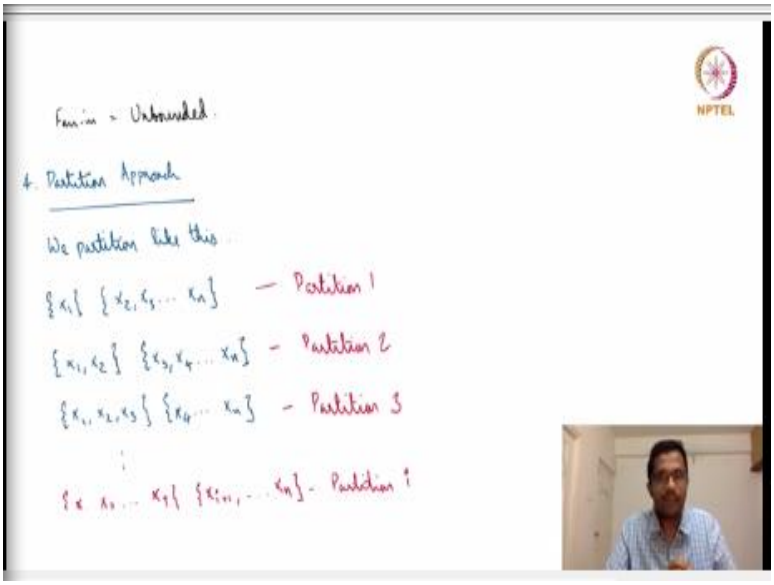
Computation Complexity Theory
Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology-Hyderabad

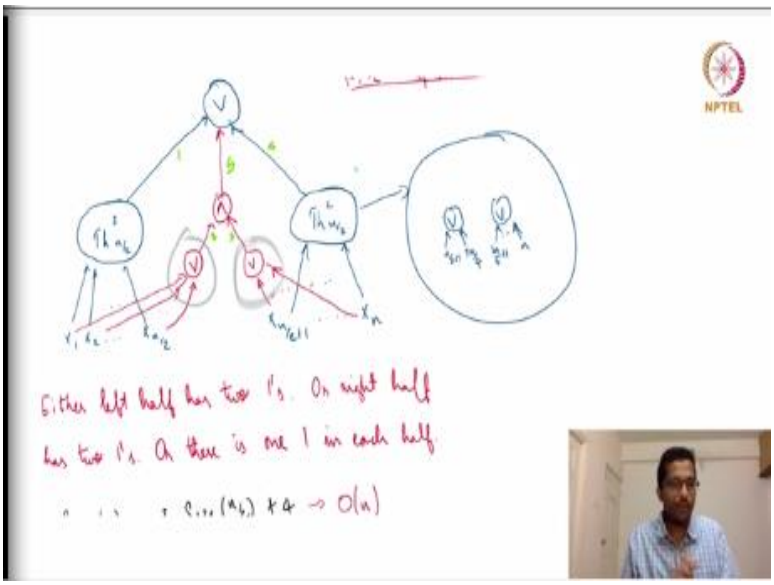
Lecture-34
Circuit Complexity: Part 2

(Refer Slide Time: 00:15)



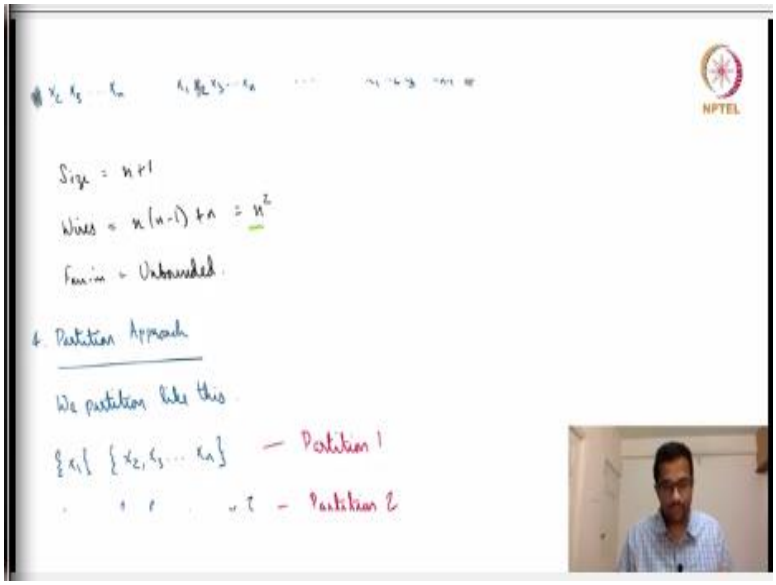
Hello and welcome to lecture 34 of the course computational complexity. In the last lecture we saw the threshold 2 functions which require 2 of the input bits to be 1. We saw 3 approaches using circuits on how to how to decide that. Now we are going to see another 3 approaches to decide the threshold 2 function in this lecture. So, in the previous lecture we saw Brute force approach, hiding approach and the recursive approach. So, the approach that we will see first is the partition approach, so again the key thing is that we there has to be 2 ones in the input.

(Refer Slide Time: 01:12)



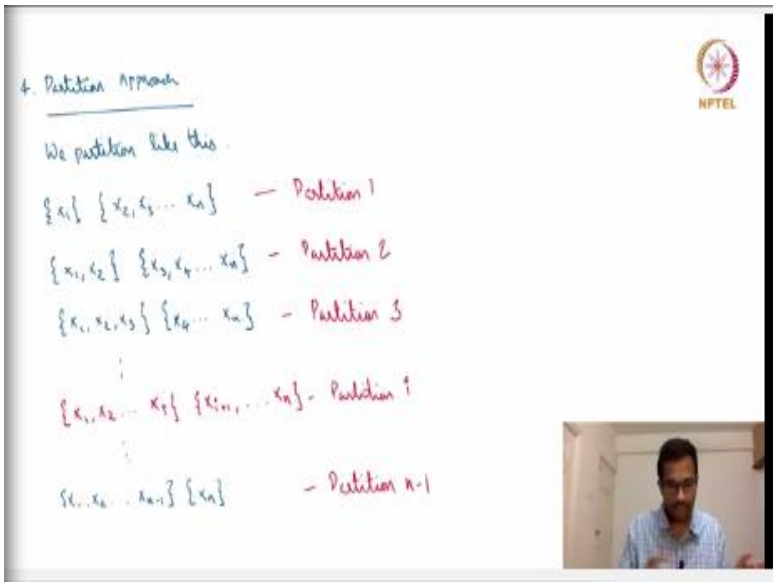
So, in the recursive approach we divided into 2 halves X 1 to X n by 2 and X n by 2 + 1 to X n.

(Refer Slide Time: 01:23)



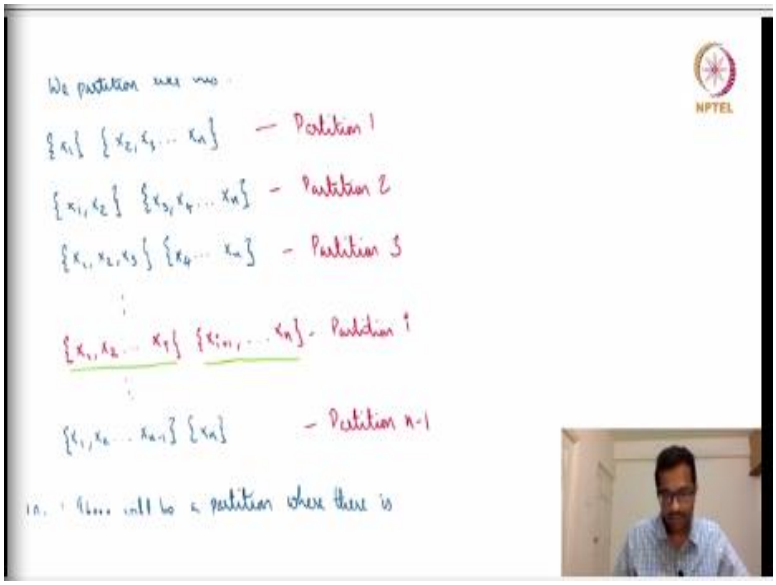
Now another way of doing that could be to and then did this recursive thing. Another way is to try all the possible partitions.

(Refer Slide Time: 01:30)



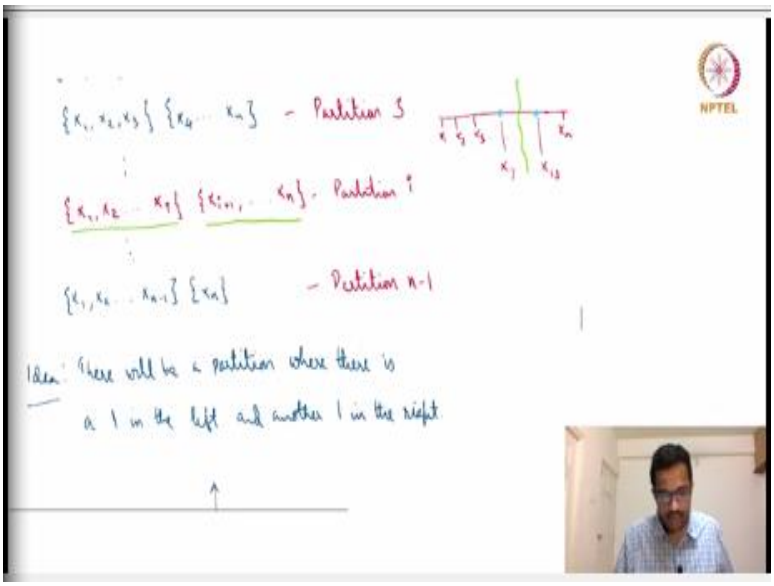
So, one way is to X 1 on one side and everything else on another side X 1, X 2 on one side and X 3, X 4 of to rest on the other side, X 1, X 2, X 3 on one side and so on.

(Refer Slide Time: 01:43)



So, we can consider partitions like this where we consider up to a certain i, so in general situation this is X 1 to X i on one side and X i + 1 to X n on the other side. So, there are n - 1 possible such partitions that we can consider, X 1 on one side, X 1, X 2 on one side, X 1, X 2, X 3 and so on up to X 1 up to X 1, X 2, X 3 up to X i on one side, this is the general case.

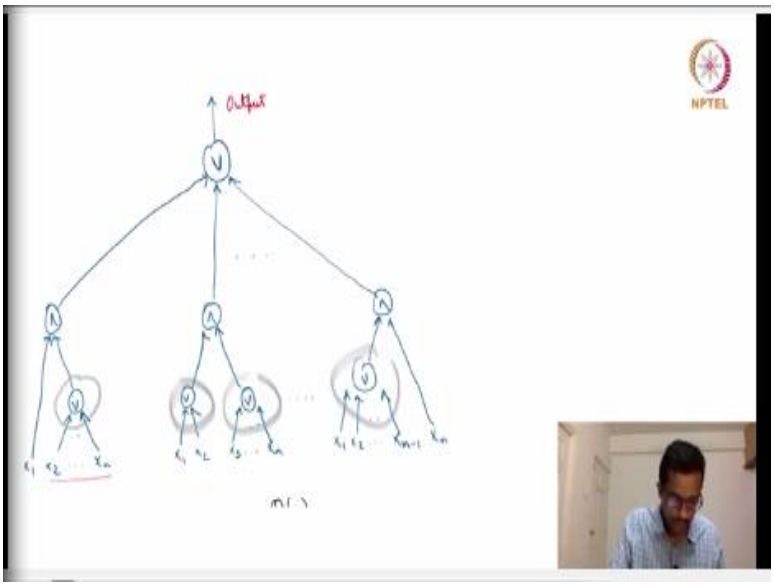
(Refer Slide Time: 02:11)



So, there are X there are n - 1 such partitions we can consider. If this is a yes instance of the threshold function meaning there are 2 values X i and X j for which both the bits X i and X j are 1. This means that for at least one of these partitions both sides should have at least 1 1. So, just to illustrate this; suppose this is the X 1 X 2 and so on X 3. Suppose let us say there are only 2 bits which are 1, let us say this is X 7 and let us say this is X 13. So, now consider any partition that breaks it between that, so now the left half has X 7 and the right half as X 13.

So, basically if X i and X j are the 2 bits that are 1 then where X let us say without loss of generality X j is bigger than i. Now consider the ith partition which takes X i on to one side and naturally since j is bigger than i j will be on the other side. So, this partition the ith partition will split the 2 ones into 2 halves then all that we need to check is, is there a 1 in both the halves? Is there at least 1, 1 in both the; halves or both the parts? These are not really halves, so this is the partition approach.

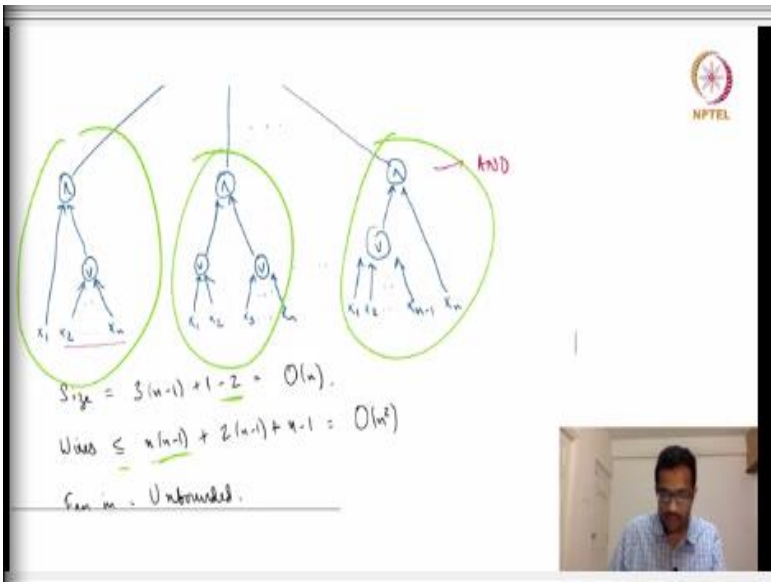
(Refer Slide Time: 04:05)



And let us try how to formalize this? So, again I have said the basic idea which is to check whether there is a partition whether there is a one in the left and one in the right. So, we do this, again this top one is the output, so we try all possible partitions X 1 on one side and X 2 X n on the other side X 1, X 2 1 one side and X 3 to X n on the other side and so on up to X 1 X 2 up to X n - 1 on one side and X n on the other side.

And then in each of these partitions we take the OR of each part, so here X 1 is alone, so there is no OR in the first part. X 2 to X n is one part, so we have this OR gate, in the second X 1 X 2 form an OR gate, X 3 to X n form an OR gate. In the last X 1 to X n - 1 forms an OR gate and obviously X n is alone, so there is no need of an OR gate.

(Refer Slide Time: 05:08)



And then for each of these partitions we have ANDs, so the ANDs are in this level. Basically we need to check we need to have an one in this half and a one in this part, so we need to have an AND gate and at the top we have an OR gate. So, I suppose the correctness is evident from what I have described above this or before this. So, size how many gates are there? There are n - 1 possible partitions and for each of these, so this is one partition, this is one partition, this is one partition.

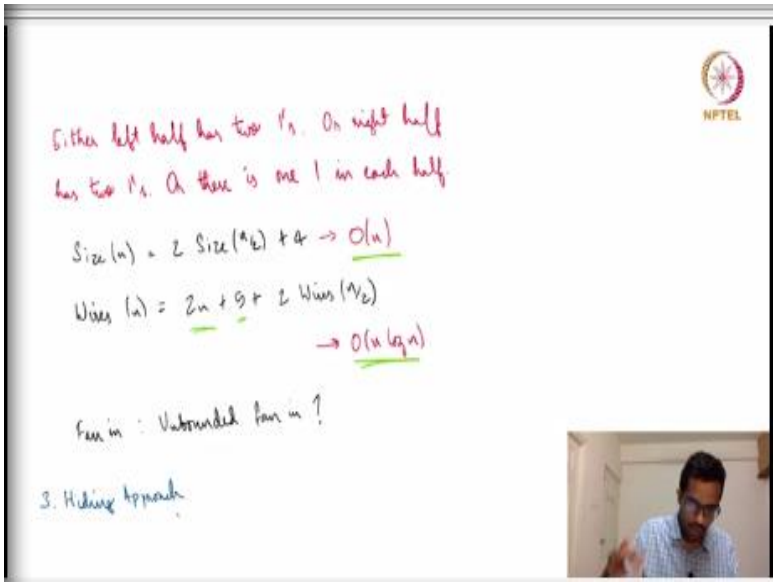
For each of these partitions in general we have 3 gates 2 OR and 1 AND, so that is why it is 3 times n – 1. But and plus 1 top level gate which is the output gate, so 3 times n - 1 plus the output gate, 3 times n - 1 + 1. And I have subtracted 2 here this -2, it is because the left most partition and the right most partition have one gate less, they just need 2 gates one OR and one AND because they have one singleton part each.

So, but anyway this -2 does not really change the asymptotic factor it is still order n, it is 3 n - 1 + 1 – 2, it is still order n which is what we are bothered about. And number of wires, so at the bottom level itself X 1 goes each partition requires n wires each partition and there are n - 1 partitions, so n times n - 1 many wires are required in the bottom level itself to the OR gates. Then to AND gates there are n - 1 partitions and there are 2 wires per part, so 2 times n – 1.

And then at the top level again n - 1 partition is going to an OR gate, so that is n - 1 wire. So, n times n - 1 + 2 times n - 1 + n – 1, again I have put less than or equal to here because the extreme left and right parts may have slightly less. But the key thing is this; n times n - 1 which contributes to an order n squared, so asymptotically it is order n squared.

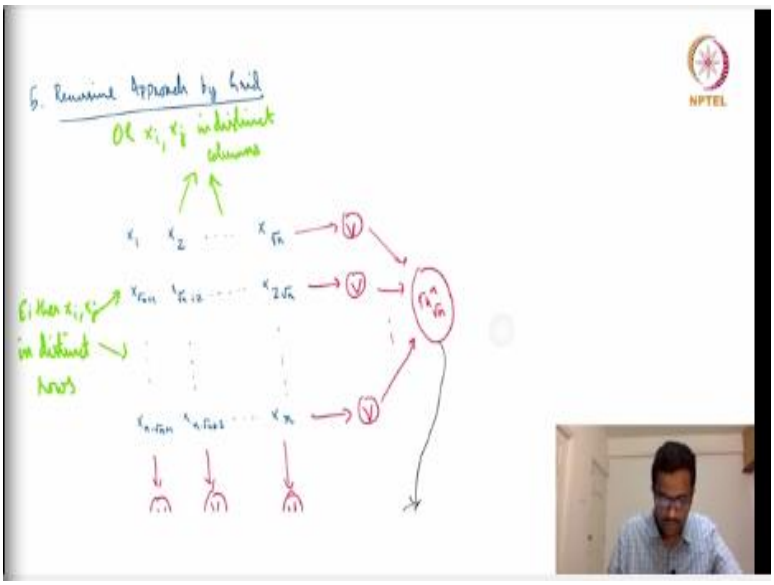
And fanin is unbounded because consider all these that either the top level gate or the bottom gates all the OR gates have take n input or n - 1 inputs, so the fanin is unbounded. So, size is linear are quadratic and fanin is unbounded. So, we saw many approaches already and so far this height the recursive approach seems to be the best.

(Refer Slide Time: 08:03)



Size is order n, wires is order n log n and fanin is unbounded on the first appearance but it seems like it can be made bounded.

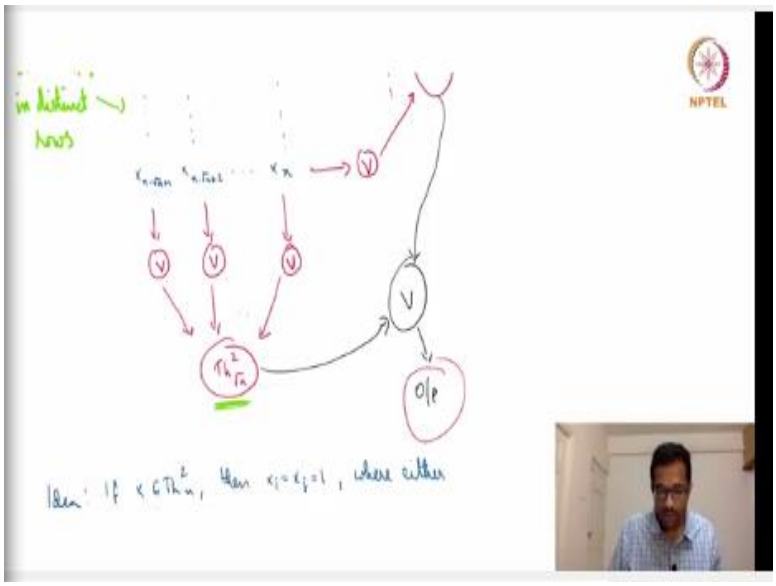
(Refer Slide Time: 08:18)



Moving on the next one is a recursive grid based approach. Again this one needs to assume that n is a perfect square for this 2 to visualize this. If n is not a perfect square we can always pad a bunch of inputs you can go to the next square and then execute this. So, suppose n is 500, 500 is not a perfect square then you can go to 529 which is the next perfect square, so add some few extra bits which we control and make it 0. Because make it false because we need to see there are 2 bits which are 1 in the first part itself.

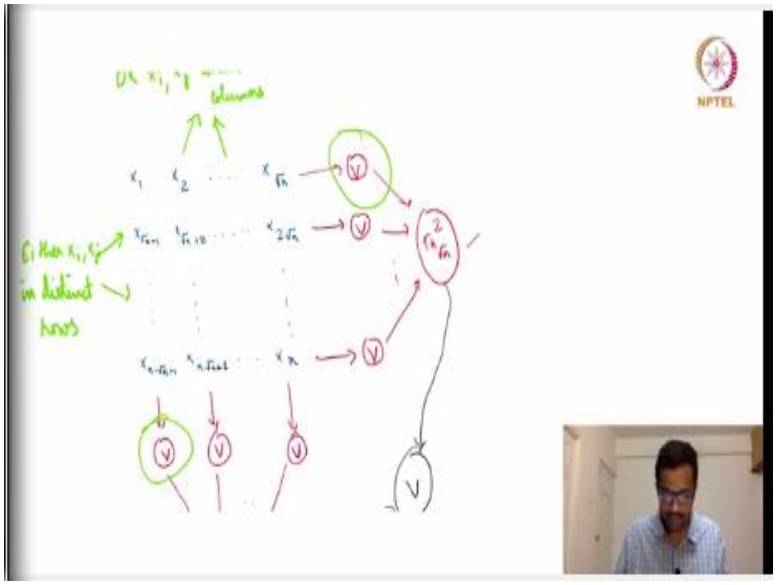
Again what are we trying to do here? If there are 2 bits, 2 positions X i and X j which are both 1, either these X i and X j are in different rows, which means there are different rows or they are in the different columns. They cannot be in the same row and the same column, same row same column means they will be at the same position. Same position means i is fixed or j is fixed, so either they have to be in 2 rows or they have to be in 2 columns. So, either 2 rows, either X i, X j in distinct rows or in distinct columns. And this is what we will formalize in this approach.

(Refer Slide Time: 10:18)



So, either they are in distinct rows or distinct columns. So, suppose they are in distinct columns, or let us say suppose they are distinct columns let us say. That means if you take an OR of all the columns there will be at least 2 columns in which there is a 1, so at least 2 of these column inputs will be true. And then we have a threshold 2 gate or threshold 2 circuit at the with square root and inputs, threshold 2 circuit with square root and inputs to decide that, so this is also potentially recursive approach. And to check whether there are 2 ones in distinct rows again we take OR of all the rows and if there are 2 ones in 2 distinct rows these OR's there will be 2 OR's which will be true.

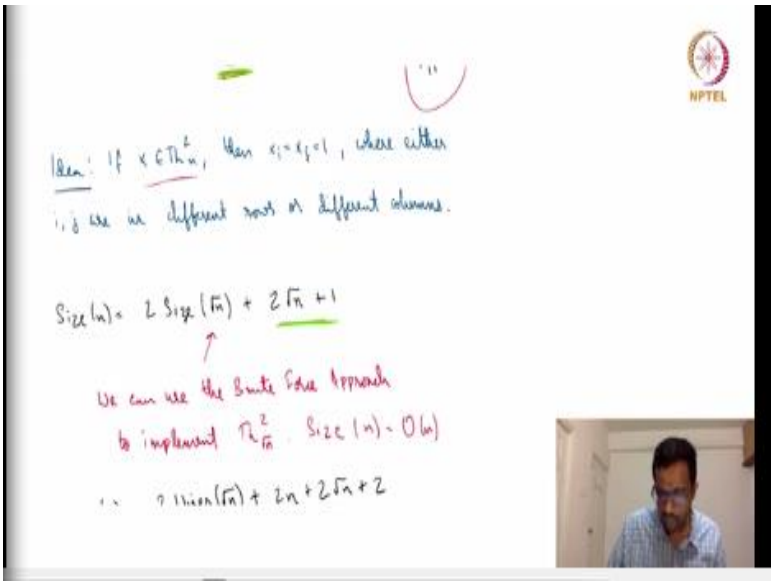
(Refer Slide Time: 11:26)



And then we have a threshold sorry this is not threshold, n is threshold 2 with square root and inputs, threshold 2 circuit with square root of inputs. And either you want the rows, the row threshold to be true or column threshold to be true, so you take an OR of bit both of them and this is the output. So, hopefully the correctness is evident either there are 2 rows in which X i and X j are 1 or 2 columns.

So, if there are 2 rows then this threshold function will fire the one that corresponds to the rows. If they are in 2 distinct columns this green underline threshold function the one corresponding to the columns will fire and finally we have an OR. So, let us see the size wires etcetera.

(Refer Slide Time: 12:14)



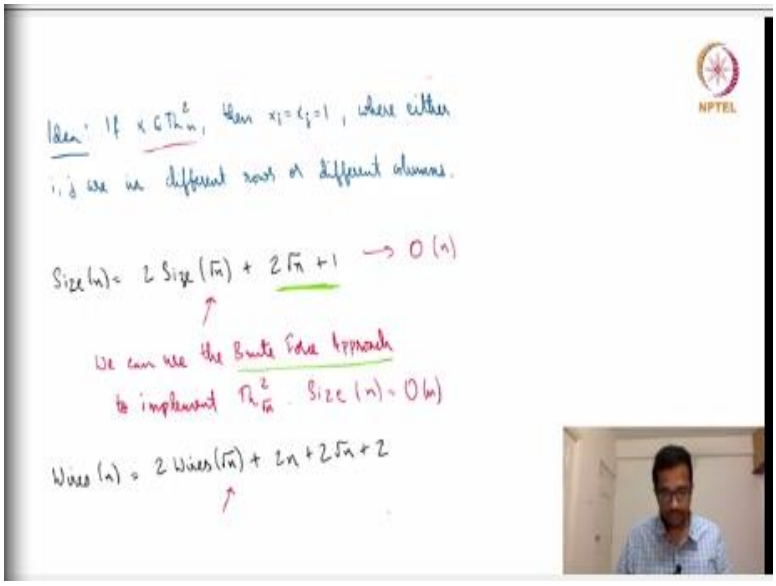
Again I have written the idea here if X is in the threshold then either then X i = X j is 1 where i and j are distinct, so either there are different rows or different columns. So, size, how many gates? We have 1, 2, 3 up to square root AND gates here, square root AND gates corresponding to columns, square root AND gates corresponding to the rows and 1 OR gate, so 2 times square root n + 1 plus whatever is the recursive component that is 2 times size of square root n.

Because this is threshold 2 gates of size square root n, so this is the size. One thing that we can try in fact we can try that as an exercise is to now compute this recursion. So, this is size n = 2 times size square root n + 2 times square root n + 1. So, this recursion can be computed, the master theorem is not immediately applicable here because it is not of that form we have n going

to square root n. But there are some transformation tricks that you can employ and solve this, I am not going to get into that because this is not a master theorem or discrete math or algorithms course but you can work it out.

However to get a decent solution we do not need a recursive approach. So, for instance this threshold 2 with square root and input need not be implemented in the same recursive manner, you do not need the same kind of grid based approach here. We could use the Brute force approach for instance, so recall, the Brute force approach had size n squared and wires also n squared. So, but that is when there are n bit input, so when the inputs are square root n bits then Brute force approach will give us a size n order n approach.

(Refer Slide Time: 14:21)



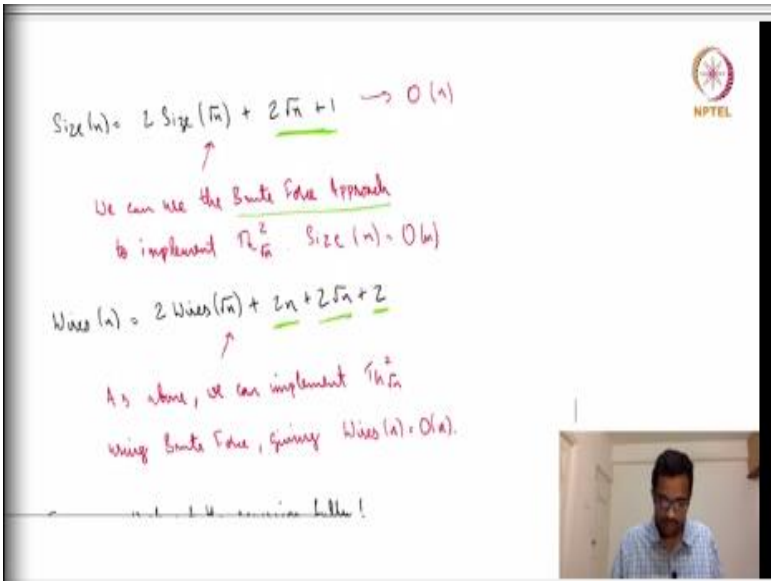
So, which is what I have written here, if we can use the Brute force approach to implement threshold function on square root n input bits, so that will make psi square root n to be order n and overall this function will also be order n. So, this will be order n sorry I will not use the green is not so nice to write, it is good to highlight because I made it thick. So, the size of this circuit will be order n.

Wires, you can see that there are 2 wires going from X 1, one for to the row base OR gate and one going to the I will circle at this OR gate this OR gate go X 1 to this OR gate and X 1 to this OR gate and likewise for any X i. There are 2 outgoing wires, so that this 2 times, so there are n

inputs, so there are 2 times n wires + square root n wires going to each of the threshold function on square root n inputs. So, 2 n I have already explained 2 times square root n for each of the threshold going into the threshold function on square root n inputs and then 2 wires going into the OR gate.

So, this is explained, what remains is the number of wires inside the threshold square root and realization of the construction? And that I am just calling it 2 times wires of square root n. Again one thing that we can do is to compute this in a recursive setting. So, you could implement it recursively where each threshold square root n itself is realized using a similar grid based approach, so again it will be a recursive calculation. As I said before the masters theorem does not lend nicely to this but you can solve it with some simple tricks.

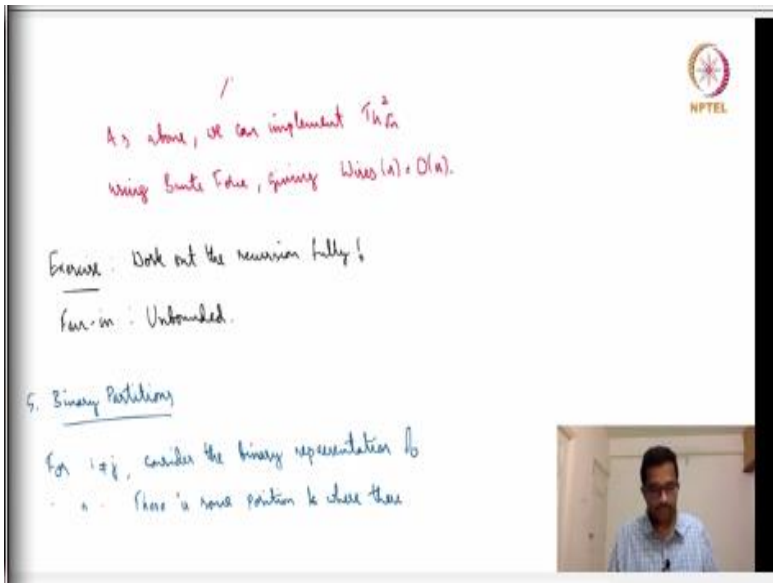
(Refer Slide Time: 16:31)



But however just like in the case of size you do not need to do that because even if you use the Brute force approach you realize the threshold 2 function on square root n inputs. Even then the number of wires will be linear or order n because it is on square root n input, so square of that will be order n. So, overall you will get this also order n even if you use the Brute force approach for implementing the threshold 2 circuit on square root n inputs, you do not need a recursive approach.

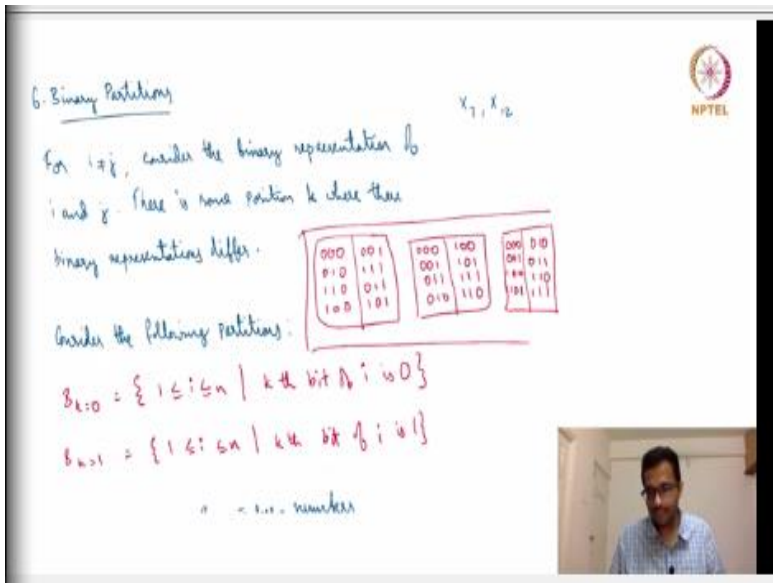
But you can calculate what happens when you try the recursive approach. And fanin again you can check each of these OR gates it takes fanin square root n which varies with the input which is not a constant, so it is unbounded fanin.

(Refer Slide Time: 17:34)



So, I am just write; that fanin and as an exercise you can work out the recursions. So, this is the recursive approach with the grid but again need not be recursive.

(Refer Slide Time: 18:05)



And finally number 6 is binary partitions. Again the idea is same here, we already mentioned this idea in the partition approach that if there are 2 bits which are both 1 then there has to be some partitioning where each part has a 1. So, we are just trying out, so basically most of these

approaches we are trying out different ways to separate these ones into 2 parts, so that and checking whether each part has a 1.

So, a binary partition is such a similar approach. So, suppose X i and X j are the 2 bits with positions which are both 1. Now you can consider the binary representation of X i and X j, so for simplicity let us just let us take X 7 and X 12. Now X 7 is nothing but 0111, X 12 is 1100. So, there are positions where they differ, any 2 numbers there will be some position where they differ, they cannot have the same binary representation if they are 2 different numbers.

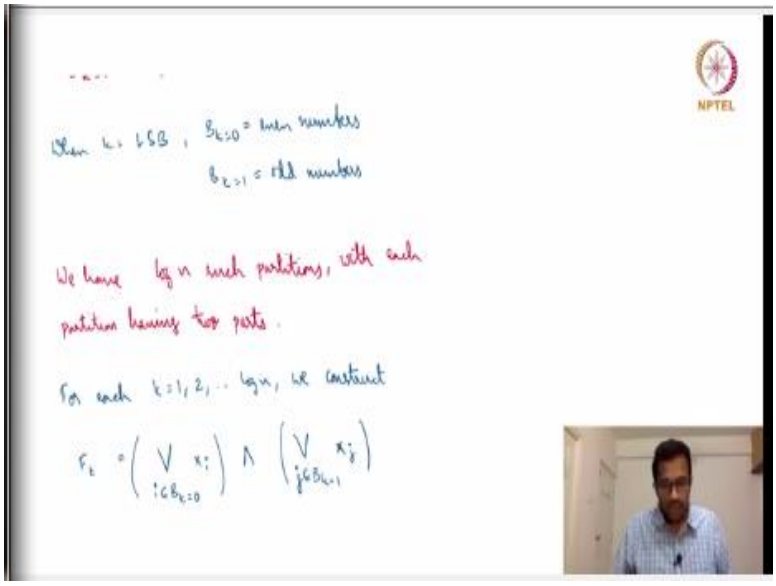
So, here is one position where they differ, let us say this position. So, now there will be some k in which i and j differ in the kth position, where the kth bit wise position. So, now we will consider the following partitions, maybe I will explain the partitions in terms of 3 bit numbers. One is where the for each k where k is a bit position, we are trying to categorize into 2 separate parts. So, one is when k is the least significant bit, so when you have 3 bit inputs when k is the least significant bit k = 0 means the least (()) (20:04) is 0 which means that we are just dealing with even numbers.

When k = 1, we are just least significant bit is 1 which means we are dealing with odd numbers. So, the partitions are so 110, 000, so maybe I do not need to write this down but similarly you can consider the partitions where the middle bit differs and the partitions where the most significant bit differs. So, the partition where the more significant bit differs will be 0, 1, 2, 3 will be in 1 part, 4, 5, 6, 7 will be in the other part.

And partition where the middle bit differs will be I think 01 will be in 1 part, 01 4 and 5 will be in 1 part, 2, 3, 6 and 7 will be in the other part, so these are the 3 partitions. One for each bit position that we get for 3 bit numbers. So, there are 3 ways to partition 3 bit numbers perhaps I will just write it down anyway. So, the first part is 000, 010, 110, 100 this is on one side and 001, 111, 011 and 101 this is first partition.

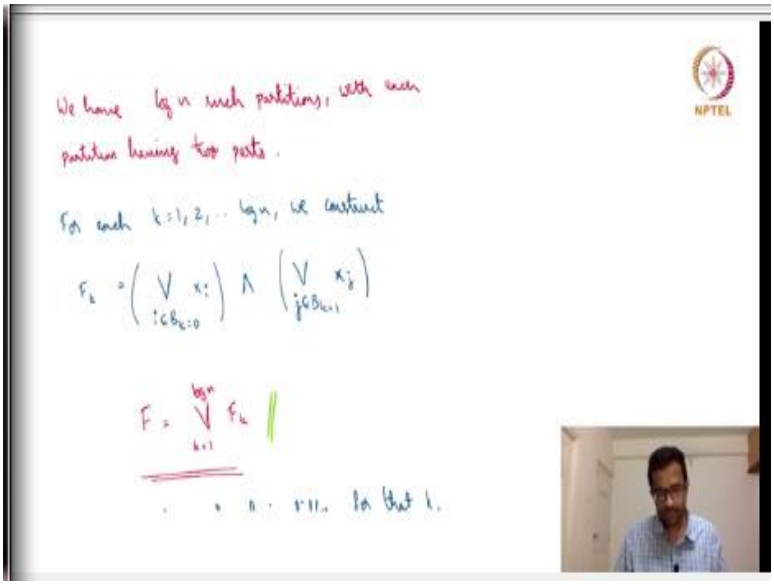
Second partition is 000, 001, 011, 010, 100, 101, 111, 110 and finally the based on the middle bit 000, 001, 100 and 101 and finally 010, 011, 110, 111. So, there are 3 ways to partition the 3 bit numbers and 4 bit numbers you will have 4 ways to do this.

(Refer Slide Time: 22:26)



So, now consider the following partitions for the 3 bit numbers or for when you are dealing with numbers 1 to n there will be log n partitions, because to represent n we require log n bits. So, we will have log n such partitions, so in this case n is 8 because here you have inputs ranging from 0 to 7, there are 8 bit inputs, 8 positions and but it can be realized using 3 bits.

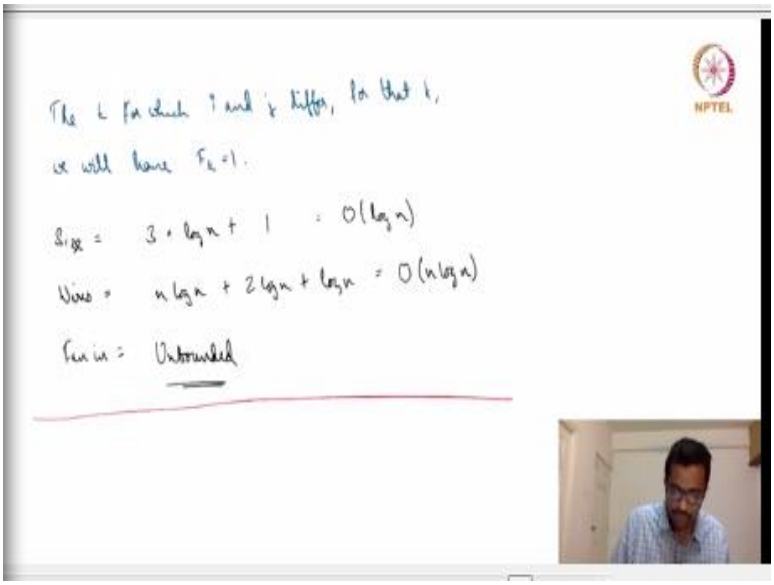
(Refer Slide Time: 23:05)



So, now what we do is if X i and X j are both 1, there is some partition where i and j differ or some position bit position where i and j differ. And now consider that partition, there has to be a 1 in this part as well as this part, so both parts have to have a 1. So, you take an OR here, OR here and an AND of both of them. So, F k is taking an OR in the 0 part and OR in the 1 part at the kth position, so B k 0 is the set of all i which has 0 in the kth position B k 1 is the set of all j which has 1 in the kth position.

And then taken OR of both and then an AND of them together. So, F k will be 1 if and only if there is an i in 1 of the parts and there is a j in the other part. And finally we are just checking is there any partitioning, such that there is i in 1 part, there is a 1 in each part. So, basically we need an OR of all the F k which is what we do here.

(Refer Slide Time: 24:28)



So, whatever k is where i and j differ for that k, F k will be equal to 1. Again the correctness I think I have explained it good enough. And what remains is to compute the size, wires and fanin, let us compute the size. Let us see, so to compute F k how many gates are there? So, 1, 2, 3 there is an OR gate for the one half OR gate for other half and an AND gate. So, computing F k requires 3 but multiplied by number of F k's.

So, how many k's are there? k's how many bit positions we need which is log n? So, 3 times log n plus and OR gate at the top level which is to compute F, so 3 log n + 1. And this is the first

time we are seeing a sub linear sized circuit. Previous everything we saw order n squared, order n I think even order n log n I think, so this is order n, this is order n, this is n, this is n, n squared.

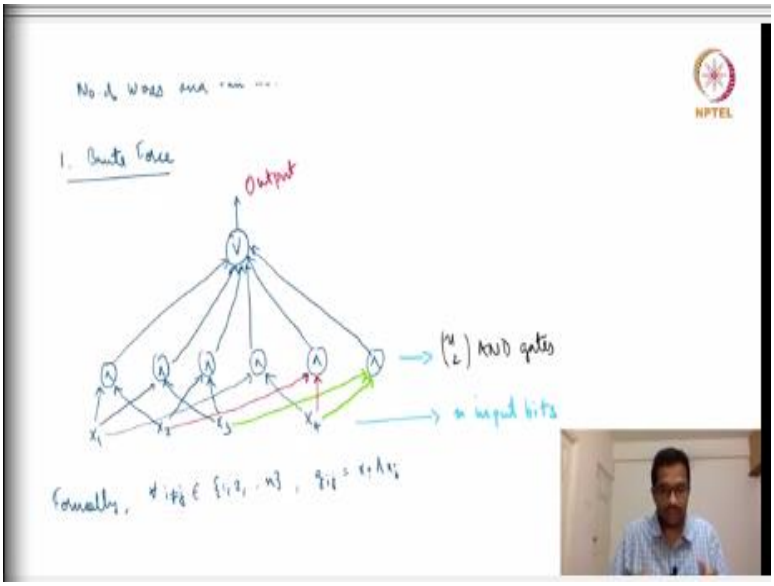
In fact we did not see n log n, we saw n and n square. Wires, so wires if you see each X i contributes to or each X i needs to give input to each part for each k, it may be the 0 part or 1 part. But for each k it needs to give input to one of the parts, so that will give us n times log n because each X i will give 1 input to 1 k, so there are n X i's and there are log n k so n times log n.

And then the number of wires, so this to the AND gate that computes F k we need 2 wires, so which is again 2 times log n plus again log n wires to compute F. But anyway log n is kind of dominated by this n log n term here, so this is again or rather this is order n log n. And finally fanin this should not be that difficult even the bottom level or the top level F requires log n inputs.

So, log n itself is a variable with n, so it is not fixed, so this is unbounded. So, here we have a sub linear sized circuit but the number of wires goes up to order n log n with that I think we are at the end of this lecture. So, we saw 3 more realizations of the threshold 2 function on n bits. And the binary partitions, the recursive grid based approach and the partition based approach. And we saw how each realization does some kind of a trade of some parameters go up, when some parameters go down.

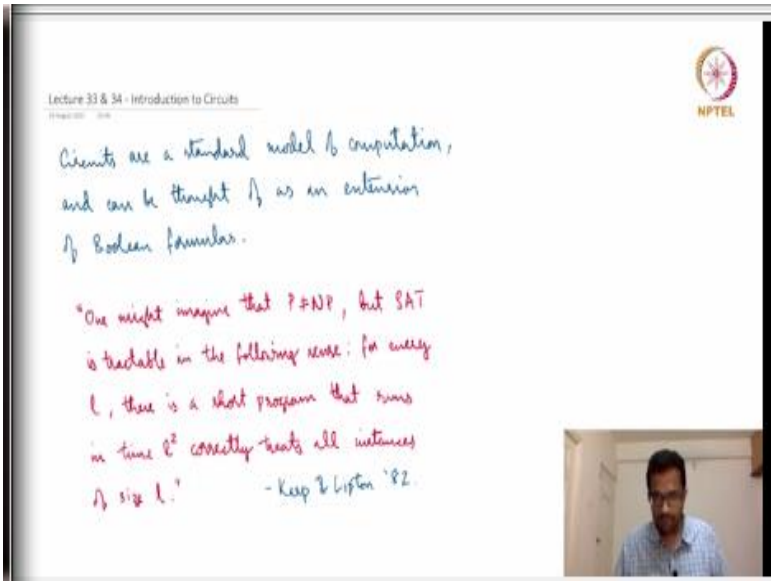
So, there is a kind of balancing that we need to do in each of these cases. There are also other parameters that are of interest in the case of circuits like depth is another parameter.

(Refer Slide Time: 28:29)



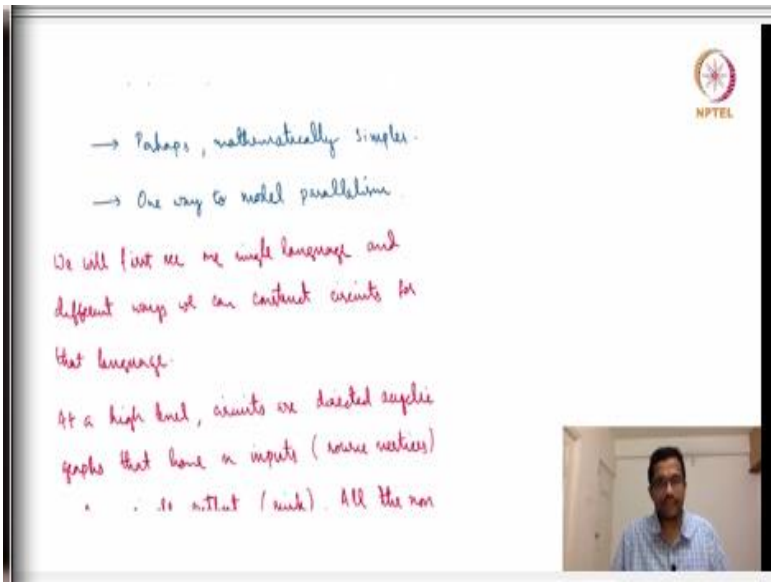
Many case we will hope this gives you an idea of what circuits are and how to compute. You may also have noted how like for a certain number of inputs like we one circuit construction only fits a certain specific input length. If I make a circuit for an n length input I can only feed n length inputs, I cannot use it to feed n + 1 or n + like 2 n sized inputs. So, this is what I meant when I said non-uniform, like I could potentially have the circuit for size n of a certain type and circuit for n + 1 input bits be of a different type. So, I could have a non uniform approach in having different circuits for different input lengths.

(Refer Slide Time: 29:24)



And for more details I think and the formal definition of circuits I gave a very high level definition, we will take it up in the next week where we will see formal definition of the circuits.

(Refer Slide Time: 29:31)



And also see some basic results like which functions have circuits and how much size does do those circuits take? Do all functions do they have circuit realizations etcetera? For instance in the case of turing machines there are languages which are undecidable but what about circuits? Are there undecidable, circuits, languages that do not have circuits? So, we will see all of that in the next week, with that we close week 6, thank you.