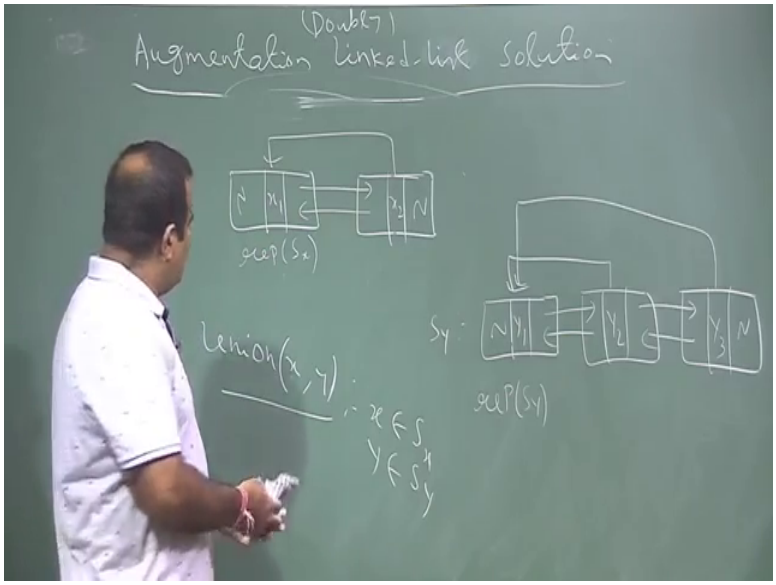
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture - 53
Augmented Disjoint Set Data Structure

So we are talking about union operation, how we can perform the how what how we can do the argumentation in the w connected linguist, we have seen the we are going to have a we are going to add a pointer from each element to the representative element that is the first element, because this w connected linguist and our first element is the representative element.

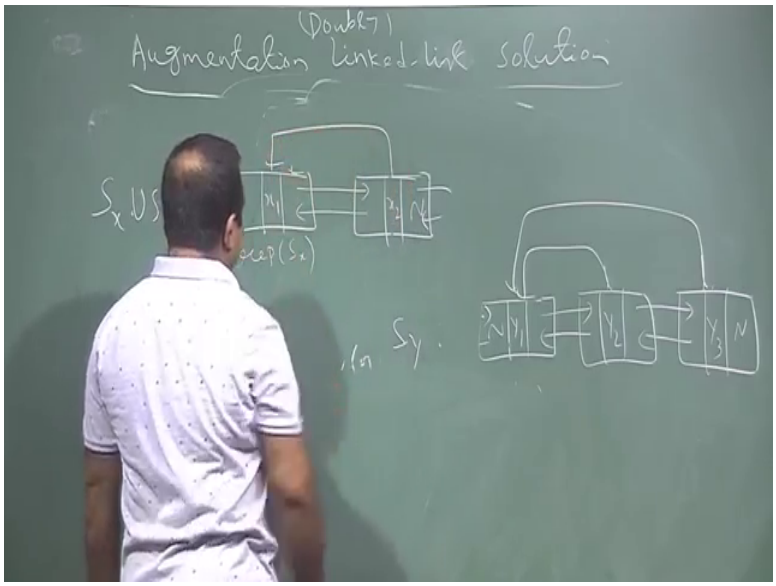
(Refer Slide Time: 00:34)



So, we are going to have. So, now, we are talking about the union operation. So, the union of 2 set union of x y. So, basically a S x. So, basically you have two sets S x and y belongs to S y, now we want to have the union of these two sets. So, that you want to do. So, this is set S x and this is set S y, S is here y is somewhere here in any one of this.

So, for the union what we are doing we are just. So, we are doing the union. So, union is basically S x union S y.

(Refer Slide Time: 01:23)

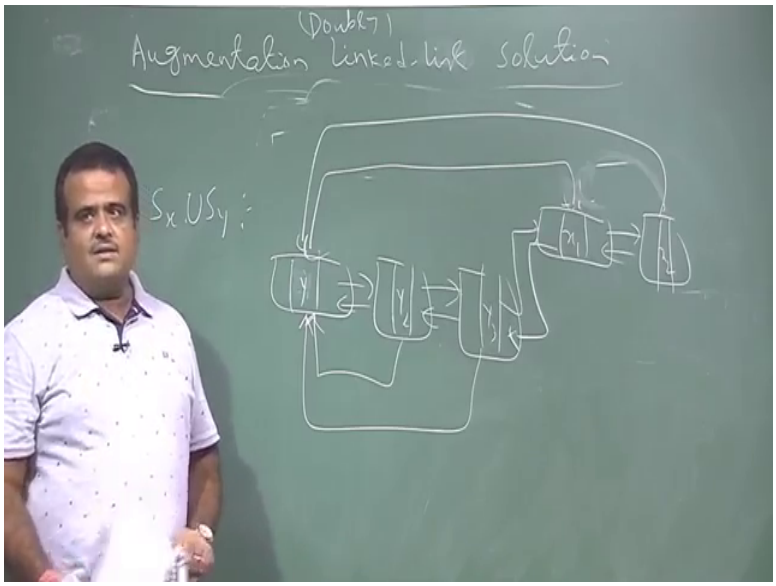


So, for that we are just adding this. So, this last pointer we are just adding this, and then this pointer also we have to change. So, this was the now this first element is the representative for this also, all this nodes. So, this we are going to change now so for this nodes also all this nodes having the representative element S1. So, this is the operation you are performing.

So, this is the union operation. So, how long time it is taking. So, this is the bigger set. So, now, the bigger set if it is order of n, then we are just taking theta of n times, because everybody has to change the pointer because this is the bigger set say now so what is the idea? Idea is to so, instead of merging this with this if we merge that does not matter if we merge this with this. So, that is the trick one. So, we merge the molar set into the bigger set. So, that is the idea of trick one.

So, then I can save it sometime because this is if this size is n and I need this size is some constant then hardly we are changing the pointer of this side. So, that is the idea. So, instead of this what we do the. So, this is our. So, this is our S y. So, everybody was pointing here in S y now this was pointing here now we are merging this set with this set. So, this is the at the end. So, this is let us try it again. So, instead of merging S x with S y we just going to merge.

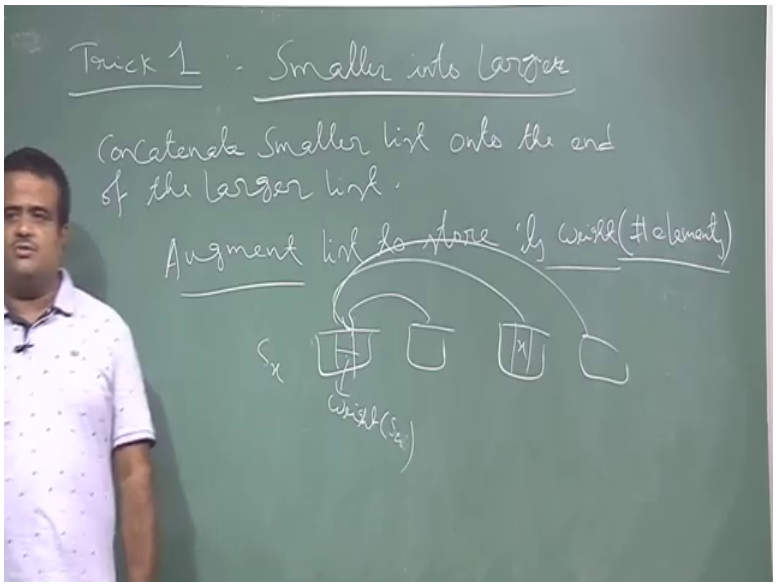
(Refer Slide Time: 03:26)



So, this is our union now. So, this is x 1 sorry this is now y 1. So, y 2 y 3 and now we are merging this, this is the smaller set x 1 x 2 now all this y here this was S y and now this was pointing here and it pointing itself now we have a connection over here and this all these are now pointing to here. So, this is the trick is we want to merge this smaller set with the larger set. So, that way if this size of this constant, but in the worst case it is order of n, but we will do the amortize analysis on this and we will see this operation we take in login time.

So, this is the idea of merging the smaller set with the bigger set. So, now, the question is how to get the smaller set. So, the idea is to. So, this is the trick one, if trick one we apply on.

(Refer Slide Time: 05:13)

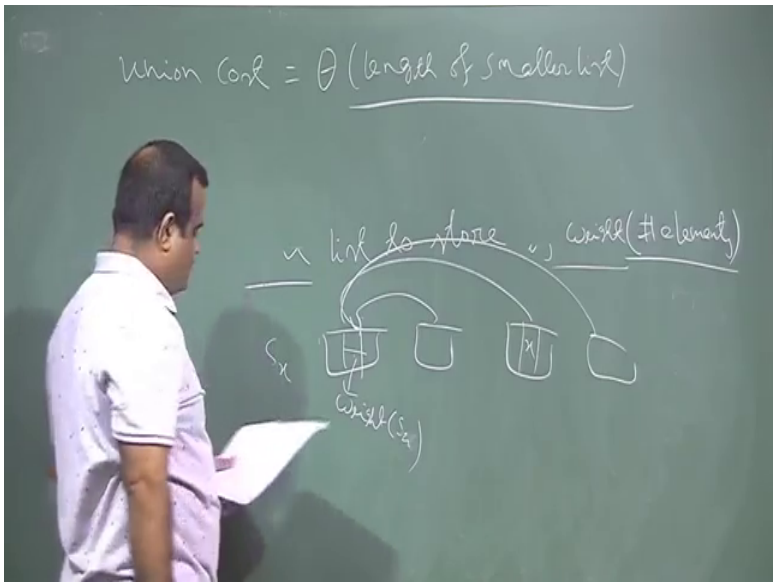


So, trick one. So, that is smaller set smaller into larger. So, this idea this type of trick we will use lot of other places also. So, the idea is to concatenate. So, we are concatenating two list. So, concatenate smaller least on to larger least, on to the end of the larger list. So, that is the idea. So, we are just because the earlier we are doing S is y we are concatenate with this, but is smaller. So, now, we after applying the trick we are going to use the S is smaller. So, we are going to concatenate S in at the end of y list. So, that is the idea.

Now, the question is how we can decide which list is smaller and which list is larger, then we need to try get the length of the list. So, again if you have to calculate the length then you have to travels the list. So, that will take again linear time. So, to save that you have to do again the argumentation. We augment the least to store the size of the list to store that is called weight to store its weights weight means number of elements this is the number of elements. So, this is the new augmentation we have to give. So, we are going to store the e on each element we are going to or in the beginning because we anyway we have access to the representative element. So, there we can store the size of the list. So, we row. So, suppose we have given this is a least this is a least and every pointer pointing here. So, we have given a x, now we can reach here somehow if we can store the weight of weight of the list. So, this is S i or this x i. So, then we came to know which one is larger and which one is smaller then we can apply this trick ok.

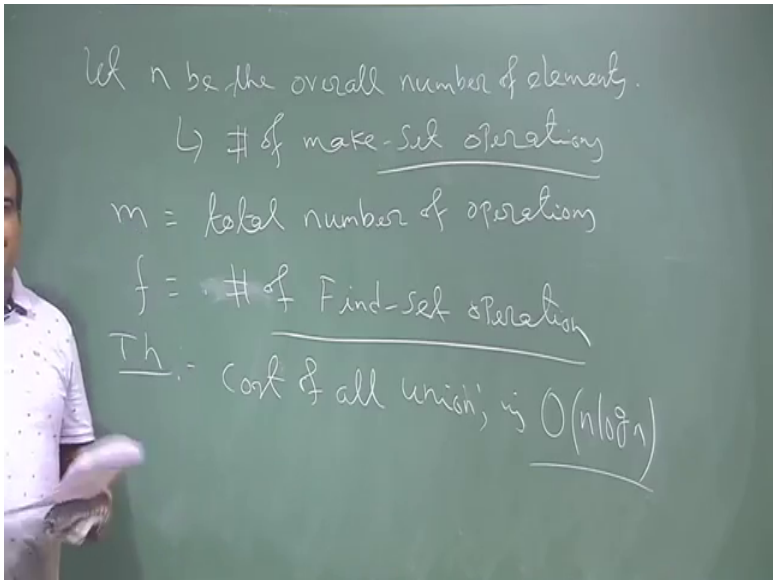
So, if we have apply this trick, then what is the time for doing this union because. So, we are just not bothering about the find operation because find operation augmentation we did and we can solve it constant time. So, we are all the concern here is now the union operation. So, how much time it will take for union.

(Refer Slide Time: 08:39)



So, union cost is basically theta of length of the smaller list or weight of the smallest smaller list length of smaller list. So, now, we have to go for amortize analysis to show that average cost is login. So, for that we have to do the amortize analysis. So, let us. So, this is the without amortize analysis this is that. So, now, we have to do the amortize analysis.

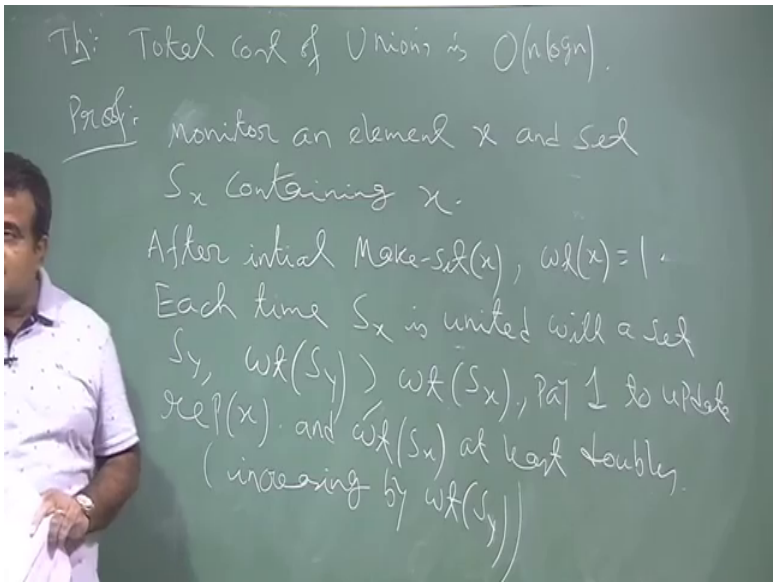
(Refer Slide Time: 09:38)



So, for that let us just denote some. So, let n be the n be the overall number of element. So, total number of elements. So, these are coming by say make sets. So, every time we are making set and doing the union and we are so, overall this is the total number of order of n times you are doing the make set. So, this is this is basically equivalently number of make set operation. So, that is how we get the elements nah. Once we make set we get a single term set then we union. So, this is the way we got the elements. So, this is the number of make set operation, this is the way how we get the sets. So, m is the total number of operation and f is denoted by the number of find operation. So, number of find set operation. So, this we will these are the notations will use in our time complexity in our amortization analysis. So, now, we are going to prove theorem in a amortize sense, that the cost of all unions because worried we are worried about union for this augmentations is order of n login, there are n elements overall. So, if cost is order of n login in a amortize sense, then the average cost is there are n element average cost is login. So, that way we are going to prove this.

So, let us rove this theorem, the cost of total cost of union operation is n login.

(Refer Slide Time: 12:19)



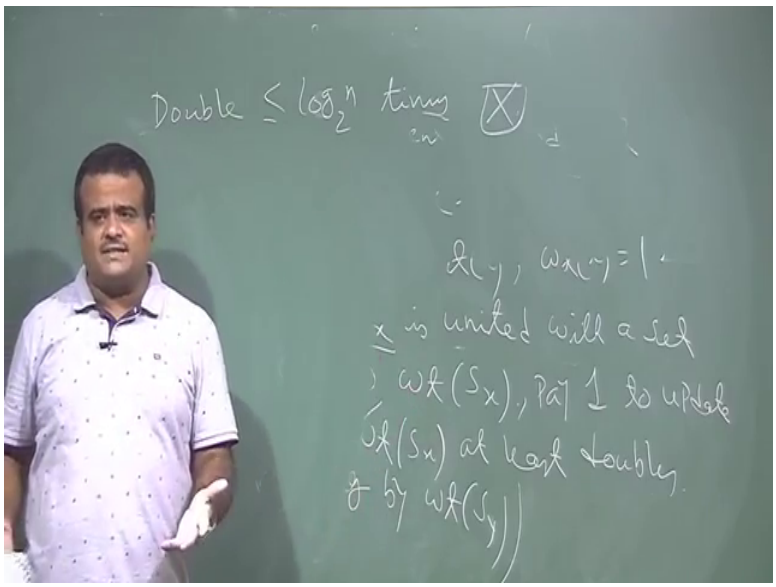
So, let us just state this theorem. So, the total cost of unions operation is order of n login. So, how to prove this? So, this is in the amortize sense. So, we basically we choose x and we monitor x we monitor an element x, we look at an element x we monitor x and it sets and set S x containing x. So, we will see how many times it will getting change and what is the happening with this. So, that is up our interest and there are n element. So, you can just multiply with that ok.

So, we want to see the cost for this union overall cost for this union for this particular element x, and there are n elements you multiply that. So, if we can so that for this x we are just doing the how many times? We are just doing the login times then overall we are doing n login time. So, that is our goal. So, how to prove. So, that is basically so after initial. So, how we are getting x? So, after initial after initial make set, we got x by doing the make set operation. So, that time weight of x is basically 1 because that is the single term element, and we in the union operation we unit x we merge x with some other y. So, each time S x the set which containing x is united. So, well we are doing the union operation with S y united with the set S y. So, we are just doing union of x and y. So, we are doing this. So, now, our trick is we are going to merge the smaller set to larger set now if this weight of. So, if the ac is S x is smaller set then only we are going to change all the element of S x to this. So, that is the idea so; that means, if the weight of S y is greater than weight of S x. Then S x is smaller than S x is going to merge with S y and then we are changing all the pointers of S x to the representative pointer of S x all the

elements for that; and then we are we are paying some we pay one to update. So, there if this if S x comma then we pay one say one dollar to update rep of S x, because then we are going to because this is the smaller set then rep of S x so that means, and then the weight of S x, then S x and S y will be replace by the union then the weight of the S x. So, then the weight of S x will be how much? The double then the weight of S x will increase by at least double because the weight of a S x weight of S y is greater than that. So, weight of the new set will be the weight of this plus weight of this.

Now,. So, it will be double. So, that is why this two is coming that login is coming. So, that is the reason this login is coming. So, this is basically increase increasing the because the weight is increasing by the weight of S y, which is basically greater than S x so that means, weight is becoming double. So, that is the reason this login is coming login base 2. And if S x is more if S y is united with S x then we are not paying anything. So, then the how many times we are doubling? Then we are doubling just login time.

(Refer Slide Time: 18:13)

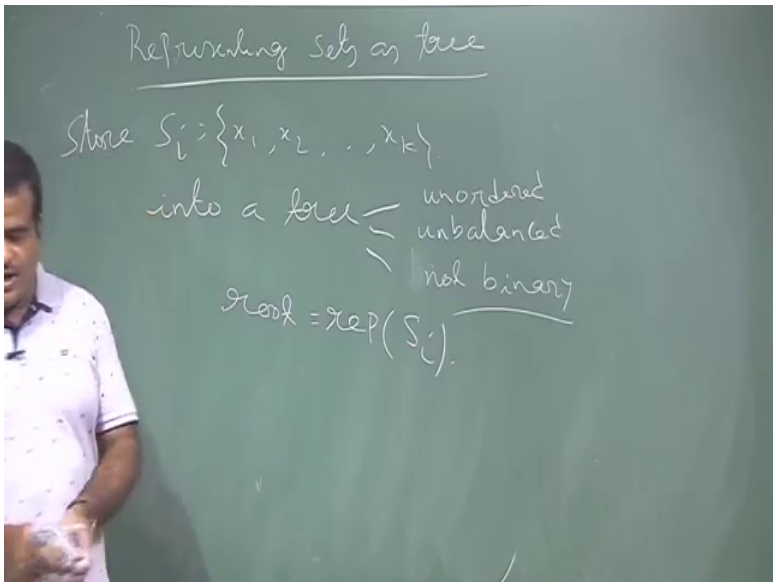


So, we are just making double is less than login base 2 times. So, this will the proof so; that means, that is why the login is coming because it is doubling. So, this is the proof. So, the total number of operation is. So, this is the one first trick.

Now, in order to bring the second trick, this analysis is amortize sense. So, amortize sense our total time complexity is the total union operation is L login, now if we have. So, we have taken the one x for that x it is login. So, there are n elements. So, it is n

login. So, this is the total cost in a amortize sense. So, now, average cost is login, but for few cases it is more like in a, but amortize sense it is average cost is login . So, now, we will discuss the second trick trick two for that we need to bring the data structure which is tree.

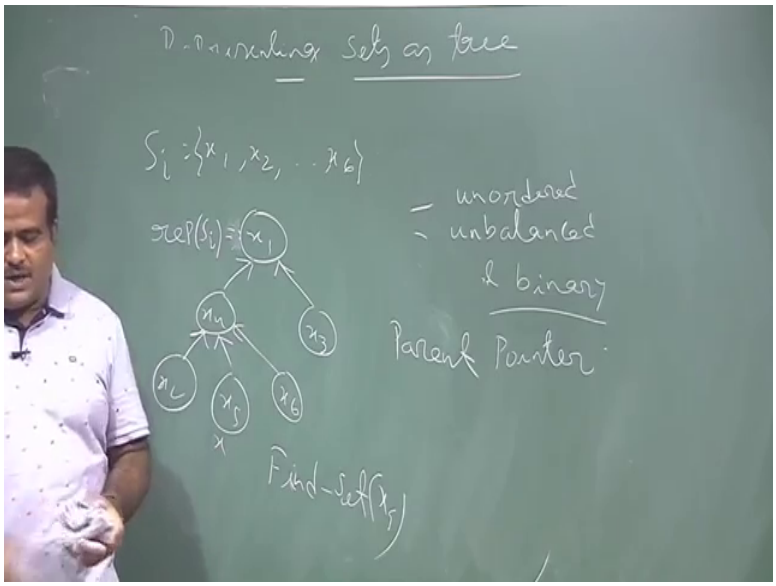
(Refer Slide Time: 19:35)



So, let us again bring the tree representation representing sets as tree. So, here we are not dealing with binary tree or it is not a balanced tree. So, suppose we have a set x, x 1 x 2 x k, now we store this into a tree and these trees unordered because our sets are unordered collections, unordered unbalanced here we are not bothering about balance ok.

So, and not necessarily a binary not a binary I mean it could be any tree, I am in the children it could number of children could be more than 2. It could be 3 4 it could be 2 also it could be null. So, it is not necessarily binary tree it could be turnery it could be I mean it is not necessarily binary tree and the parent is basically the root, root of the tree is the representative element rep of S i. So, basically you have a tree. So, let us take an example. So, suppose we have a set say x 1 x 2 dot dot dot x 6.

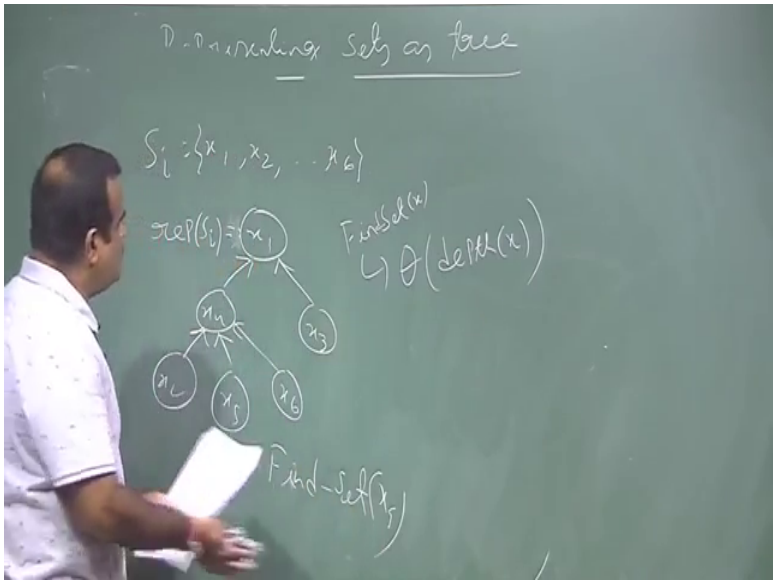
(Refer Slide Time: 21:18)



So, let us draw a tree x 1, x 4, x 3, x 2, x 5, x 6. So, this is not a binary tree and this is this is not a balanced tree also. So, we do not care about that.

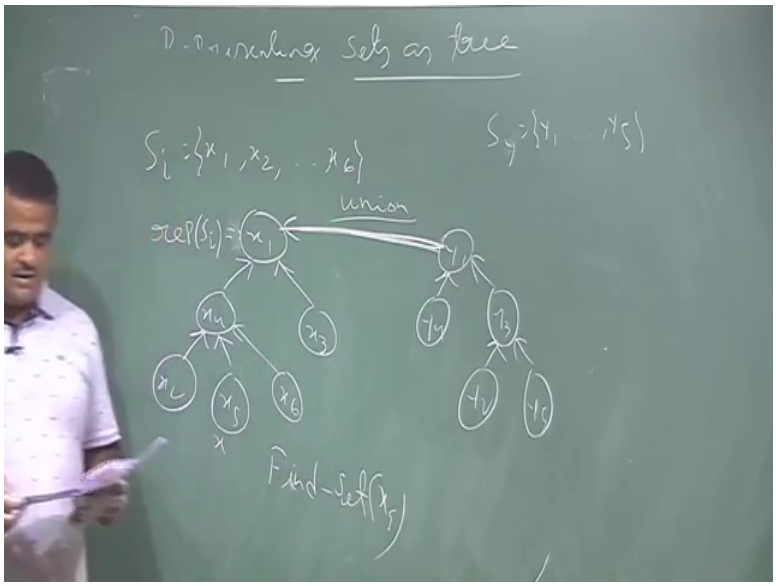
So, now we only have we do not have child pointer we only have parent pointer and this is the root rep of a S i is this one the root of the tree. So, we only have the parent pointer of each node. So, we do not care about the child because no need to care about the child pointer. So, this is the root. So, you only have the parent pointers. So, why we need parent pointer because to find the set? So, we have to return this root element. So, two if we have parent pointer. So, suppose you want to this is our x say, suppose we want to do the find set of x find. So, what we do? We look at the parent pointer this way we reach to the root and we written x 1. So, that will take height of the depth of x basically.

(Refer Slide Time: 23:03)



Now, this is our data structure, now how to perform this operation like make set make set is easy just we make a these tree I mean a single turn tree that is it, and the find set is this one an union. So, find set will take how much time? Find set will take order of depth of x, this is the find set cost find set will take order of depth of x. Now what about the union? A union suppose we have a another set y.

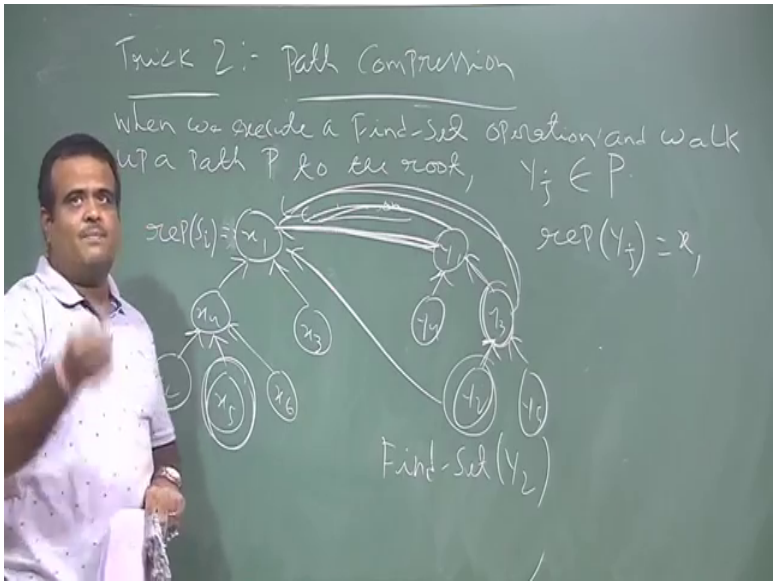
(Refer Slide Time: 23:41)



Y 1 say y. So, we have another set S y, y 1, y 2, y 5 say. So, y 1, y 4 y this is another tree we have y 2, y 5. So, now, this tree is also like this, we have all the parent pointers now how to find set.

So, we can add this root to any one of this node, that will do because if we add this root to any one of this node, then the we can just from this we go to that node and then we visit to the root or else we can just add this two here. So, this way we can do. So, this is the union operation we are doing. So, now, the question is if we adopt the trick one here. so; that means, if we just add the smaller set to the larger set then how it is helping us.

(Refer Slide Time: 25:11)



So, that is the next discussion we are going to do this is called. This set is completely unordered. So, this is the trick one adopted to trees. So, we adopted to trees for. So, that means if we just find the smaller is merging with larger. So, if we can find the smaller tree then we can add it to the larger set then the find will take the same time as find of x and height of the tree will increase because height of the tree will not increase because we are having smaller set we are adding merging with the larger set. So, height will remains same.

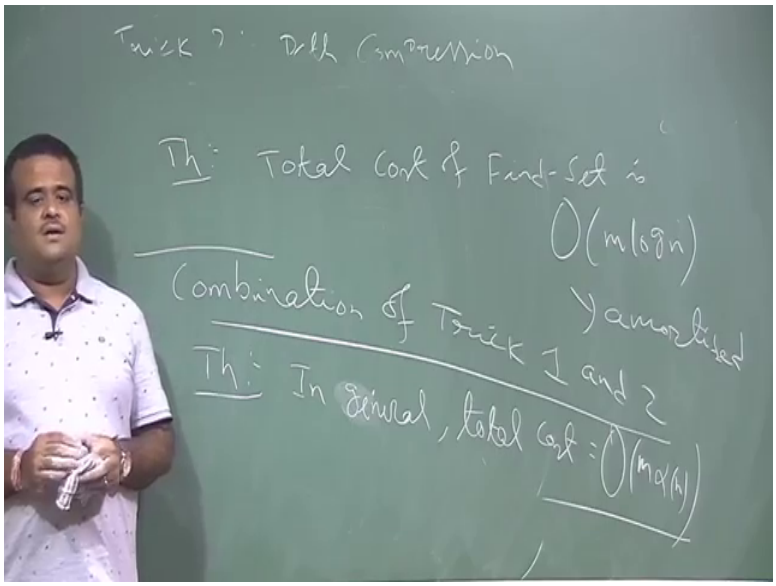
So, this is the trick one, now let us talk about trick 2. So, that is the trick 2 which is also referred as path compression. So, what it is telling? Now this is telling all about the find now. So, when we perform a find operation say we are performing the find operation on this set say x 5 or say yeah x 5. So, what say we are no we have say we are performing the find set operation on say y 2. See if you perform the find set operation on y 2 what we go to do we go to the parent of this, then we go to this, then we go to this. Now if we do this now. So, once we got the parent of this, we have parent pointer from here to here

this is the new augmentation we are doing. So, that if the next time we again search for the find y 2 then we will get it directly. So, this is a path not only this. So, we are going to add the parent pointer for each of this node in this path.

So, that is the idea of trick two. So, that is called compression path compression. So, let us write that. So, when we execute a fine set of a node, fine set operation and walk of a path P to the root. So, we know the representative. So, so we are going to change the representative element of each node in that path. So, if a node Y J belongs to this path P then we are going to change the rep of Y J to be x 1. So, we are going to have a. So, direct pointer to this. So, direct pointer to this all the elements in that path. So, this is called trick 2.

So, this is the operation we are doing. So, this is basically called compression path compression. So, instead of only one element we are just compressing the path I mean we are taking all the node, representative element of all the node, we are changing to this root. So, this we can have a theorem to. So, this is in amortize sense the total cost is. So, let us just write the theorem we then have time to prove it just will state the theorem. So, this theorem is telling.

(Refer Slide Time: 29:22)



The total cost of find set is order of n login amortize sense.

So, this is the amortize analysis. So, there are m times we are doing this operations. So, now, the on an average it is login. So, this proof we are not doing, now we are do not have time. So, now, this is if we only applied trick 2. Now if we combined apply trick one and trick one and trick two. So, that is the combination of trick 1 and 2. So, this is more interesting if we apply both the tricks then this is the general case, this is the another theorem in general if we apply both the trick the total cost is order of m. So, this we this proof is there in the code member and that going to do this book.

So, this is the again in amortize sense. So, this is the amortize analysis we do we did for our data structure augmentation and we apply the two trick one is the we are going to merge the smallers with the larger we are or we are going to combined the smaller with the larger and the second one is path compression. So, we are going to change each pointer of this representative element of each pointer of this set note on this path to the root.

Thank you.