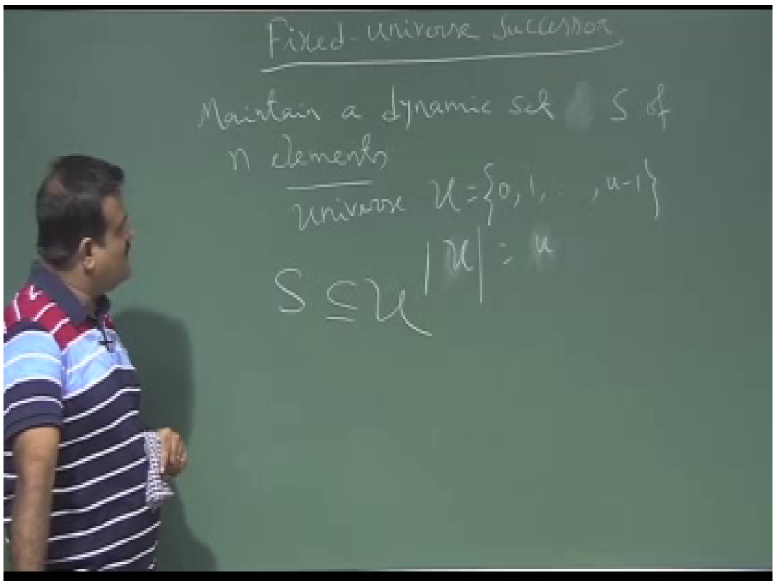
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture - 31
Fixed Universe Successor

So we will be taking about fixed universe successor problem, this is also a problem to deal with the dynamic set. So, the goal is to maintain a dynamic set S of n element.

(Refer Slide Time: 00:29)



Fixed-universe Successor

Maintain a dynamic set S of n elements

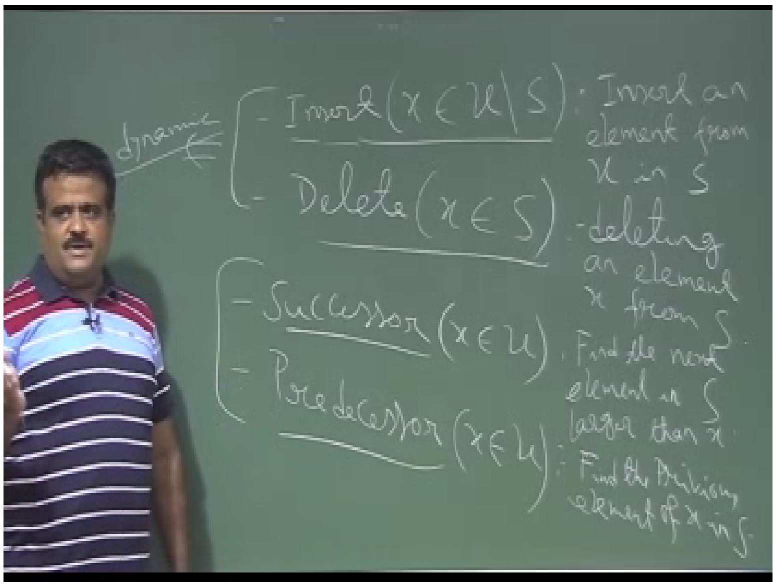
universe $\mathcal{U} = \{0, 1, \dots, u-1\}$

$S \subseteq \mathcal{U}$ $|\mathcal{U}| = u$

So, at some point of time the n elements in the set and this set is coming from a fixed universe. So, universe is fixed. So, we know the elements who are elements. So, this is a. So, we have seen many dynamic set problems like we have seen the heev. So, we have seen the binary balance binary tree balance bst. So, we have seen many dynamic set problem, but here problem is say I told you about the elements are coming the from the fixed universe. So, we know the elements where from they are coming, and the elements are coming from the universe U which is basically 0 1 to small u minus 1 and the size of the universe is size of this universe is small u. So, this is the problem.

So, S is a subset of u. So, the elements of S are coming from a fixed universe. So, we know the elements of S. So, this set is dynamic. So, our goals is to perform the operation like. So, dynamic operation what are the dynamic operation? We should perform the insert delete and also the successor and predecessor. So, this in the in S set. So, we should able to perform this operation say insert.

(Refer Slide Time: 02:20)



dynamic

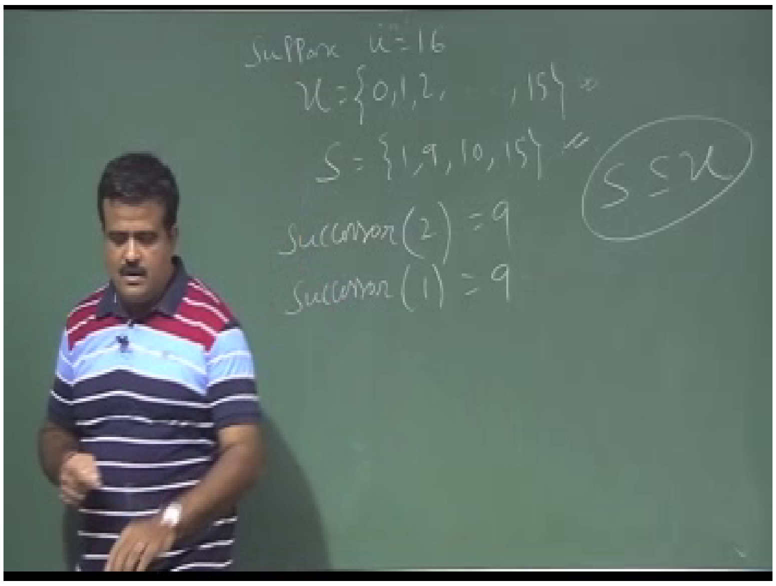
- $\text{Insert}(x \in \mathcal{U} \setminus S)$: Insert an element from $\mathcal{U}$ in S
- $\text{Delete}(x \in S)$ - deleting an element x from S.
- $\text{Successor}(x \in \mathcal{U})$. Find the next element in S larger than x.
- $\text{Predecessor}(x \in \mathcal{U})$: Find the previous element of x in S.

Insert means we should able to insert a x which is from the universe, and which is not in s. So, this operation we should able to do. So, this is basically insert an element in S, insert a element from u in S. So, this is basically insert operation and another dynamic operation is delete. So, we should able to delete a element from S. So, this is basically deleting an element x from S. So, these two operation is basically the dynamic operation make this set dynamic. So, these two operation make this set dynamic. So, another operation is the successor query.

So, we want to find a successor of x, x is coming from this. So, we want to find a successor means next element after x in s. So, that is the successor query. So, this is basically find the next element in S larger than x. So, x maybe in S or maybe from the universe. So, we have to find the our next element of x in S we will take an example and similarly the another query is finding the predecessor. So, the previous element find the previous element of x in S.

So, these two query you must able to do successor query and predecessor query. So, we need to have a data structure for these dynamic set S. So, we need to think of a data structure so, that we can perform these operation in a efficient way. So, let us give an example suppose our universe is say 0 to 15. So, our u is 16ok

(Refer Slide Time: 05:54)

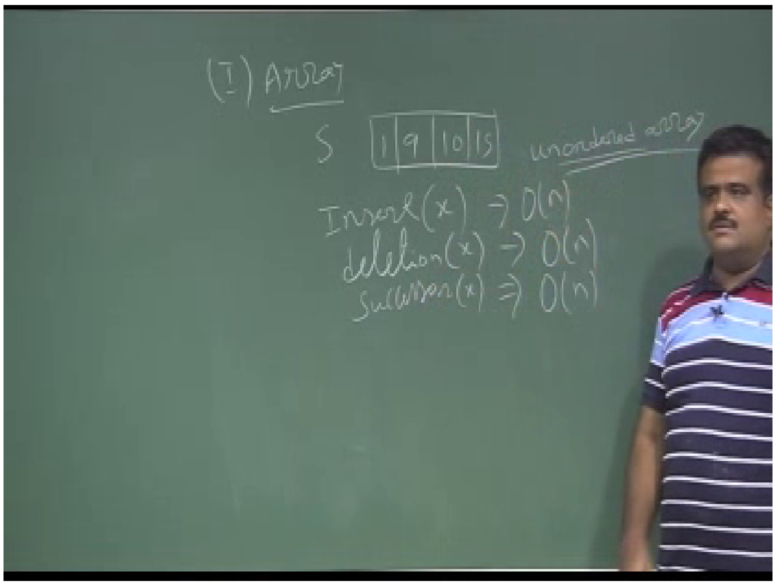


So, suppose small u is 16; that means, our universe is 0 one to up to 15. So, our set is coming from this universe, we know the dynamic set element dynamic set is whose elements are coming from this.

Now, suppose this we have S say for at some point of time S is a 1 9 10 15 suppose this is our S at some point of time. So, now, if we what do you mean by successor? Suppose successor of successor of say 2. So, successor of two is basically. So, we have to find the next element of 2 incase. So, after 2 we have nine. So, successor of 2 is 9 similarly successor of one this is also 9, we have to find the next element of this x, x could be any x or may not be an S. So, but we have to find the next element after x, which is the next element in S. So, that is the successor query similarly predecessor query and so, we need to have a we need to find a data structure, we need to think of data structure for this dynamic set S so, that we can able to insert an element x delete an element x and the, but the here the this is fixed universe; that means, universe is fixed. We know the element are coming from this universe the elements are coming from this set. So, S is always subset of u. So, that is called fixed universe successor problem. So, this is called fixed universe. So, now, what type of data structure you can think for this this problem.

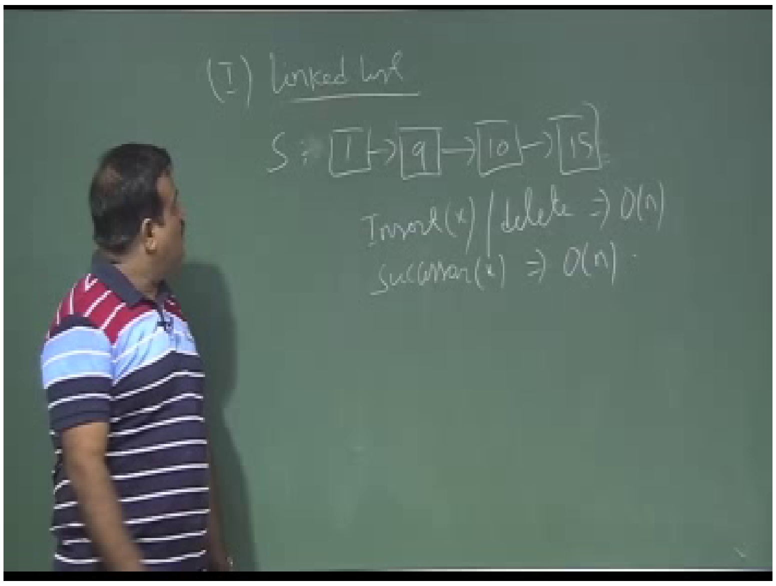
So, any suggestion for data structure like array. So, first suggestion is a array.

(Refer Slide Time: 08:10)



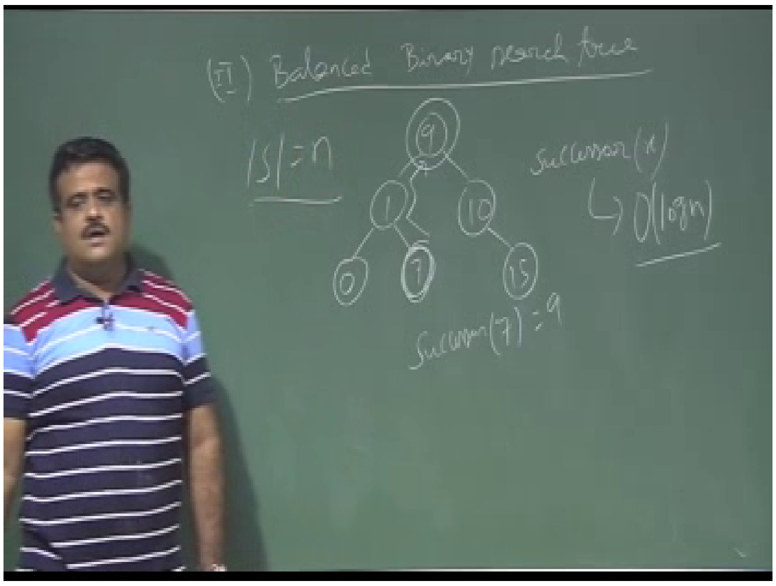
So, suppose we maintain a array for S. So, we have S is basically (Refer Time: 08:21) we have say 1, 9, 10, 15. So, we have invented array I mean n dimensional array I mean it could increase. So, it is a array. So, if it is array then how much time it will take to insert an element. So, this is not a sorted array I mean we may not have sorted array just a unordered array unordered array, or we can make it sorted then problem is to insert we have to again shift this. So, insert it we will get order of n times, similarly deletion we will take order of n time and if it is unordered array then the successor query successor of x will get also order of n time because we just need to scan the array I mean we do not know whether it is greater than or not so.

(Refer Slide Time: 09:41)



So, maybe we can think for linked list if it is linked list let us think for linked list another list. So, we have this S set we will just maintain a linked list for this S set. So, 1, 9, 10, 15. So, again for unordered if it is not sorted if it unordered then insert, we will take depending on how you will maintain the point insert or delete. So, we have search that element for delete. So, this all we will take order of n and even successor also successor or predecessor, we will also take order of n; because we need to find the next element this is unordered array unordered linked list. So, we have to find the next element after this.

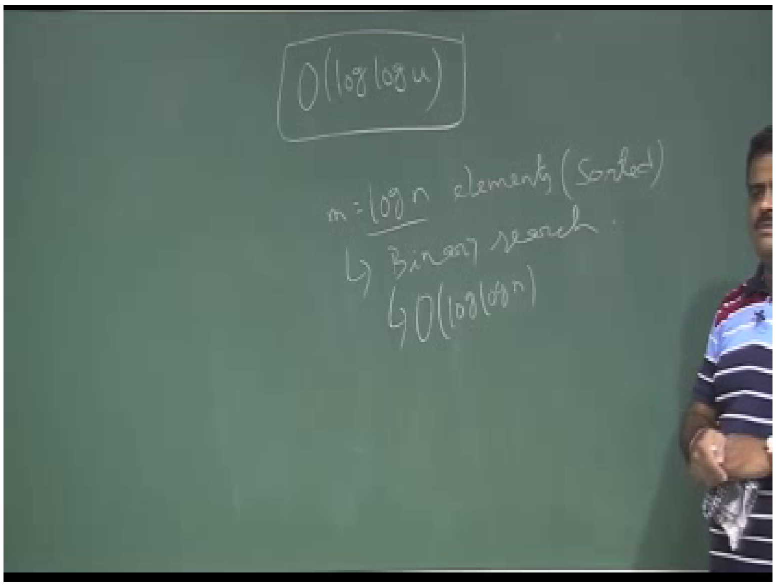
(Refer Slide Time: 10:49)



So this is second option linked list now think about binary search tree, balance binary search tree. So, if you maintain this in a balanced binary search tree balance bst (Refer Time: 11:00) balanced binary search tree. So, we store this into the balanced binary search tree. So, 9 maybe 1, maybe 10, maybe 15 like this now, the search will take. So, insert will take log n time because you have to just insert in the bst, and again we may need to make it balance. So, if you are using (Refer Time: 11:33) it is just a (Refer Time: 11:34) insert deletion is also same. So, and successor also successor means. So, successor of. So, this is a binary search tree. So, suppose we have few more element like say we have here say we have a 9.

So, we have say seven and say we have 0 over here. Now if we ask which is the successor for this node suppose successor of 7. Suppose successor this is basically inner traversal who is the next element of seven. So, it is basically we need to go like this successor of seven is 9. So, basically we need to go height of this tree. So, this successor will take basically order of log n because it is balance tree. So, height will be log n. So, successor query take (Refer Time: 12:42). So, for successor of 7 we need to go to the height of this tree. So, this a order of log n if size of S is n, but now, but we want do we want to have a data structure so, that we can perform this successor query or insert or delete we can perform this in order of log of log u. So, this is our target.

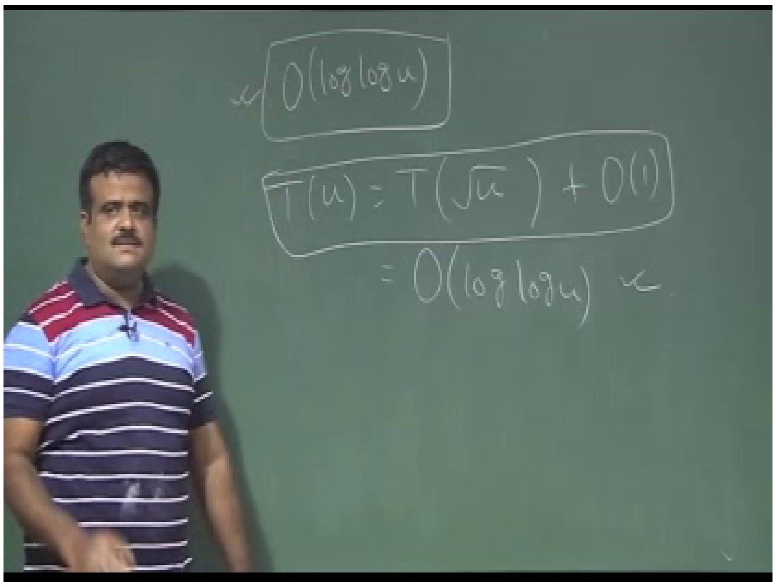
(Refer Slide Time: 13:12)



So, we want to have a data structure for which we can we should able to find the successor or we should able to insert in order of log order of log log u. So, how to get the time complexity log log u?

We have seen it is not log log a log log yeah. So, how we can get a time complexity of algorithm log log a or log log u any idea how you can do it log log? Suppose we have say a log n elements and they are sorted and now these suppose this is a and on these elements we are doing the binary search then we can get this binary search will get order of log log n. So, this binary search will take order of log log n. Any recurrence can give us this solution order of log log n, do you have any recurrence can you think of any recurrence which can give the solution of order log log n? Like T of u equal to T of something plus order of one can you suggest this so, that we get order of log log u.

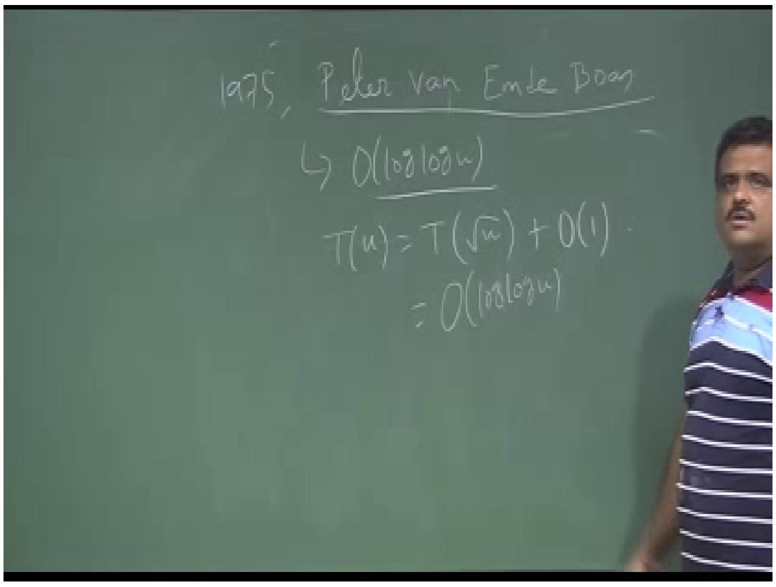
(Refer Slide Time: 14:40)



Any recurrence we will have this solution just think about it, if you put just root u. If you put just root u and you just replace root u by log u, then this is log u then half will come out. So, this will give us the solution of this will give us order of log log u by master method.

So, we can easily check this. So, this recurrence have the solution order of log log u. So, somehow you need to achieve this recurrence in order to get the time complexity this. So, that is the goal and this is the data structure which is invented by van peter van Emde boas.

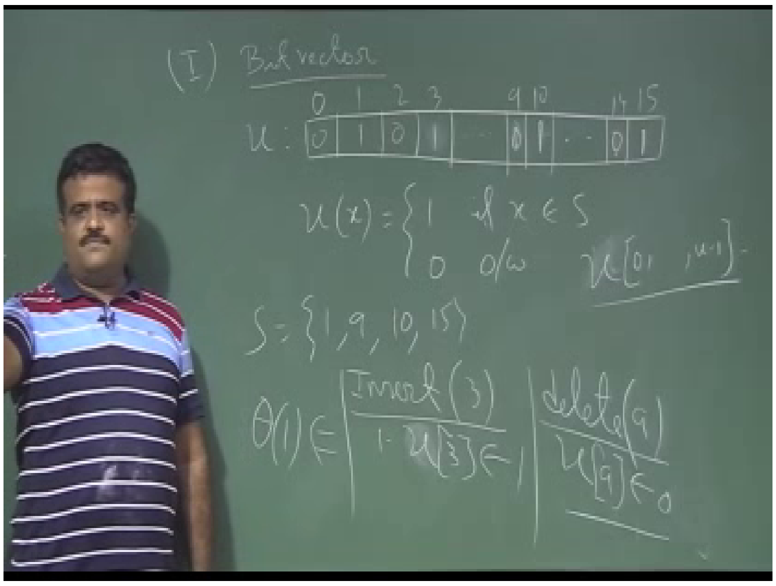
(Refer Slide Time: 15:55)



So, the person is in 1997 this data structure was invented by Peter van Emde boas. He solved this problem and he suggested a data structure which is basically the successor query or the insertion in the time is order of log log u. So, we have to get that data structure. So, we have to build that data structure. So, how we can build the data structure. So, we have seen that to get this you need to have this type of recurrence, T u is equal to. So, this recurrence are the solution order of log log u. Just you put u is equal to log u then by master method we get this solutions we get the solution ok.

Now, so, let us start. So, for the starting point we need to think of a array data structure simple array data structure.

(Refer Slide Time: 17:19)



So, we have already seen an array data structure, but this array is for on s. So, can you think of this is the starting point. So, can you think of some other type of array for this problem? So, can you think of a array of. So, we have universe of size u. So, can you think of a array of size u. So, this is called bit vector. So, what is a bit vector? So, we just went in array of u size. So, this is basically. So, our u is 15. So, this is upto 15; 0 to 3 like this dot dot dot.

Now, how to insert? So, to insert we just. So, if u of x is 1, if S in case 0 otherwise. So, just a bit vector I mean switch on switch off very nice data structure 0 one vector like these are room if person is there then we switch on the lights of the room otherwise we switch off the light. So, if our S is a 1, 9, 10, 15 then we have corresponding to this positions. So, this is 14 this is a dot dot dot this is 9 this is ten dot dot dot this is 14.

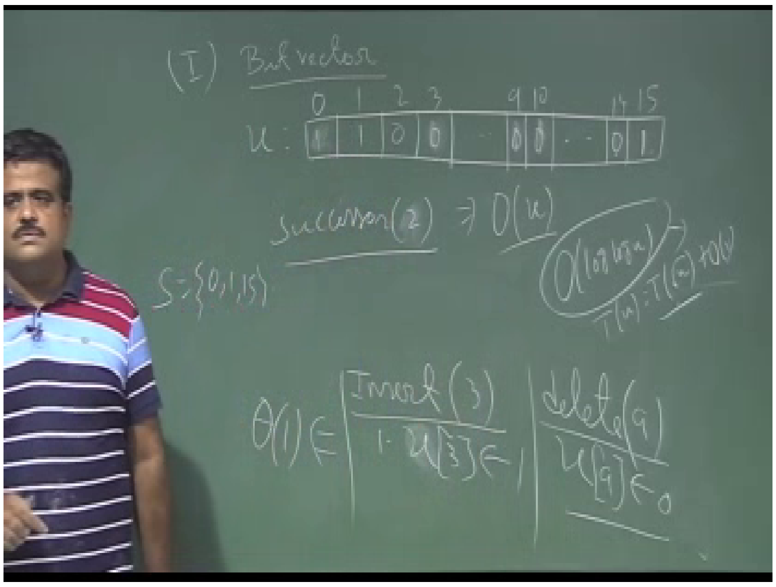
So, we have one over here one over here and 14 we have sorry 14 will be an one over here remaining are all zeroes. So, this is the current position of this data structure u. So, this is a simple array, this is a array of this is called bit vector just a switch off switch on 0 very nice very simple data structure very effective. So, we just have a array of size n. So, u is a array of size capital U this is a array of size. So, we start here array from 0 to 0 like in c, in c language our array starts from 0 to u minus 1 ok.

So, now if we use this data structure so, how to insert how to insert an element. So, suppose we have to insert say insert 3. So, what do we do we just go to u 3 and we put.

So, insert three means. So, we just go to u 3. So, this is the u array u 3 and we put it one switch on. So, we put it one very simple and delete.

Suppose we want to delete an element which is already there suppose you want delete say 9. So, how to delete it? We just go to that position 9 and we put it 0 switch off the room is empty nobody is there in that room lights off. So, this is very simple data structure, but very powerful. So, this the. So, how much caused and the these two insert and deletion will take both we will take? Constant time just we are going to that corresponding that field and we put it 0 or 1 depending on we are deleting it ok.

(Refer Slide Time: 21:22)



Now, how to do the successor query for successor query how time will be it will take successor of say successor of x. Now suppose all the elements are 0 except the last one and suppose we have to find the successor of one set or maybe we have one here. So, we have suppose a array position is s, 0 1 and 15.

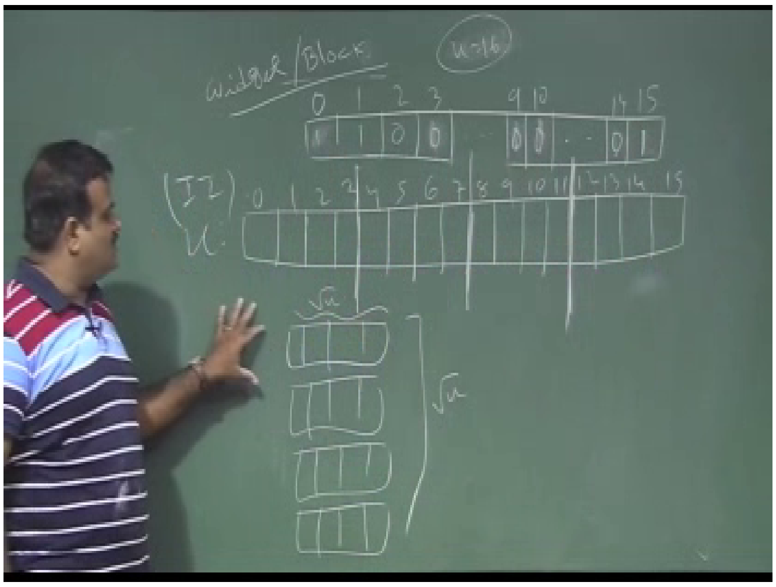
So, for successor query basically we need to scan whole array and this array of size u. So, this will take first case order of u because we may need to scan the whole array if the array is like this. Suppose you are searching an element our x is such that which is the after this nobody is there except the end. So, that is the that is even we if search the successor of say two, this is also we have to scan whole elements after two. So, that wille take order of this, but we want to achieve by this is not even log u.

So, we want to achieve this by log log u, but anyway this is starting point this is good start, this is very simple data structure because deletion and insertion is very powerful just a constant time, but only problem is with the successor. Now how we could we can make it into root u because our goal is to achieve this in log log u, in order to get log log u we have to have the recurrence this is our goal.

Log of log u this is the goal you want to achieve. So, in order to get this we need to somehow covert this into root u. So, how to convert this into root u? So, that is the question. So, we have given array. So, how we can make it into square root of u? So, we have a element of u we have element of sixteen over here u is 16 for our this example u is 16small u is 16. So, how we can make it into root u I mean root u is 4. So, any idea how to get a root u from this. So, that is the next step.

So, how we can give them a array how we can make it root u. So, this is step 2.

(Refer Slide Time: 24:13)



So, to make it root u what we can do? We can just take a square I mean if we have a 4 by 4 matrix, then it will give us 4 into 4 16 element. So, we have to make a square we have to make a square. So, what we do? We suppose this is our array of. So, here array is 15. So, 0 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.

So, this is our u array. So, this is our u array and here u is small u is 16. So, now, we want to make it into square root of u. So, how we can do it? We want to break it into the blocks. So, basically square. So, we want to make it into blocks of 4. So, this is first block this is called blocks or widget widget or blocks. So, this is first block 1, 2, 3, 4 this is second block or second widget 1, 2, 3, 4 this is the next one 1, 2, 3, 4. So, just we will break it into the blocks then we have. This first block 1, 2, 3 second block third block and the fourth block like this. So, this way we have total u this way we break it into. So, how many blocks are there square roots of u and what is the size of each block square root of u.

So, this the way we get the square root I mean from u, we get square root from u. So, this then we will talk about how we will make the insertion and deletion in this data structure, this is the just the argumentation of this data structure. So, this is just the one dimensional array this is two dimensional array. So, we will continue this in the next class.

Thank you.