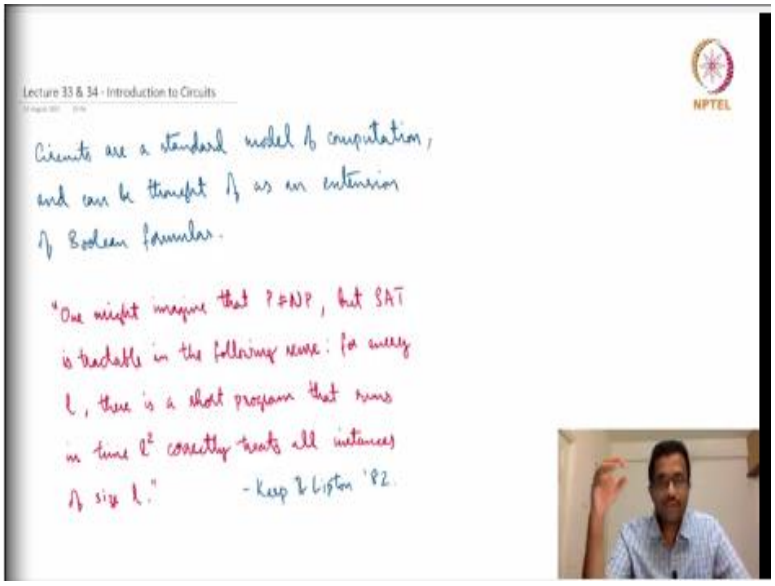
Computation Complexity Theory

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

Lecture-33
Circuit Complexity: Part 1

(Refer Slide Time: 00:15)



Lecture 33 & 34 - Introduction to Circuits

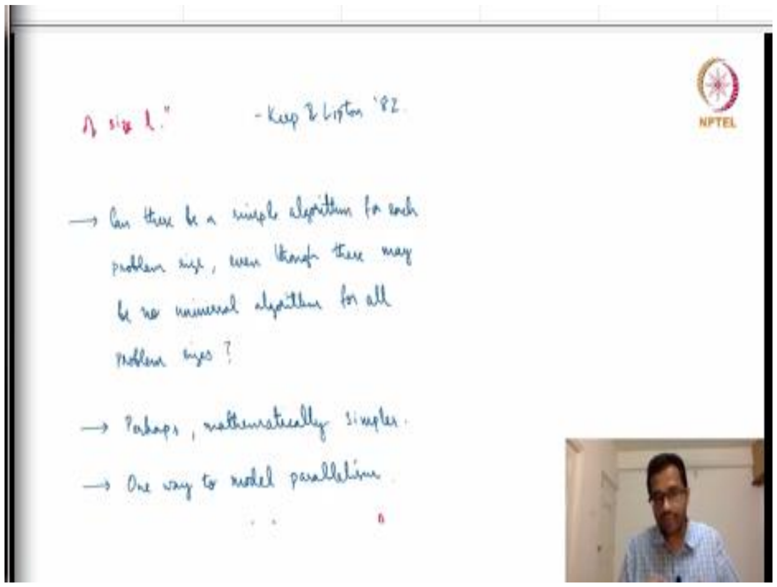
Circuits are a standard model of computation, and can be thought of as an extension of Boolean formulas.

"One might imagine that P ≠ NP, but SAT is tractable in the following sense: for every l, there is a short program that runs in time l^2 correctly treats all instances of size l." - Karp & Lipton '82.

Hello and welcome to lecture 33 of the course on computational complexity. So, far in this course we have seen time bounded complexity classes; space bounded complexity classes and randomized complexity classes. In this lecture we are going to introduce a new model of computation called circuits. And in the upcoming lectures we will see computational complexity classes which are based on the circuit computation model.

So, circuits are rather standard form of computation even though we are seeing it a bit later into this course and it is a very simple very intuitive form of computation. So, it is like Boolean formulas generalized into a computation model.

(Refer Slide Time: 01:05)


of size l." - Karp & Lipton '82.

→ Can there be a simple algorithm for each problem size, even though there may be no universal algorithm for all problem sizes?

→ Perhaps, mathematically simpler.

→ One way to model parallelism.

So, I will first quickly explain very briefly the motivation for the circuit complexity model. Why people began to study it and what were the reasons, what are the motivations? Before explaining the model itself with a few examples and we will not formally define the model in this lecture but we will see a lot of examples. So, in 1982 there was a paper by Richard Karp and Richard Lipton where they made the following quote. They said that "one might imagine that P is not equal to NP but SAT is tractable in the following sense"

For every l, there is a short program that runs in time l squared and correctly treating all the instances of size l. Meaning, it could be the case that SAT is not solvable in polynomial time but given all the instances of a certain size, let us say if for a fixed size instance fixed size formula maybe there is a way to solve it. And this way may not be a general approach, so for a certain l there is a fixed way, for a different l it is a different way.

But and even coming up with the way may not be straight forward. So, because of which you cannot get a polynomial time algorithm that applies to all the problem instances all the problem sizes. So, this is basically a motivation to consider what is called a non-uniform model of computation. Can we have algorithms or ways to decide different size instances differently?

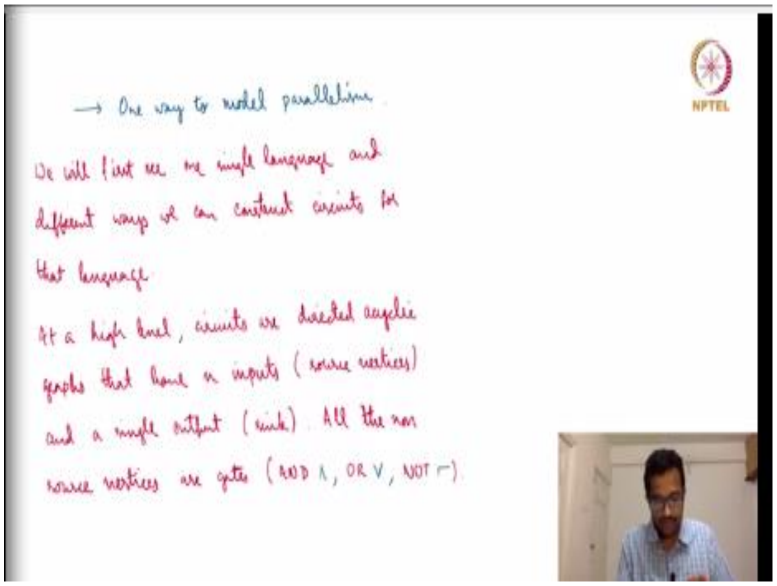
And circuits are a computation model where we can do the non-uniform computation. So, in other words can we have different algorithms for different problem sizes and this is one motivation because of this reason people have studied this was one motivation to study the circuit complexity model rather than the turing machine model because in a way to resolve P versus NP.

People thought that because of this peculiar property maybe circuits are the way to go and people tried various approaches using the circuit computation model. However it was later shown that it is unlikely that even this question is going to be true, the same question that Karp and Lipton are have asked here. It was shown that this is unlikely to be true, because if this was true something else unlikely happens.

However this was a initial motivation to study circuits perhaps this was the way to go and this distinction may help us resolve P versus NP. And turing machine even though we started with turing machine in this course and we consider it equivalent to our everyday computers that we see. However it is still a rather complicated model, so you have a tape that runs around, you have this tape that reads and then moves or left and then erases rewrites.

So, this is a rather involved intricate model, whereas as you will see circuits are very, very simple straightforward model. So, perhaps it may be easier to study a simpler model perhaps this may lead to an easier resolution of P versus NP, this was another hope. And until now lots of people have worked on P versus NP using the circuits but still there has not been much success.

(Refer Slide Time: 05:13)



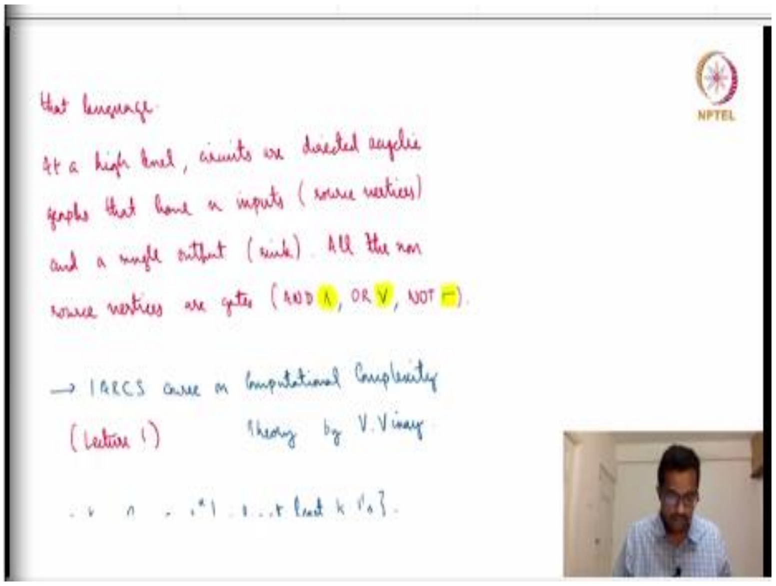
→ One way to model parallelism.

We will first see one single language and different ways we can construct circuits for that language

At a high level, circuits are directed acyclic graphs that have n inputs (source vertices) and a single output (sink). All the non source vertices are gates (AND $\wedge$, OR $\vee$, NOT $\neg$).

Another reason for studying the circuit complexity model is the fact that this captures the notion of parallelism in nice manner. And that is one reason why this may be good to understand, so this is a third reason why circuit complexity circuit computation model has been studied. So, these are the motivations and we will first start as I said with just one language and we will see different ways to realize that language, so this is what we will start with. And that will give you an idea of what circuits are before going to the formal definition.

(Refer Slide Time: 06:04)



that language

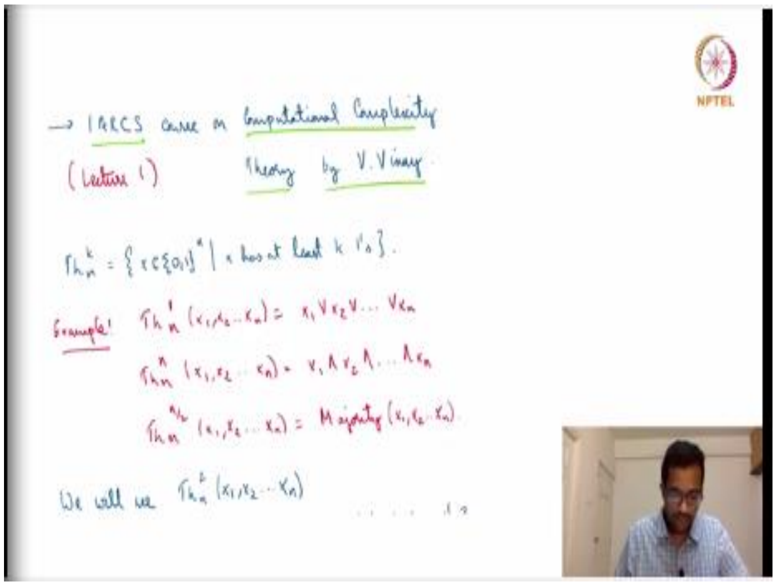
At a high level, circuits are directed acyclic graphs that have n inputs (source vertices) and a single output (sink). All the non source vertices are gates (AND $\wedge$, OR $\vee$, NOT $\neg$).

→ IARCS course on Computational Complexity (Lecture 1) Theory by V. Vinay

[illegible] least k 1's}.

And I will just briefly describe what they are, so that you have some context and we will formally define it in the later lectures. So, circuits are directed acyclic graphs, so they have direction, they have graph without cycles where you will have many inputs, let us say n inputs and a single output. So, in the directed acyclic graphs there will be sources and syncs. Sources are vertices that do not have incoming edges and syncs are vertices that do not have outgoing edges. So, in this we have n sources and 1 sync and all the non-source vertices are in logic gate like AND, OR or NOT. So, we will use this symbol to denote and this symbol the V symbol to denote OR and this symbol denote NOT.

(Refer Slide Time: 07:05)



→ IARCS course on Computational Complexity Theory by V. Vinay

(Lecture 1)

$Th_n^k = \{x \in \{0,1\}^n \mid x \text{ has at least } k \text{ 1's}\}$.

Example: $Th_n^1(x_1, x_2 \ldots x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$

$Th_n^n(x_1, x_2 \ldots x_n) = x_1 \wedge x_2 \wedge \ldots \wedge x_n$

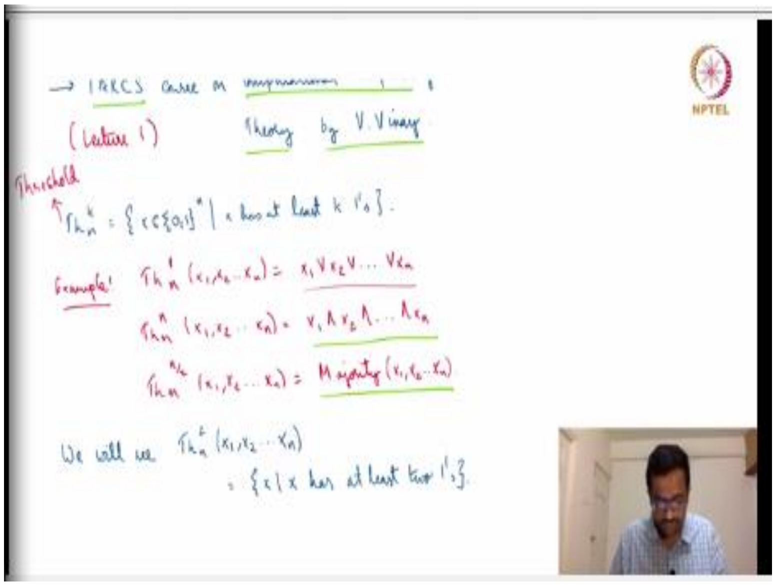
$Th_n^{n/2}(x_1, x_2 \ldots x_n) = \text{Majority}(x_1, x_2 \ldots x_n)$

We will use $Th_n^k(x_1, x_2 \ldots x_n)$

And what I am going to describe is one language and different ways to realize that language using the circuits. And in fact this lecture is most of this material for this particular lecture and the next lecture where I think I will move on to is from if you search online for IARCS and computational complexity theory by professor V. Vinay. Google for IARCS computational complexity theory and V. Vinay.

IARC stands for Indian association for research and computer science, it is an old association. There is a bunch of lecture notes that you will get by professor V. Vinay and this is the first lecture of that course. So, that is one reason and it is a very nice example to illustrate the circuit complexity model. So, again you will find the same things but anyway I will also provide you with these notes.

(Refer Slide Time: 08:23)



→ IRKCS course in [illegible]

(Lecture 1) Theory by V. Vinay

Threshold

$Th_n^k = \{x \in \{0,1\}^n \mid x \text{ has at least } k \text{ 1's}\}$.

Example: $Th_n^1(x_1,x_2 \ldots x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$

$Th_n^n(x_1,x_2 \ldots x_n) = x_1 \wedge x_2 \wedge \ldots \wedge x_n$

$Th_n^{n/2}(x_1,x_2 \ldots x_n) = \text{Majority}(x_1,x_2 \ldots x_n)$

We will use $Th_n^2(x_1,x_2 \ldots x_n)$

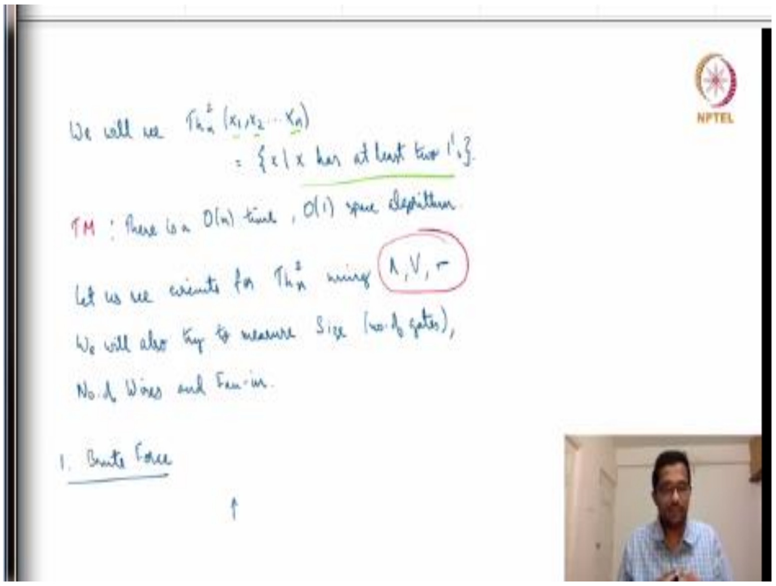
$= \{x \mid x \text{ has at least two 1's}\}$.

So, what is the language that we are going to consider? The language that we are going to consider in fact it is a family of languages called TH subscript n superscript k, so TH stands for threshold. So, this is the set of all n bit strings n with 01 and n bit strings that have at least k ones. So, this is set of all n bit strings that have at least k ones, so just to understand this we will see some simple examples when k is 1, this is the set of all n bit strings that have at least 1, 1.

In other words it is just the OR function. And when basically at least one of the X 1, X 2, X 3, up to X n at least one of them has to be 1, so this will be 1. If this is will be in the language, so if at least one of them is a 1. Threshold n n means that of the n n bits this has to be at least n of them has to be 1 have to be 1 which means all of them have to be ones which means it is the AND function, all of them have to be ones.

And next is when k is n by 2, so half the bits have to be ones in other words it is what is called the majority function, so at least half the input bits have to be ones. So, when n is odd this is the actual majority when n is even it is it is kind of like the majority, half of them have to be ones.

(Refer Slide Time: 10:09)


We will use $Th_n^2(x_1, x_2 \cdots x_n)$
$= \{x \mid x \text{ has at least two 1's}\}$.

TM : There is a $O(n)$ time, $O(1)$ space algorithm.

Let us use circuits for Th_n^2 using $\wedge, \vee, \neg$

We will also try to measure Size (no. of gates),
No. of Wires and Fan-in.

1. Brute Force

So, this is sometimes called the majority function and what we will see **is** from this threshold family we will see the language threshold in with k = 2. So, X 1, X 2 up to X n are the individual bits, these are the individual bits and the strings in the language are those strings that have at least 2 ones, set of alls X that have at least 2 ones. And maybe you may want to ask how would the turing machine compute this?

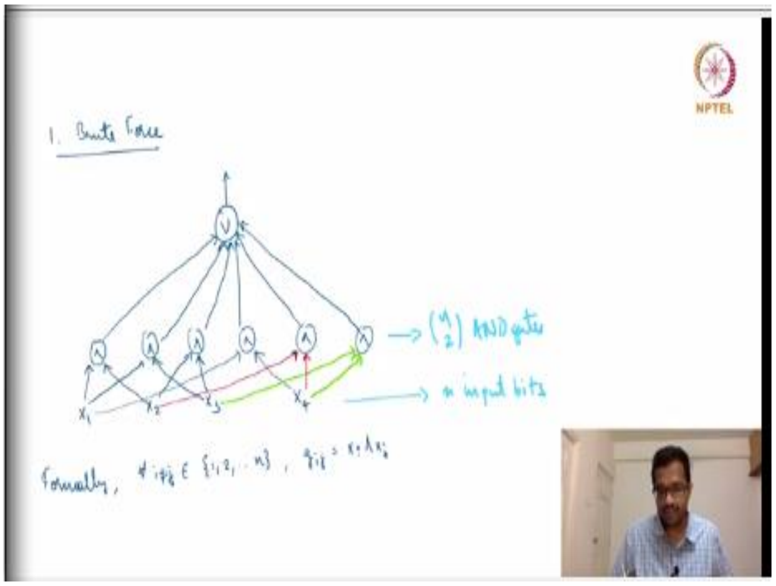
So, you could just write a simple turing machine algorithm that just scans the input from right to left and just looks for 2 ones. And once it sees 2 ones it can just stop, so all it needs to remember is whether it has seen at least 1, 1, so that it can be just one bit flag. And of course it also needs to it does not even need a counter it can just go 1, 1, 1 like one step one cell, one cell to the right till it hits the end of the string, it does not even need a counter.

So, all that it requires is order n time, so if you have a turing machine there is a order n time, this is an n sorry if you are not able to read my handwriting order n time and constant space algorithm, this is a standard thing. Anyway what we are going to see today is not turing machines but circuits. And we will see again as I said before we will see circuits using for threshold n 2 threshold 2 using AND, OR and NOT gates.

So, this is all is written like a V and is written like a wedge it is called a wedge but it kind of looks like an A without the horizontal bar and NOT as this. And some of the interesting parameters, so later we will see complexity classes for the circuit computation model. So, just like we studied time and space of the turing machine model, some of the interesting parameters to measure are size of the circuit.

Size of the circuits are just the number of logic gates that are used in the circuits. The number of wires of the circuit, so number of wires you will see what they are with some examples. And fan-in, again fan-in means what is the maximum number of inputs any gate can take, this is what is called fan-in.

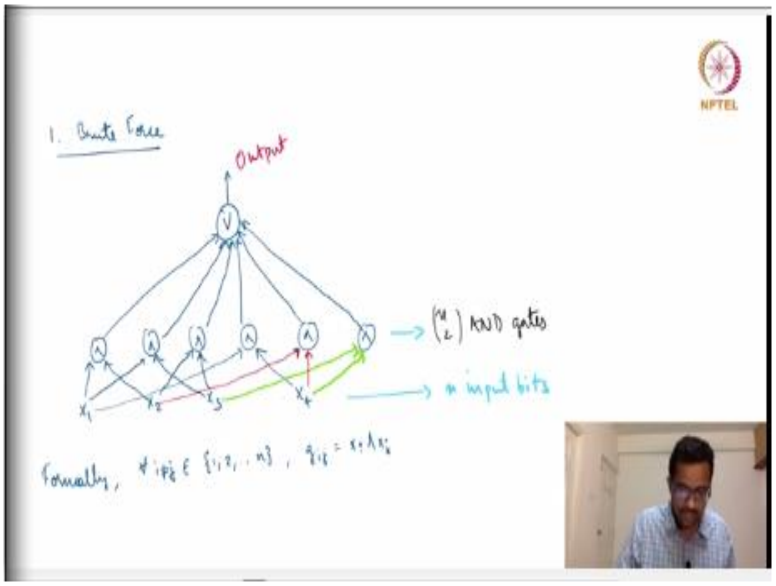
(Refer Slide Time: 13:03)


1. Brute Force
$\rightarrow \binom{n}{2}$ AND gates
$\rightarrow$ n input bits
x_1
x_2
x_3
x_4
Formally, $\forall i \neq j \in \{1,2,\dots n\}$, $g_{ij} = x_i \wedge x_j$
NPTEL

Again size and fan-in you may have heard when you study digital logic in your undergraduate. So, let us try to understand let us try to realize this function threshold 2 using some circuits. So, the first approach is what is the Brute force approach, so we are saying that out of the n bits input there are at least 2 ones. So, why not have a very simple idea? Try all possible pairs of in input bits.

So, like here I have drawn it for 4 bit input, X 1, X 2, X 3, X 4 and basically I have there are 4 input bits, so n input bits. And I have basically n choose 2 AND gates. So, because it is 4, we have 6 AND gates basically all possibilities X 1 X 2, X 1 X 3, X 1 X 4, then X 2 X 3, X 2 X 4 and X 3 X 4, 6 possibilities with the 4 inputs. So, in general it is n choose 2 AND gates maybe this is not very visible I will write with another colour, n choose 2 AND gates.

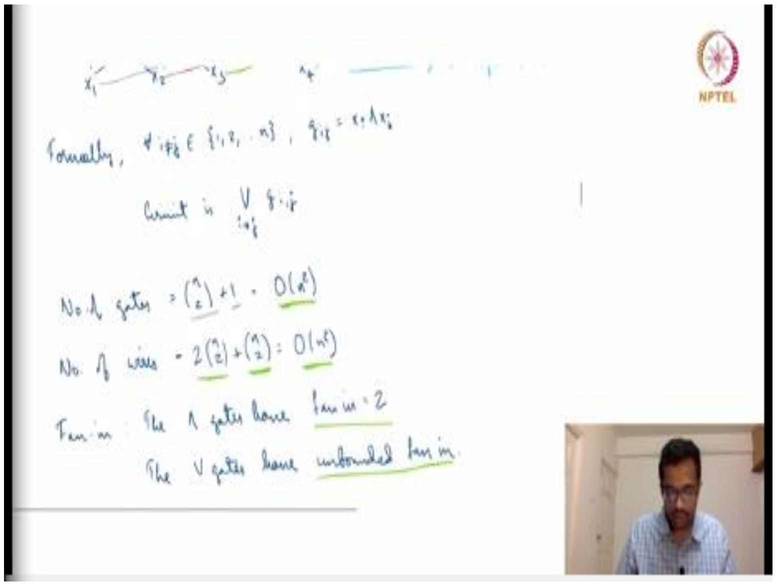
(Refer Slide Time: 14:29)



Formally, $\forall\, i \neq j \in \{1, 2, \ldots, n\}$, $g_{ij} = x_i \wedge x_j$

And if there are 2 ones in the input at least one of these AND gates will give us a 1. If there are 2 ones in the input X i and X j be the input, so there will be an AND gate with where which takes X i and X j as input and that will report a 1. And so we finally have an OR gate to check if is there any AND gate that is reporting a 1 or that is reporting true.

(Refer Slide Time: 15:19)



Formally, $\forall\, i \neq j \in \{1, 2, \ldots, n\}$, $g_{ij} = x_i \wedge x_j$

Circuit is $\bigvee_{i \neq j} g_{ij}$

No. of gates $= \binom{n}{2} + 1 = O(n^2)$

No. of wires $= 2\binom{n}{2} + \binom{n}{2} = O(n^2)$

Fan-in: The $\wedge$ gates have fan in = 2

The $\vee$ gates have unbounded fan in.

So, this is or just a bit more formally we have the AND gates between each pair is called g i j so which is an AND between X i and X j and the circuit is just an OR between all the g i j's. So, suppose there are 2 inputs i and j for which X i and X j for which X i and X j are both 1, then g i j will fire, meaning g i j will report true, which means the OR gate will also report true.

And the STOP gate is output gate and that will try to illustrate the output as well. So, let us see how many gates we have, how many gates do we have? So, we have 6 gates here 6 AND gates and 1 OR gate, so 6 is n choose 2, so we have n choose 2 AND gates and 1 OR gate. Which is so n choose 2 is like - n into n - 1 by 2, so it is order n squared, so number of gates is order n squared.

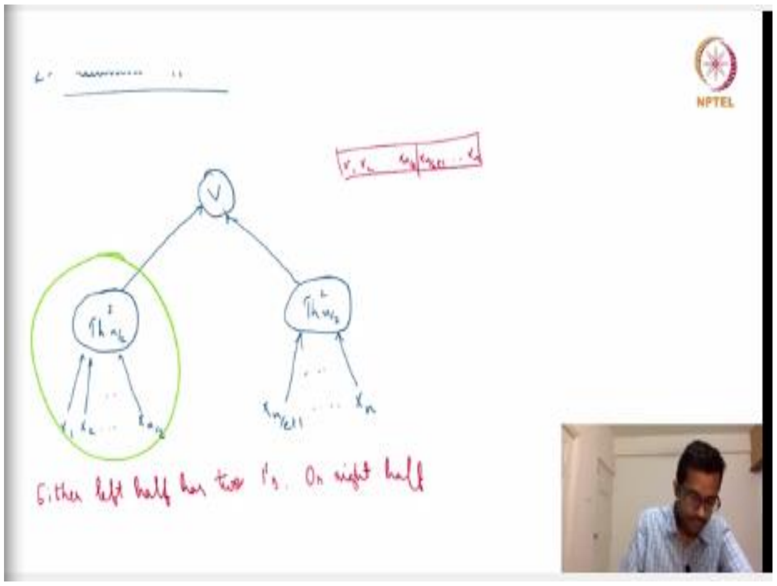
Number of wires, so number of wires is just how many wires are there? Let us say each input go into one of the gate and each gate going to another or each output of a gate going to another gates input. So, let us see each input X i or X 1 goes to 3 gates here, that is because X 1 is involved in it has to be compared with each other. So, or in another way to check is there are n choose 2 gates and each takes 2 input wires, so n choose 2 into 2 is the number of wires to the first level of gates.

And above there is an OR gate that takes again one input and n choose one input from each of these AND gate, so that is n choose 2 wires. So, this is the n choose 2 wires are for the inputs to the OR gate. So, 2 times n choose 2 for the AND gate and n choose 2 for the OR gate, so total is 3 times n choose 2 even that is asymptotically order n squared, so both number of gates as well as number of wires is order n squared.

And fan-in is the maximum number of inputs that any gate has, so here all the AND gates take fanin 2 but the OR gate has fanin n choose 2. So, the AND gates have fanin 2 but the OR gates have n choose 2 and n choose 2 is what we call unbounded. Meaning, the number of inputs or the fanin for that gate is actually dependent on the input itself. Meaning if n is bigger then it will the fanin also will be bigger.

But the AND gates it is always 2, so it is fixed, whereas the OR gate it is unbounded when the input increases the number of inputs to the OR gate increases, so that is what we call unbounded fanin. And in circuits, so as you may have imagined already it is desirable to bring down each of these parameters, the gates, wires, fanin it is better to have smaller of each of this. These are all resources and we want to be economical in using them. So, we saw the Brute force approach which uses order n squared gate, n squared wires and unbounded fanin.

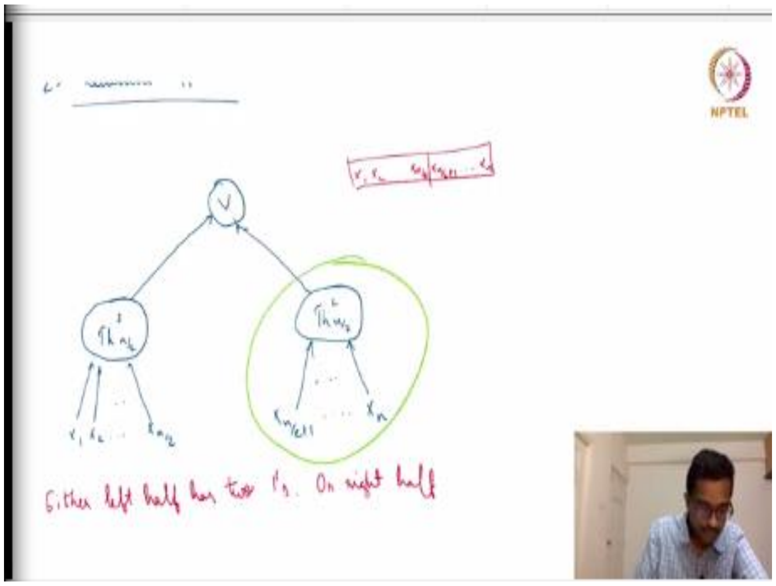
(Refer Slide Time: 18:50)



Now let us consider another approach which we will call the recursive approach. So, what is the recursive approach? So, so we want to check whether there are 2 input bits X i and X j which are both 1. So, now so consider, so X 1, X 2 etcetera X n divided by 2 and then X n divided by 2 + 1 up to X n. So, there are 3 possibilities either there are 2 ones in the first half X 1 to X n by 2 or there are 2 ones in the second half X n by 2 + 1 2 X n or each half has only 1, 1.

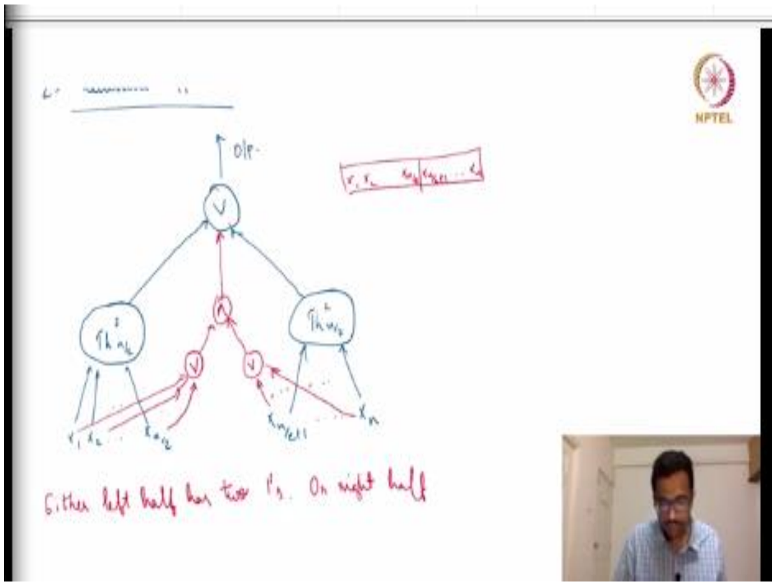
But then each half separately has 1, 1, so the 3 possibilities in by which you could have 2 ones. So, if the first half has a 1 we could use a recursive formula, so that is what we are trying to implement here. So, if the left half, we are trying to implement if basically we are using the same circuit a smaller instance of the same circuit. We are using a threshold realization which takes n by 2 bits.

(Refer Slide Time: 20:21)



And similarly in the right side we are again using a threshold realization which takes n by 2 bits.

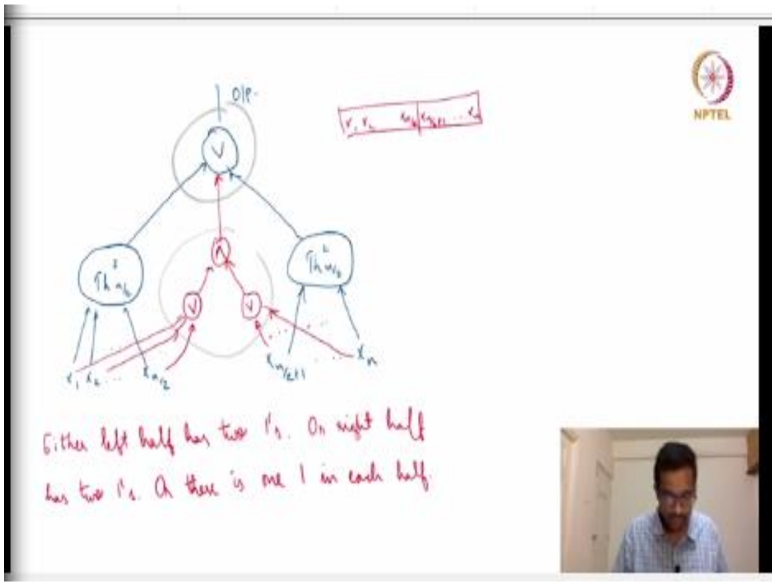
(Refer Slide Time: 20:26)



However there is a 3rd possibility that is there is 1 1 on the left side and 1 1 on the right side. So, in order to realize that we have to basically check is there a 1 in the left side and if there is a 1 in the right side and if both of them are there then we can say ok. So, we have an OR gate here and we have an OR gate here. Basically this OR gate will fire if the first the one on the left side this will fire if there is at least 1, 1 on the left side.

And this OR gate the right side 1 will fire if there is at least 1, 1 on the right side. And if both of them fire we are ok**,** so we just want to check if both of them will fire meaning there is 1, 1 on the left and 1, 1 on the right. So, and this can go as input to the top level OR gate, so there is a recursive implementation for threshold over n by 2 bits and the left and the right and then there is this red part which is the verification of whether there is 1, 1 on the left and 1, 1 on the right. And at the end we are just taking an OR gate and this is the output.

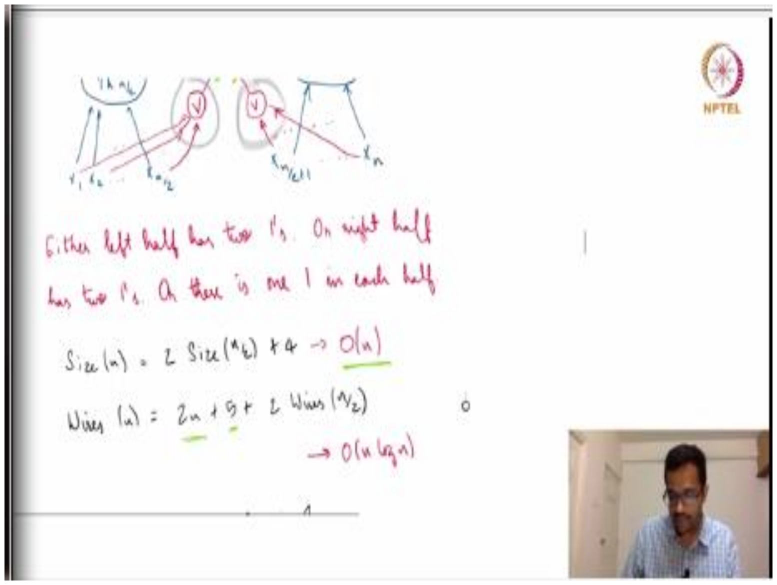
(Refer Slide Time: 22:04)


O/P
$x_1, x_2 \ldots x_{n/2} \mid x_{n/2+1} \ldots x_n$
$\vee$
$\wedge$
$Th^1_{n/2}$
$Th^1_{n/2}$
$\vee$
$\vee$
$x_1\ x_2 \ldots x_{n/2}$
$x_{n/2+1} \ldots x_n$

Either left half has two 1's. Or right half has two 1's. Or there is one 1 in each half.

So, this is the basic idea and what is this number of gates here? So, just like you might have done recursion of recurrence relations in your undergraduate algorithms. Now we need to see, so how many gates we have? We have 1 top level gate this one and we have 3 gates over here the red part that is 4 gates. But then we have these 2 recursive components, this is basically the threshold function itself implemented recursively. So, again we will have similar structure in the threshold part.

(Refer Slide Time: 22:40)



Either left half has two 1's. Or right half has two 1's. Or there is one 1 in each half

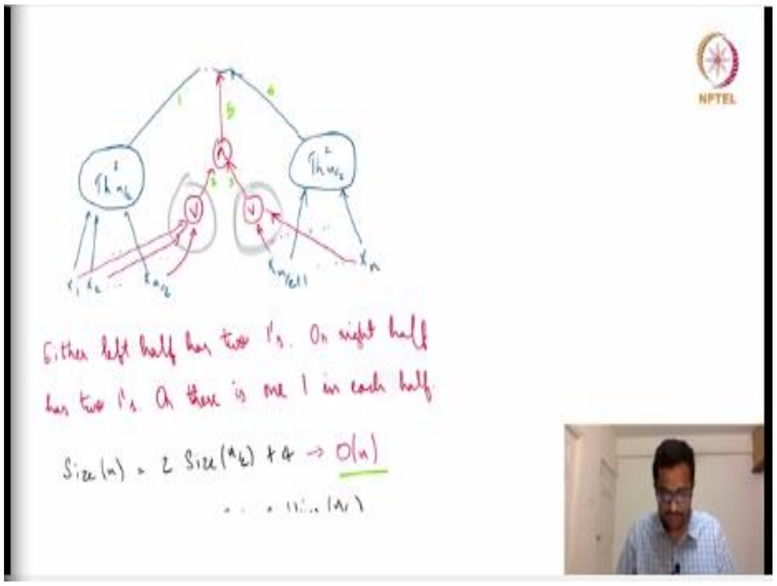
$$Size(n) = 2\,Size(n/2) + 4 \rightarrow \underline{O(n)}$$

$$Wires(n) = \underline{2n} + \underline{5} + 2\,Wires(n/2)$$
$$\rightarrow O(n \lg n)$$

So, basically the size is 2 times the size of n by 2, so this is threshold function for an n by 2 sized input. So, 2 times the size of n by 2 plus the 4 gates that I marked, so I am erasing this plus the 4 gates that I marked. So, 2 times size n by 2 + 4 and this is something that you may have solved in an undergraduate using the standard masters recurrence theorem and this turns out to be a linear.

So, this is linear order n if you solve this recurrence, so this is not very difficult to see you can just try solving it yourself, this may also help you brush up how to solve recurrence relations. The number of wires, so there are n by 2 wires going into the left threshold there are n by 2 wires going into the right threshold, there are n by 2 going into this OR gate the left OR gate there are n by 2 going into the right OR gate.

(Refer Slide Time: 23:34)

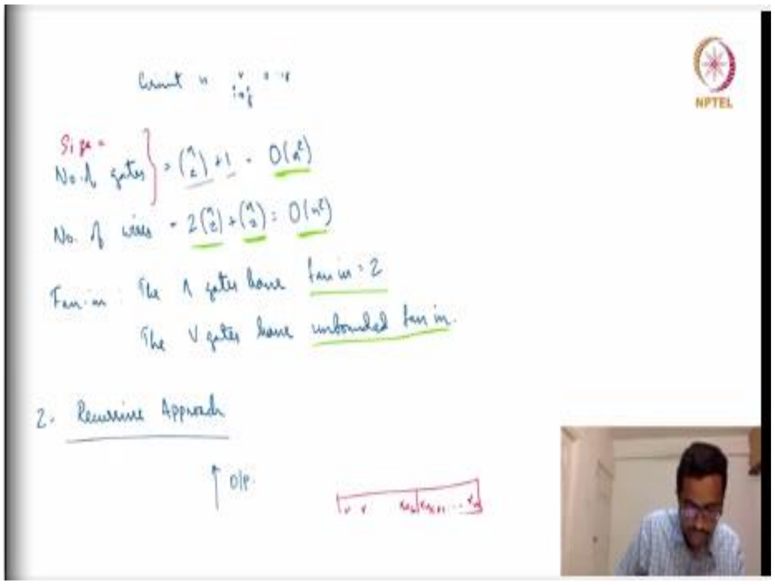

Either left half has two 1's. Or right half has two 1's. Or there is one 1 in each half

$$Size(n) = 2\,Size(n/2) + 4 \rightarrow O(n)$$

So, n divided by 2 times 4, so that leads to 2 n wires over here, then there are 5 wires so which I will mark with green, there is 1, there is 2, there is 3, there is 4, there is 5, there are 5 wires extra, so 2 n + 5. And then there are whatever wires are there inside each of these threshold n by 2 gates. So, 2 times wires of n by 2 which is basically the recursive called.

And if you work that out you will get order n log n. In fact if you work this out you may get n log n. But there may be a bit of over counting because you may be counting the same wire that is going out of this X 1, X 2 inside as well but still I think you will get order n log n. Again this is something that is standard recurrence solving that you can work it out. So, now from n squared and n squared gates and wires.

(Refer Slide Time: 24:58)



Circuit

Size = No. of gates } $= \binom{n}{2} + 1 = O(n^2)$

No. of wires $= 2\binom{n}{2} + \binom{n}{2} = O(n^2)$

Fan-in : The $\wedge$ gates have fan in = 2

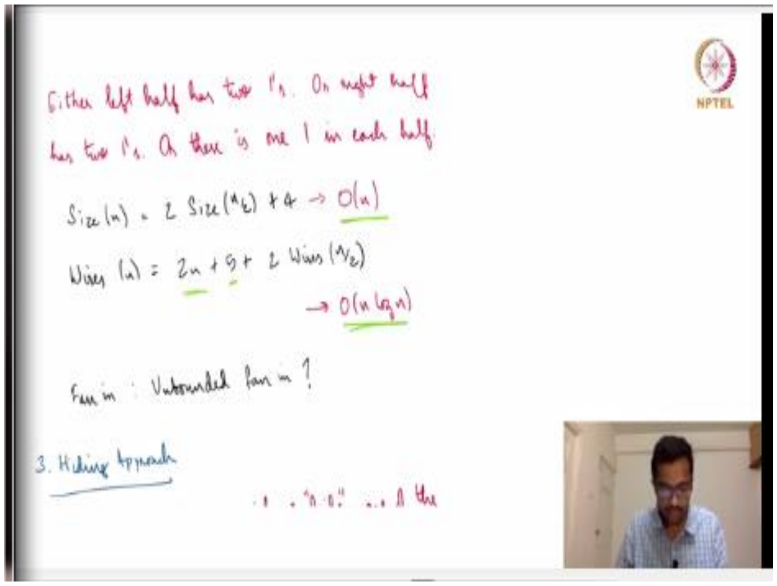
The $\vee$ gates have unbounded fan in.

2. Recursive Approach

$\uparrow$ O/P

So, this one is the number of gates is size n squared and n squared, now we have n n log n.

(Refer Slide Time: 25:05)



Either left half has two 1's. Or right half has two 1's. Or there is one 1 in each half

$Size(n) = 2\,Size(n/2) + 4 \rightarrow O(n)$

$Wires(n) = 2n + 5 + 2\,Wires(n/2)$

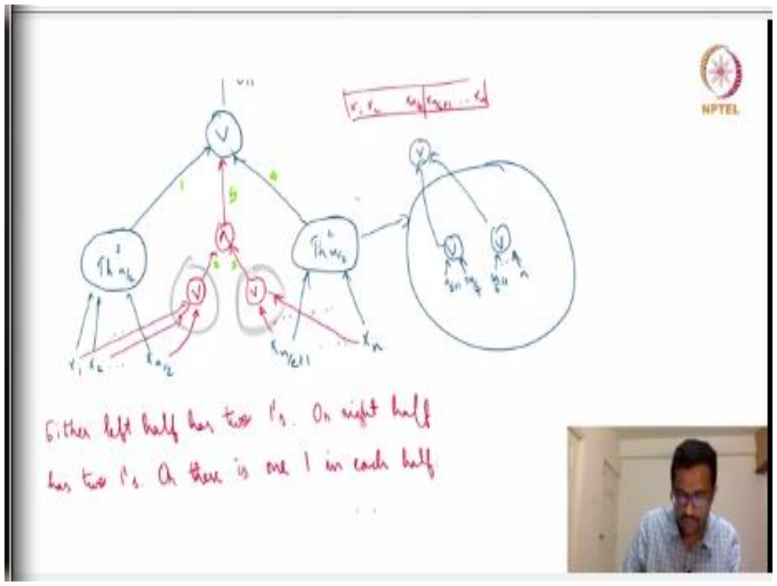
$\rightarrow O(n \log n)$

Fan in : Unbounded fan in ?

3. Hiding Approach

And the first one had a bounded fanin whereas this one also has seems to have unbounded fan-in. But there may be a way to reduce the unbounded fanin, so for instance this red OR gate, the one that I have circled that takes n by 2 fanin. Whereas the other gate the top level OR gate takes 3 then the AND gate takes 2. But however notice that inside these threshold functions inside the recursive calls you will have other OR gates which you may be able to combine to get the OR gate over here. So, you may not have to actually implement an OR gate with n by 2 fanin or at the top level.

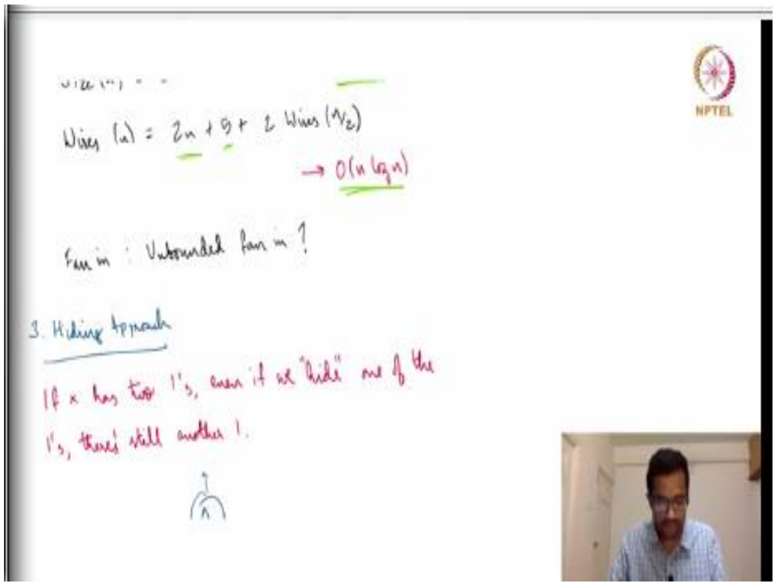
(Refer Slide Time: 26:05)


Either left half has two 1's. Or right half has two 1's. Or there is one 1 in each half

So, what I am saying is if you expand this threshold n by 2 gate, you will again find one OR gate that takes n by 2 + 1 to 3 n by 4, so I am expanding the right side threshold gate. And then another OR gate which has 3 n divided by 4 + 1 to X n, so I omitted the superscripts. So, it is X n by 2 + 1 to X 3 n by 4 and X 3 n by 4 + 1 to X n.

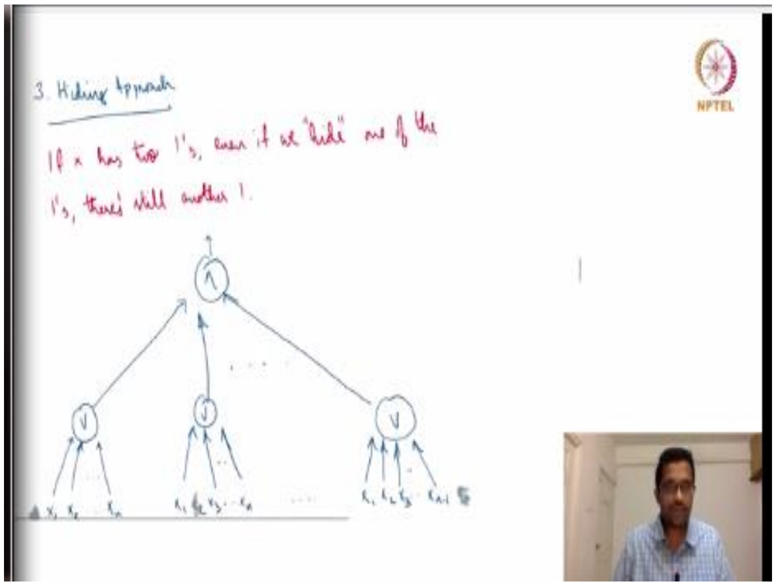
And this would already be implemented in this threshold gate, so all you need to do outside for the or from X n divided by 2 + 1 to X n is to just take the R of these 2. So, and sorry R of these 2, so you could just bring it out and take another R, this is something that you can do, I will just erase is this part. And so you may not need an unbounded fanin.

(Refer Slide Time: 27:12)


Wires (n) = 2n + 5 + 2 Wires (n/2)
→ O(n log n)
Fan in : Unbounded fan in ?
3. Hiding Approach
If x has two 1's, even if we "hide" one of the 1's, there's still another 1.

So, I will just leave a question mark, so it is not really unbounded fanin, if you want you can implement it with a bounded fanin. Bounded fanin meaning fan-in 2 or 3 or so something like that, it will not vary based on the input.

(Refer Slide Time: 27:28)



3. Hiding Approach

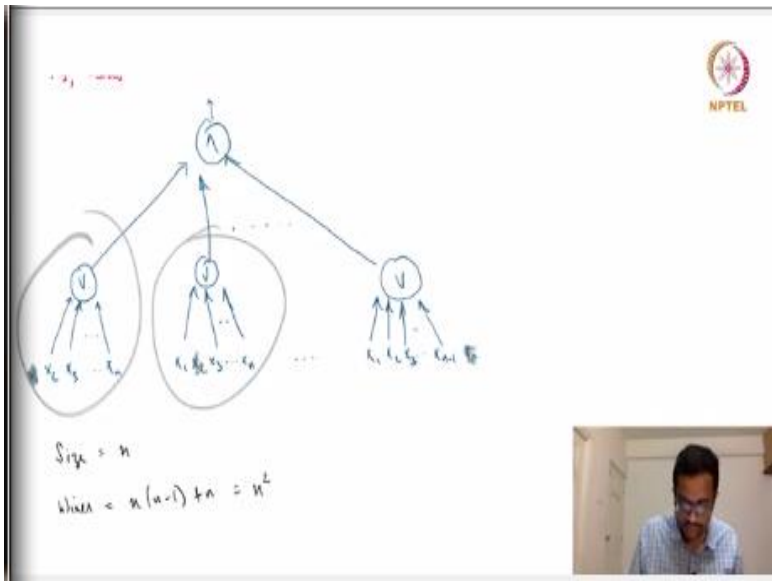
If x has two 1's, even if we "hide" one of the 1's, there's still another 1.

The 3rd approach that I want to consider is what I will call the hiding approach. Again all of these are interesting and clever in their own ways. So, we have 2 approaches seen already one where size and wires were both n squared, the second where size was n and the wires was n log n. Second we said had bounded fan-in, third one is what is called hiding approach. So, again the question is are there 2 X i's which has both one X i = 1, X j = 1 where i 0 = j.

So, if there are 2 such X i's which are both ones then even if we hide one of the X i's and take an R of the remaining bits, it should still give us 1 because there are at least 2 ones. So, even if you hide any input bit the R of the remaining things should give us a 1. And this will happen only when there are 2 input bits equal to 1, if there is only one input bit equal to 1 and then if you hide that the rest of them will not given 1 upon (()) (28:44).

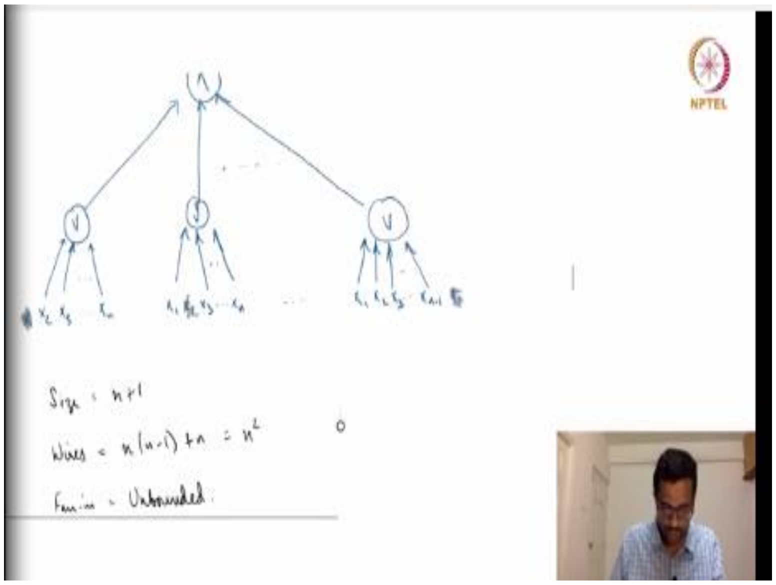
So, what we do in this approach is to simply take all the possible, there are n input bits, so you hide X 1 first then you hide X 2 first, then you hide X 3 first, X 3 and so on till finally you hide X n. And for all of this once you hide X 1 you take the R with the remaining bits then you hide X 2 you take the R with the remaining bits and so on. And then finally you check whether all of these are 1, if all of these are 1, this means that there are 2 ones in the input. So, you have OR’s at the bottom level and then an AND at the top level.

(Refer Slide Time: 29:27)



So, first I have so maybe I will just write this X 1 here which I have hidden, then X 2 I have hidden and then X 3 I have hidden and finally X n is being hidden. So, X 1, X 2, X n, so this merging indicates it being hidden. And we are taking R of all the input bits except, so I am looking at this one, we are taking R of all input bits X at X 1 in the first part and similarly we are taking R of all input bits except X 2 in the second part and so on till the R of all bits except X n in the last part. And what we really want to do after this is to take an AND of all the R’s. So, we have an AND of all the R’s.

(Refer Slide Time: 30:27)



Size = $n+1$

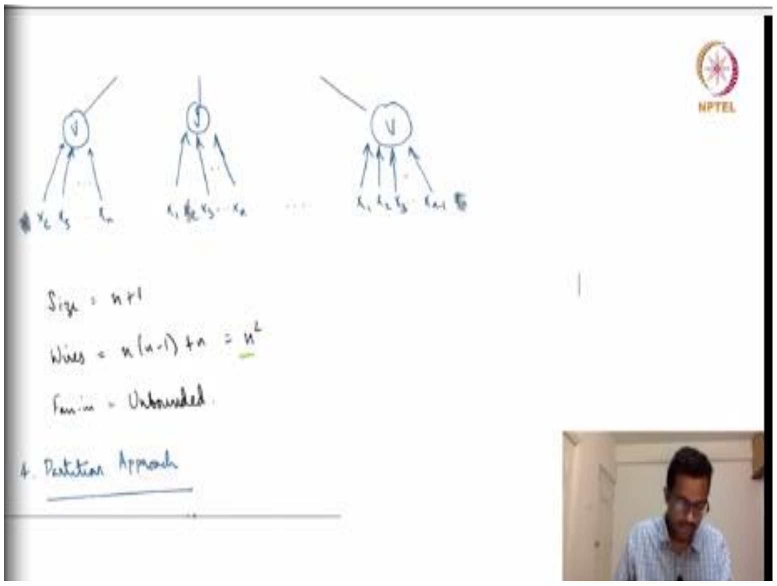
Wires = $n(n-1)+n = n^2$

Fan-in = Unbounded.

So, this is the output, so what is the size here? The size here is I have written n here which is not quite correct, so there are so how many X 1 can be hidden, then X 2 can be hidden, then X 3 can be hidden? So, there are n OR gates at the bottom and +1 AND gate, so it is actually n + 1. The number of sizes is the number of gates there are n OR gates and 1 AND gate. And the number of wires again this is not very difficult to see, each of the OR gates has n - 1 inputs, so n times n - 1 plus the AND gate has n inputs.

So, that is +n, so n times n - 1 + n which is n squared. So, size is n + 1 and wires is n squared, the fanin is unbounded however because both OR gate and AND gate take n - 1 or n inputs which obviously varies with the input size, so fanin is unbounded. Again this is so we saw the recursive approach which took order n size and order n log n number of wires.

(Refer Slide Time: 31:54)


Size = n+1

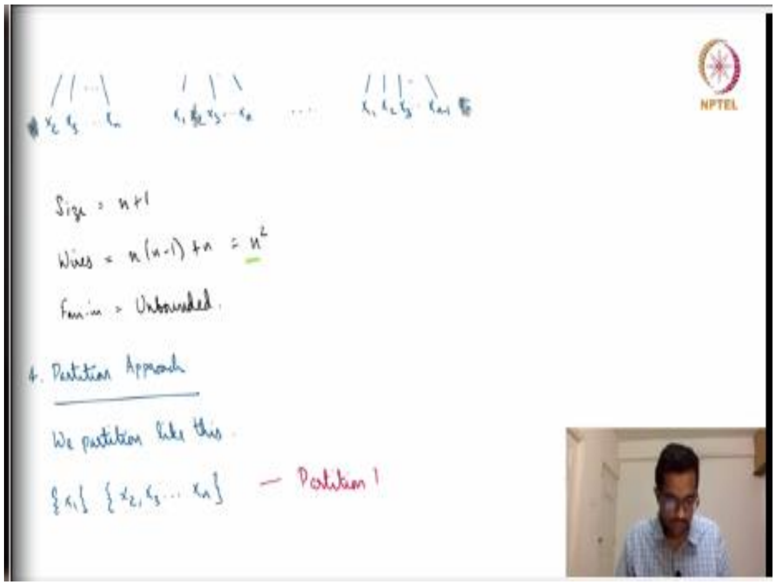
Wires = n(n-1) + n = n^2

Fan-in = Unbounded.

4. Partition Approach

This one takes linear size and n squared wires quadratic number of wires.

(Refer Slide Time: 32:00)


Size = n+1

Wires = n(n-1) + n = n^2

Fan-in = Unbounded.

4. Partition Approach

We partition like this.

$\{x_1\}$ $\{x_2, x_3 \ldots x_n\}$ — Partition 1

So, again this is not that but it is a simpler construction. So, I believe I have 3 more approaches which I think I will cover in the next lecture. So, just to summarize this lecture I introduced what is the circuit model of computation and the motivations explained. Then explain what is the threshold function, threshold k function and what we will see, we saw in this lecture of threshold 2 function.

And we saw 3 approaches the Brute force approach, the recursive approach and the hiding approach and we saw the size number of wires and the fanin of each of these approaches. And just to reiterate these the parameters like size, wires, fanin are standard measures of measuring the circuit complexity. So, just like we saw time, space etcetera in the case of turing machine model, these are the parameters that are considered for the circuit complexity model.