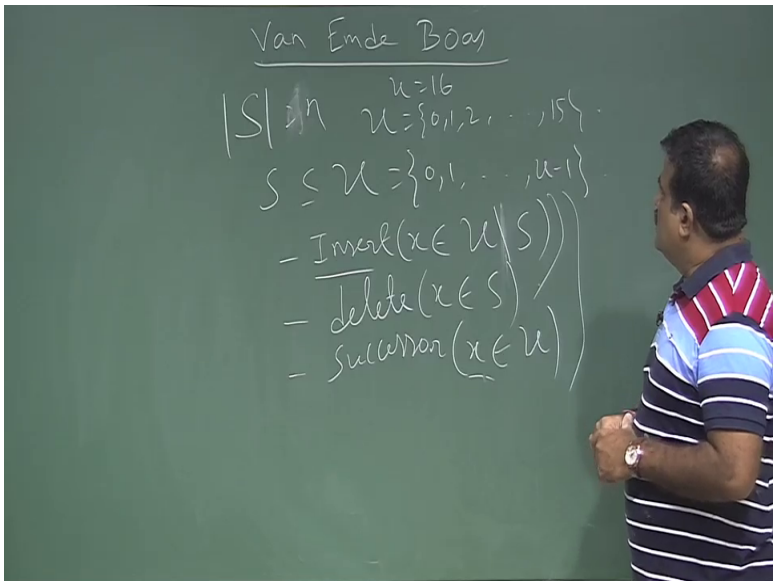
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 32
Van Emde Boas Data Structure

So, we are talking about fixed universe successor problem. Just to recap we have a, so the goal these to maintain the dynamic set S which is of n element the size of S is (Refer Time: 00:31) and S is coming from a fix u (Refer Time: 00:34) which is basically having u element. So, it is starting for u to 1. So, this is to; so that we can able to perform this operation we should able to insert an element x which is not in S. So, this is the insertion and we should able to delete an element from S and we should able to find the successor. So, we should able to get the next element of next element after x in S.

(Refer Slide Time: 00:29)

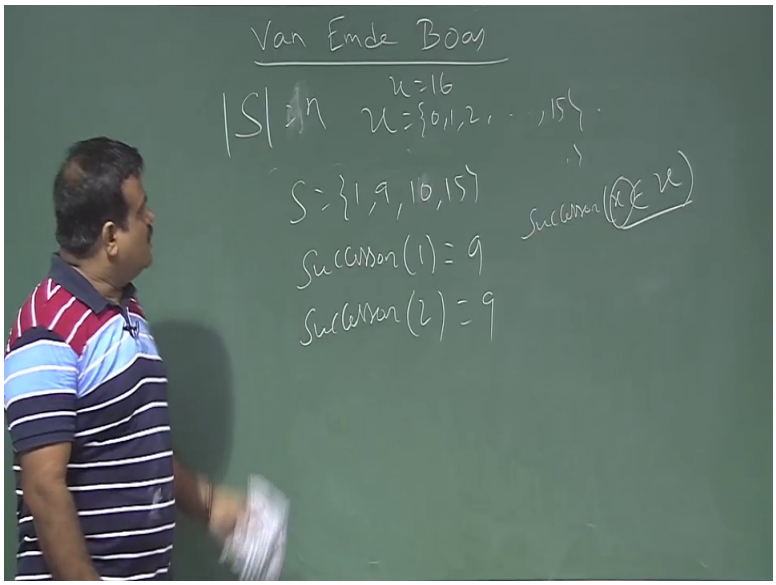


So, this operation we should able to perform and these two operation make this it is dynamic and this is the successor query similar we have a predecessor query. So, suppose S is a suppose u is 16. So, our universe is, this is fixed universe; that means, we know the elements of base we have seen the many dynamic set data structure to handle the dynamic set, but here the set is some specific elements are fixed we know the elements where from they are coming elements are coming from this universe we know

the universe and then we have to do this operations these 2 are dynamic operation and this is query operation successor query.

So, now, this operation now suppose S this is u and suppose S is a 1 9 14 15 15 say at some point of time.

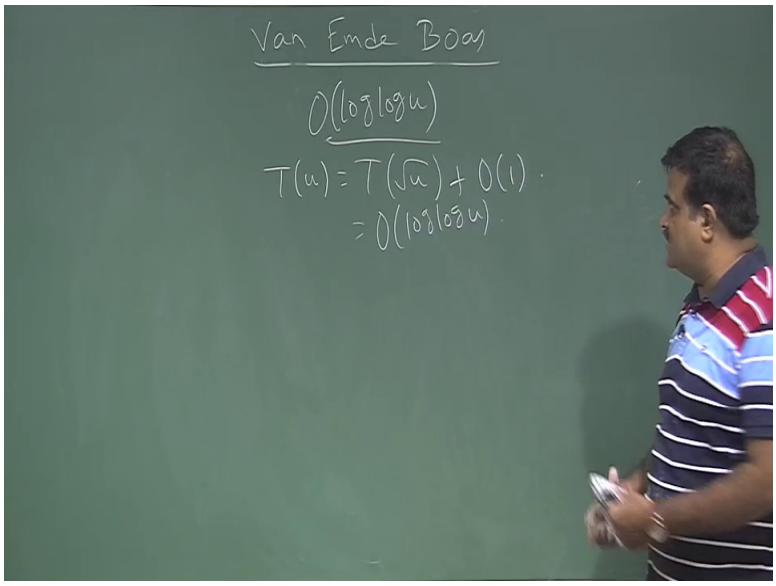
(Refer Slide Time: 02:09)



Now the successor of successor of 1 is what, successor of 1 is basically 9 and the successor of 2, 2 is also 9. So, successor of x, x mean the next element which is in S after this 2 and this x need not be in S, this x is coming from; successor of x x is coming from the universe. So, we can we can query the successor for any element from the universe need not be in S may be in S for example, this one, but may not be in S for example, this one ok.

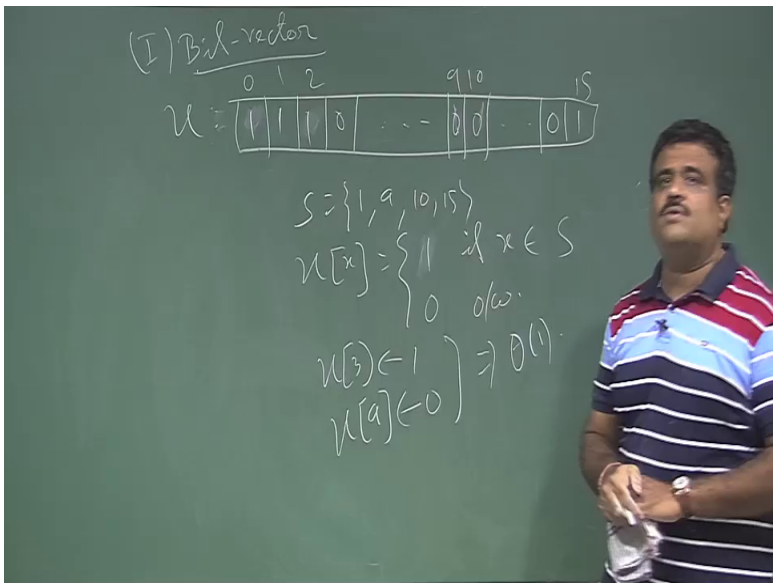
So, these problem we want to we have seen some data structure in the last class, but this problem we want to solve in log log u time the successor query or insert.

(Refer Slide Time: 03:14)



So, we have we know we have seen in the last class to get log log u you have to have this type of recurrence, this type of recurrence. So, we have to achieve this type of recurrence then we can have the solution. So, this solution is log of log n. So, the idea is from peter Van Emde Boas this was person who develop this data structure.

(Refer Slide Time: 03:57)

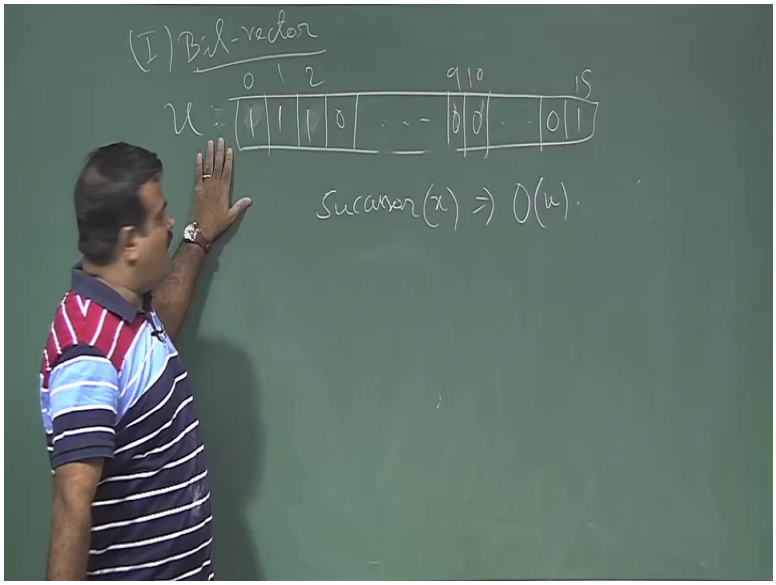


So, let us start with the, so the starting point is a array. So, this is a bit vector these we have already started in the previous lecture just to recap. So, we start with the starting point bit vector. So, what we are doing we are having a new array of size small in. So, we

just have an suppose our S is say 1 9 10 15. So, we have 1 here 1, say here 9 at 10 say dot dot dot dot dot, so 1 1 rate start 0. So, this is just a bit vector. So, light of light on.

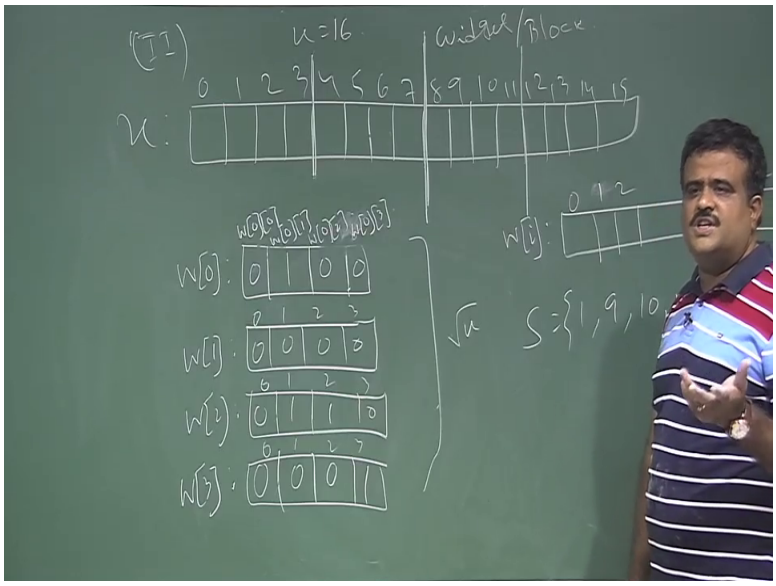
So, this array will be so u of x. So, u of x will be 0 if x in is e S sorry 1 if. So, switch on other wise switch off like this the room is empty. So, and we have seen the insertion and deletion will take which insertion and deletion will take the consentient because if we want to insert 3. So, we go to u 3 and we put it 1, if you want to delete 9 you go to u 9 and we put it 0. So, insertion and deletion both will take the consentient but the problem is with the successor. Now what is the successor query? Suppose our array is like this. So, all are 0 except the last one and few beginning one now suppose we want to have a successor query on 2 or 3.

(Refer Slide Time: 05:49)



So, it has to scan the whole array. So, successor query will take order of u. So, successor will take will take order of u, but we want to achieve this by log log u. So, for that we want to break it into square root of u. So, how we can break this array into square root of u array? So, that is the step to. So, that is the second step. So, we just, this is the array of size u. So, this is step 2.

(Refer Slide Time: 06:23)



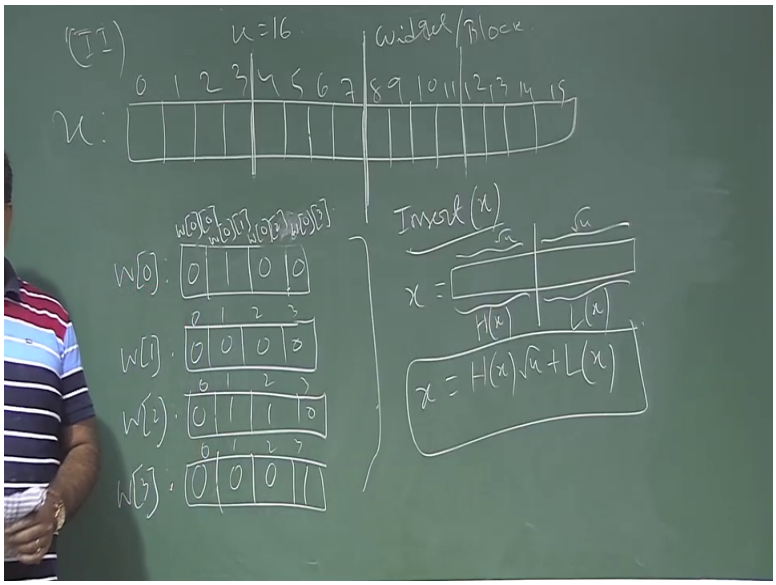
So, this is our u array. So, say 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15. So, our u is here our is 16 17. So, we have a array. So, how we can break it into square root of u square root of u spots what we do we just break it into 4 size blocks or this is called also widgets or blocks.

These the first block or 0 block 1 2 3 4 this is the next block this is the 1 block 1 2 3 4 this is the next block this the next block like this. So, this is our, this is the first block this is 0 we denoted by this then W 1 W 2 and then W 3 and this is. So, how many blocks are there how many blocks are there? So, there are root of bar of u blocks here u is 16. So, root of bar of u is four. So, there are 4 blocks and what is the size of each blocks size of each block is also root of bar of u. So, u started from so each blocks. So, W i is basically W i is basically in general. So, it is basically 0 1 sorry 0 1 2 up to root of bar of u minus 1 0 1 2 up to root of bar of u minus 1 this is the ith block. So, here u is 4. So, for the (Refer Time: 08:38) this is the situation.

So, we break it into, so basically the idea is this was 1 dimensional array. So, we are visualizing as, so we are breaking into 2 dimensional array. So, this is basically. So, this is this is again start from 0 1 2 3, 0 1 2 3, 0 1 2 3, 0 1 2 3 just 2 dimensional array in c you can see. So, this is basically W, this is basically W 0 0 like this. So, this is basically this is basically W 0 0, W 0 1, W 0 2, W 0 3 just 2 dimensional array. So, we break we just break the 1 dimensional array into 2 dimensional array ok.

So, now, our S is basically suppose our S is 1 9 10 15. So, this is basically 1 and 9 is basically where 9 is basically this is 1 2 3 4, 0 1 2 3 4 5 6 7 8 9 and 10 and this is 15 and remaining are 0, this is our data structure. So, now, how to use this data structure for our operational like insert delete?

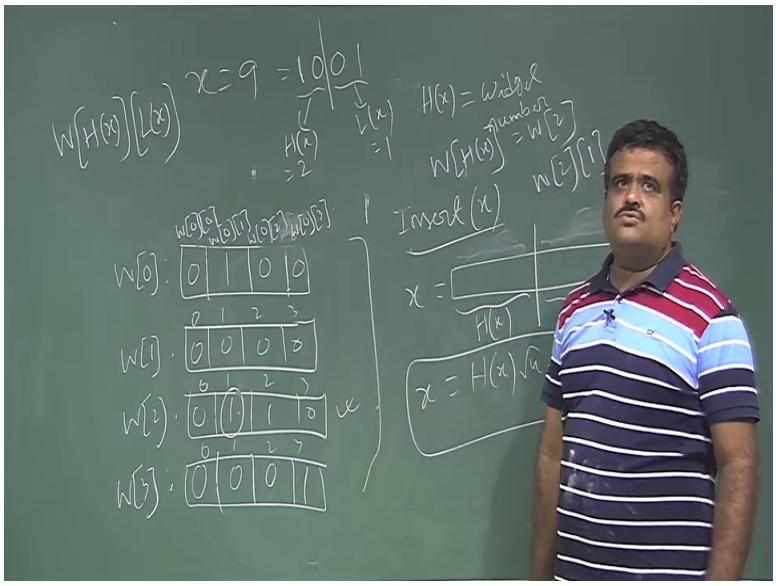
(Refer Slide Time: 10:32)



So, suppose we want to insert x. So, we want to insert x. So, what we do we just convert x into binary and we write x into 2 parts. So, we just convert x into binary and it will be of size root u and we take the this part is high of x and this part is low of x. So, basically x is high, high of x into root u plus low of x.

So, this is the square root of u bits and this is the square root of u bits. So, we convert x into binary representation and of u size and we take the first root u bit as a high of x and then next root u bit as a low of x.

(Refer Slide Time: 11:44)

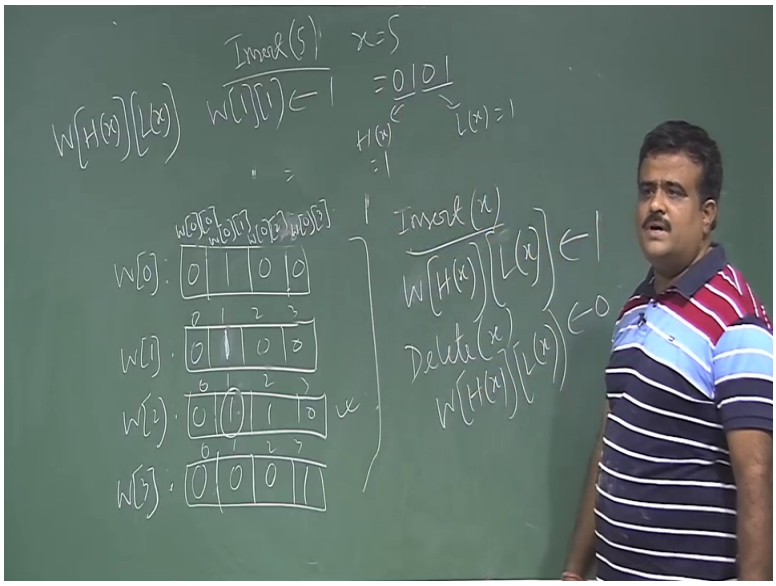


For example, if we takes x equal to 9 suppose x equal to 9. So, for 9, how we can convert? So, this is of u is 16. So, square root of u is basically 4, so 9 how we convert into 9 into binary 9 is basically 1 0 0 1. So, this is 9 plus 1 - 9. So, this is 9 now this is basically our this is basically our high of x and this is basically our low of x.

So, this we are, so this is basically, this not square root. So, this is basically, so first half the lower half is called low of x upper half is called high of x. So, this is basically high of x this is basically low of x and this high of x will give the block number. So, this is, work is H is 2 and this is basically 1. So, this is basically our position of 9 in this 2 dimensional array. So, because this is basically give us H of x will be x will basically give us is the block number widget number, basically W of H of x, here H is 2. So, basically W 2, W 2 is basically these block.

So, in W 2 we have to we have to see the in that block this basically gives us the position of x, this is 1, basically we need to look at this 0 1. So, these position is basically W 2 one. So, this is basically W of, W of H of x and low of x. So, theses with this is basically gives the position of x in this 2 dimensional representation. So, this is nice direction this is just 2 dimensional array. So, these will give us the position. So, to insert x, what we need to do? So, to insert x we will do just suppose we to insert x will go to. So, we break it into. So, x into 2 part high of x and low of x. So, we basically convert x into binary then we take the first half most significant bits H (Refer Time: 14:45) significant bits L.

(Refer Slide Time: 14:30)

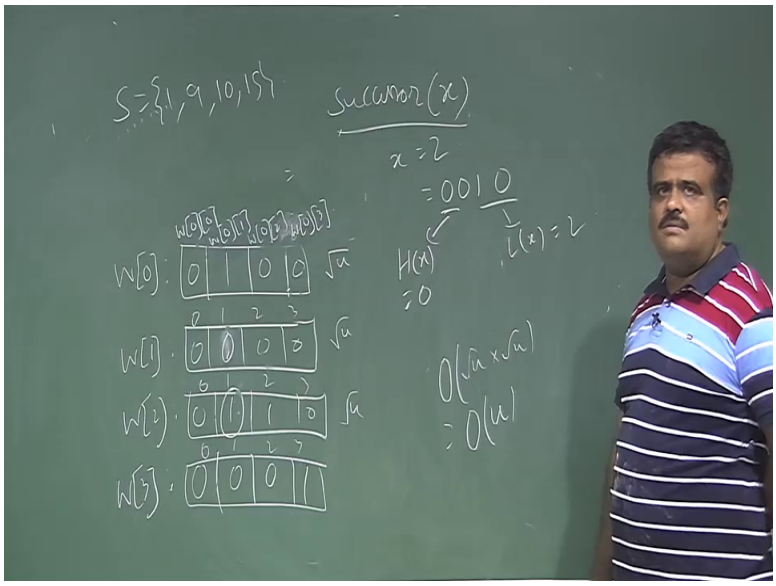


So, what we do we go for W of H of x then we go for low of x and then put it 1 that is it and similarly delete also if you want to delete an element delete an x. So, we just go to that position of x in the 2 dimensional array and in put it 0. So, suppose we want to inserts a 5. So, suppose x equal to 5 we want to insert 5 S here, here x equal to 5. So, what is the binary presentation of 5?

So, this is 0 1 0 1 this is 5. So, this is high of x this is low of x. So, this is basically high of x which is basically (Refer Time: 15:41) this is 1 low of x this is also 1. So, we have to just go for W of 1 1. So, W of 1 is this, 1 is this. So, this is basically (Refer Time: 15:57) into 1. So, this is the position for x equal to 5. So, you just put it 1 similarly if you want to delete 9. So, we have to convert into this high of x low of x. So, you go to the position 9.

Now, how much will take the successor query? So, the how to do the successor query suppose successor of x. So, suppose x is like this S is 1 9 10 15.

(Refer Slide Time: 16:22)



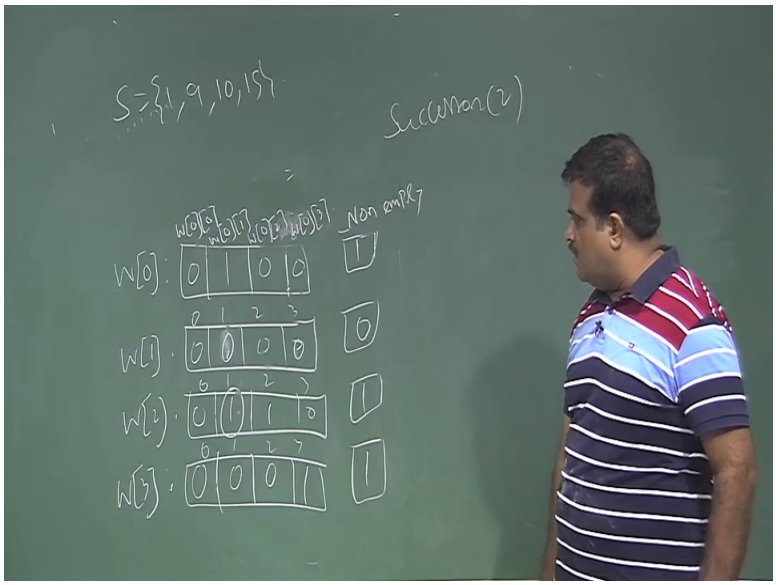
Suppose we want to find the successor of say 2 what to do. So, we just convert if suppose x equal to 2 when we find the successor of 2. So, what we do we convert 2 into this low of x high of x, 0 0 1 0. So, this is low of x this is denoted by is it block number and this is basically. So, this is 0 and this is 2. So, basically you need to find the successor after this. So, what we do we first find the successor in these and then we find the successor in these and we find the non empty bit in this after this and then we will find the non empty bit in these after these and then we will find the non empty bit in this like this.

So, to find the non empty bit in these it will take order of u time because size is order of u again these we may not get the successor here we may have to find the non successor here like this. So, then may be here also we need to locate like this or maybe we can get at the end. So, these way it will take order of root u into root u, so order of u.

So, now, again, but can you think some augmentation of this data say because we know this this this block is 0 block there is no 1 in this block. So, to search in this block is wastage of time. So, how to avoid that can you think of some augmentation in these data structure so that we can avoid to search in these blocks because we know there is no 1 in this block. So, I we can have this knowledge beforehand. So, if you can do some augmentation in the data structure. So, that will avoid search in that block. So, to do that will do some augmentation of the data structure. So, we will do the just a they non empty

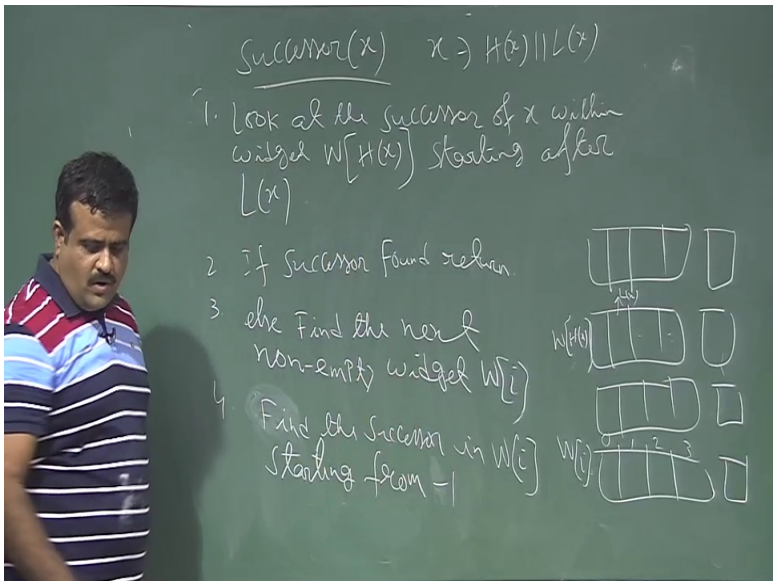
block I mean we just put a single bit information, this is the augmentation of the of this data structure we can put a single bit of information this will denote as say summary bit or this is denoting as the non empty bit whether these block is empty or this in this block there is somebody is there.

(Refer Slide Time: 18:51)



So, to denote this what we do we just or we can say this is the non empty. So, for these block there is 1. So, this is 1 this block it is 0 nobody is there these block it is 1 these block it is 1 now what we do suppose we have to find that. So, these will help us how now suppose you want to find the successor of 2. So, we first find the successor in that that block at 2 belonging. So, this is to this block. So, after these we find the successor if we are getting otherwise we will next look at the non empty block we know that there is, these block is empty so there is no question of looking at these block for one. So, we just go for the next non empty block. So, we just we know. So, we go for these block and then we find, we search for a successor in this block. So, that is the idea. So, let us write that code. So, this is basically the idea.

(Refer Slide Time: 20:34)

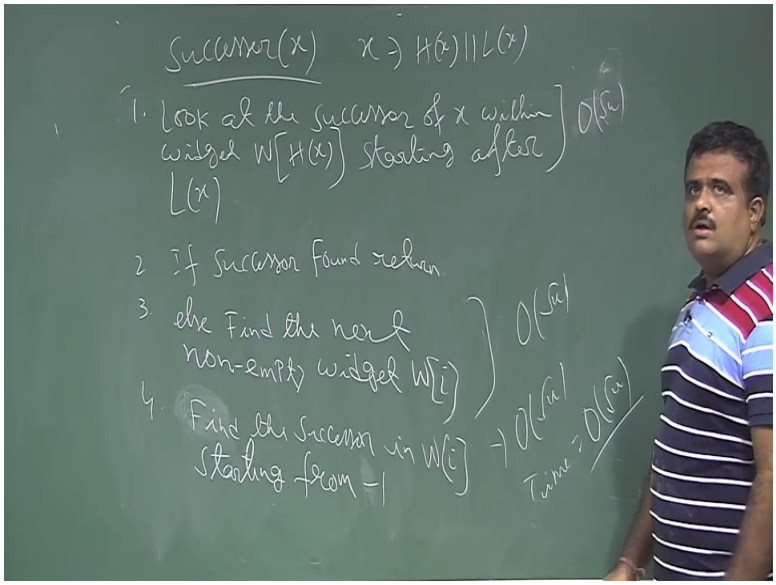


So, let us write the code this is the yeah. So, basically we are finding the successor of x. So, what we do we convert x into high of x and low of x. So, we look at the look at the successor of x within the widget of block W of high of x after v starting after starting after low of x. So, basically we have this blocks we have this blocks and we have this summary bits or non empty bits. So, this is our w. So, suppose this is the W of H of x this is the block where this x belonging and suppose this is the low of x position of x.

Now we find the successor after this in this block and if you are not getting then what we do will find the next non empty block. So, if successor found we return return otherwise else we find the next non empty block suppose else we find the next non empty widget and suppose and suppose that is W i suppose next non empty widget is say W i and then we know this in this W i there is a 1 we do not know where it is, but we know there is one, so then what we look at. So, we have look at 1 we then find the successor of what find the successor of say we start from the beginning find the successor of say minus infinity or minus 1 minus 2 because it is starting from 0 1 2. So, basically we are looking for first 1 here. So, basically we find the successor of say minus infinity or minus one. So, find the successor of successor in W i starting from beginning basically you are looking for first one in that block starting from starting from say minus 1 or minus infinity like this.

So, this is the code now what is the time complexity. Now these will take square root of u time and if we and then we found then otherwise we have to find the next non empty non empty use it which is have been one. So, 2 that is also sort of finding successor query. So, these also in take order of u this is order of u and these which is also take order of u.

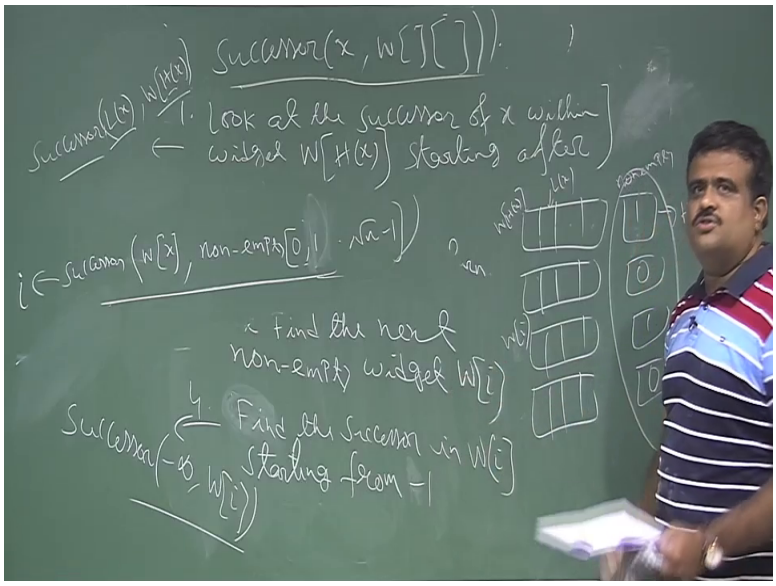
(Refer Slide Time: 24:26)



So, this total timings basically time complexity for these allegories order of square root of u sorry this is square root of u. So, this is the time complexity, but we want to do it log log u, so to for doing that we need to get a recurrence relations, how this algorithm can be thing for a recursive call. So, for that can you think this steps are in the recursive call. So, this is a successor calls a yeah. So, this is a successor call to successor in W, so W is a successor of x in 2 dimensional arrays it. Now this is (Refer Time: 25:22) this call is a, this is also successor call. So, this is a successor of successor of what successor of l of x in W of H of x. So, this is a successor call after l of x. So, this is a successor call successor of low of x in that block.

And if found otherwise this is also successor call this is basically J which is giving us successor of what? Successor of, basically what we are doing here we are looking at the, these are all blocks.

(Refer Slide Time: 26:13)



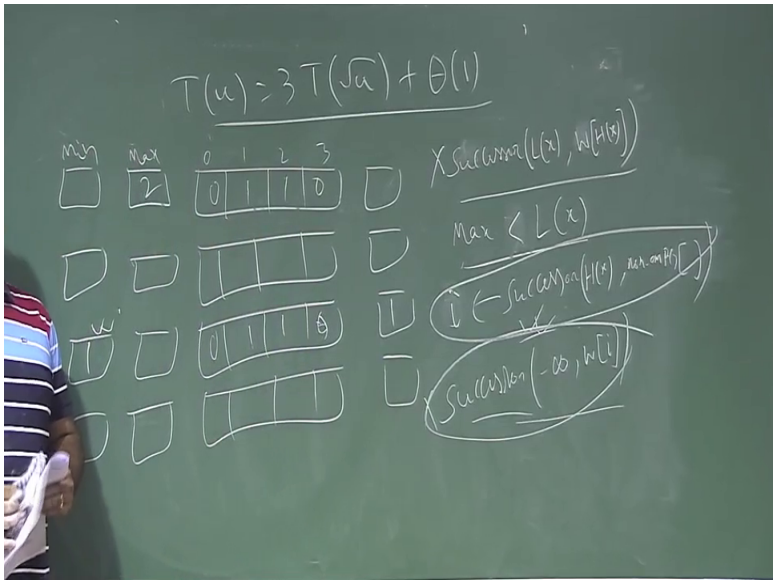
So, these are all blocks and these are all non empty bits. So, this first we have looking at that W of say suppose this is x is belonging here. So, this is basically W of H of x block and this is basically say position of l of x. So, we are looking at the successor here we found otherwise there we are looking at this non empty non empty array. So, we are looking at the successor in this non empty array after this h of x. So, the x is this call.

So, this is basically i is basically successor of successor of what after W of x in this non empty array non empty array. So, this is the successor call because we are looking at. So, here some 1 some 0s like this. So, we are looking for the next one in this array. So, this is the array of size W of W of u. So, this is array of size W of square root of u. So, 0 1 up to square root of u minus 1 we are looking at the successor after H of x. So, this position is H of x. We are looking at the successor; that means, next one. So, this is also successor call. So, we are looking at the next 1 after this H of x.

So, this is basically one. So, this is successor call and then we had here finally, we have a last successor call. So, this is one, this is the aeth block. So, this is basically W i. So, here we know there is a one, but we do not know where it is. So, we have to look at that one. So, that is also successor call successor of minus something because. So, that is basically also successor of the last one say minus infinity of W of i. So, this is also successor call. So, how many successor call? If we, in the first case we are not getting here then you have to go for this call and this call so 3 successor call, but we want to had all the 2

successor calls to how we can get it in 2 successor call. So, we have 3 successor call; that means, our recurrence is for this algorithm or recurrence is this a recurrence, but want all the 1 successor call. So, how we can achieve all the 1 successor call?

(Refer Slide Time: 29:04)



So, we need to do that is the idea of van emde boas we need to do some augmentation more augmentation in the data structure. So, how we can do? So, suppose this is the case, this is the non empty bit or the summary bits. So, now, suppose, now, i either, we are looking for the first successor call is basically successor of l of x in W of H of x. So, if we if we get it then just 1 call otherwise we have to go for 2 more calls. So, how to avoid this how to avoid this? So, we can do one thing, so how to avoid this call, to avoid this call we can just put the maximum with this is a another augmentation of the data structure.

So, we put a max array, max array. So, what is this max array is telling? Max array is telling we store the maximum index up to where this one is. So, suppose if is like this say 1 0. So, this is 0 1 2 3. So, this maximum will be 2 because we know up to 2 there is one. So, after 2 there is no one. So, that is the another augmentation of the data structure. So, like this. So, if you do this then these successor query is not require because we just first check yeah this required if max is get up than if max is get up than low of x then we know there is a 1 after low of x then there is a meaning of the successor call otherwise will not go for his successor call. So, these where we save 1 successor call because in

the, but in the worst case suppose it is not there suppose max is less than low of x then what we do will not do these successor call then we have to look at this non empty bit so that successor called these mandatory, so successor of successor of h of x into the non empty into the non empty array. So, these successor call is must it is mandatory it will look at the non empty bit over here, so 0 1 say 0 1 like this. So, this was 9. So, these successor call is mandatory 1 successor call mandatory.

Now next, next we are doing another successor call, like in this say, this is ith say. So, this is W i, i, successor of say minus infinity W i. Now can you think of some more augmentation so that you can save this successor call? Yes we can store the minimum we can store the minimum over here. So, where is the index where is the minimum. So, that is basically so this is basically 0 1, so this is 1 because minimum store here. So, these will save this successor call we do not need the successor call by getting the minimum values. So, these will give us the answer. So, this is the just 1 call, so we have to take this call to find the non empty bit. Either we find here if the max is max is greater than low of x then we know there is a 1 which is after low of x. So, it will do the successor call they are then we got the value successor otherwise we have to look at this, otherwise we will just do not call this successor call for this. So, we are save be 1 successor call, but we have to do this successor call to get the non empty bit non empty block. So, suppose this is the none empty block.

Now, earlier we are doing we are doing a successor call say minus infinity in that, but this is not required if we do the further augmenting by storing the minimum where the minimum index is, so this is the 1, this is 0 1 2 3. So, first one is here. So, this index we store and this index will be the output, so no need to do the successor call. So, this is a 1 successor call, so you reduce 3 successor call to 1 successor call by doing this augmentation, but here we have to be careful here we have storing the index it is not just the 0 1 vector. So, these would able to store the index. So, these are the integer array. This is the good news, so this is our recurrence and the solution of this recurrence is log log u. So, this is the idea of Van Emde Boas. So, this is the basically the data structure augmentation on the basic the bit vector array.

Thank you.