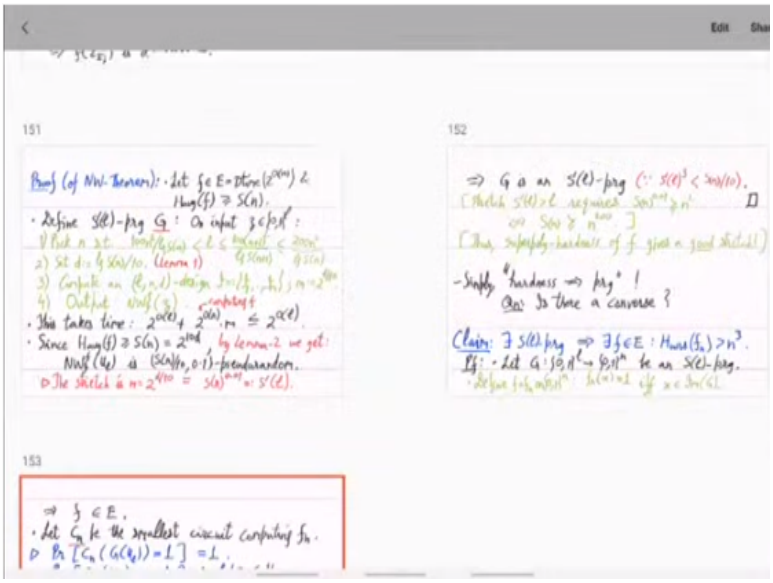
Randomized Methods in Complexity
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 20
Partial Derandomization

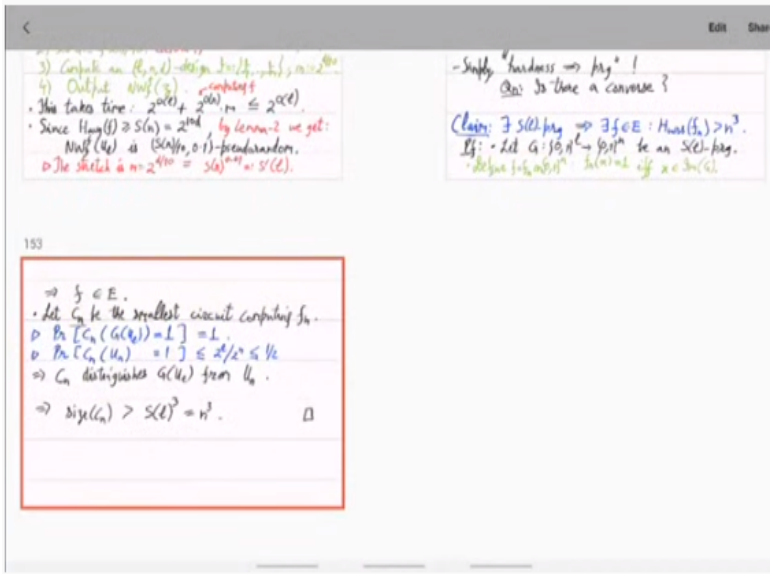
So, in the last class we finished the proof of this very important theorem. And in a way of surprising theorem, it was long conjectured in cryptography and finally proved by Nisan Wigderson in 1988.

(Refer Slide Time: 00:34)



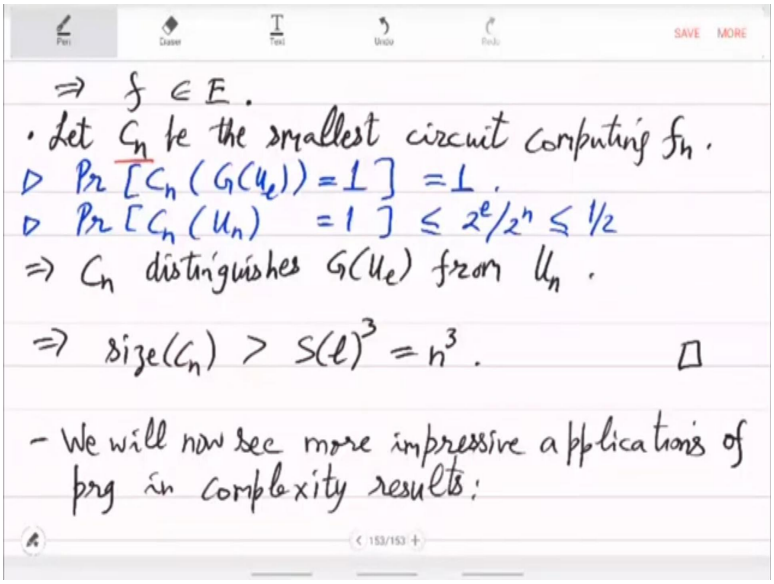
So, the theorem showed that, if you have E explicit Boolean function that is average case hard then, accordingly you will also have a prg, with a non trivial stretch. So, hardness would allow you to stretch strings, so that they look random to limited resource computation.

(Refer Slide Time: 00:55)



And then we showed a very minor lower bound, assuming the existence of a prg.

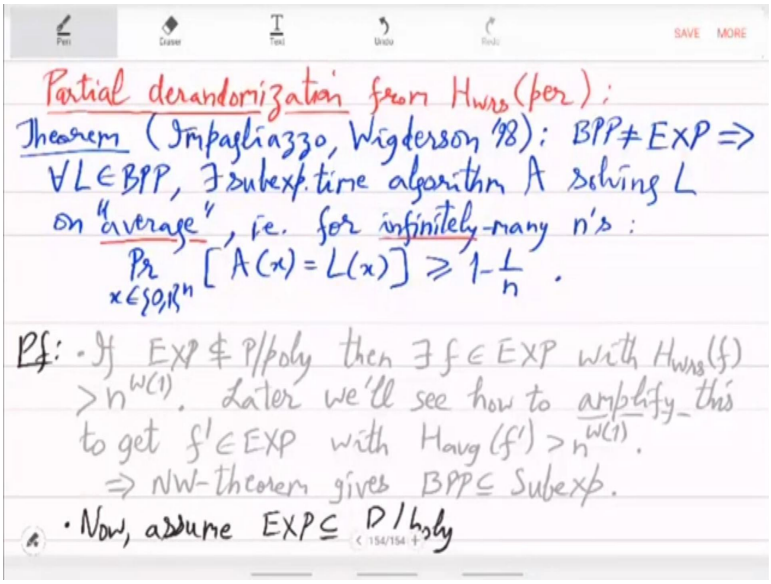
(Refer Slide Time: 01:02)



So, prg implies lower bound, lower bound implies prg. What we will do next is, we will see more applications of this concept of prg in complexity theory. In particular, at the end of this topic or this section, you will see the complete proof of the first theorem that we started with in the course which was if PIT is in P, identity testing, if it is in P then it implies either Boolean circuit lower bounds or arithmetic circuit lower bounds.

So, there was a Lemma complexity theorem there which we did not complete the proof will complete it now using prg's. So, we will now see more impressive applications of prg in complexity theory. So, first is using the hardness of permanent you can derandomize BPP.

(Refer Slide Time: 02:42)



So, first is partial derandomization from H_{wrs} (per) But theorem statement will look more impressive than this. So, actually in the theorem what we will say is, it is a

Theorem: (Impagliazzo and Wigderson from 1998). If BPP≠ EXP which is a very reasonable hypothesis, everybody conjectures this. That using randomized polynomial time algorithms you cannot solve all the problems that are computable in exponential time.

So, if this statement is true, notice that this statement does not have any circuits in it. So, if this statement is true, this already will imply a partial derandomization. What will be shown is $\forall L \epsilon BPP$, ∃subexp.time algorithm A solving L on "average", will mean that on almost all the inputs A can solve L and it is sub exponential time deterministic algorithm.

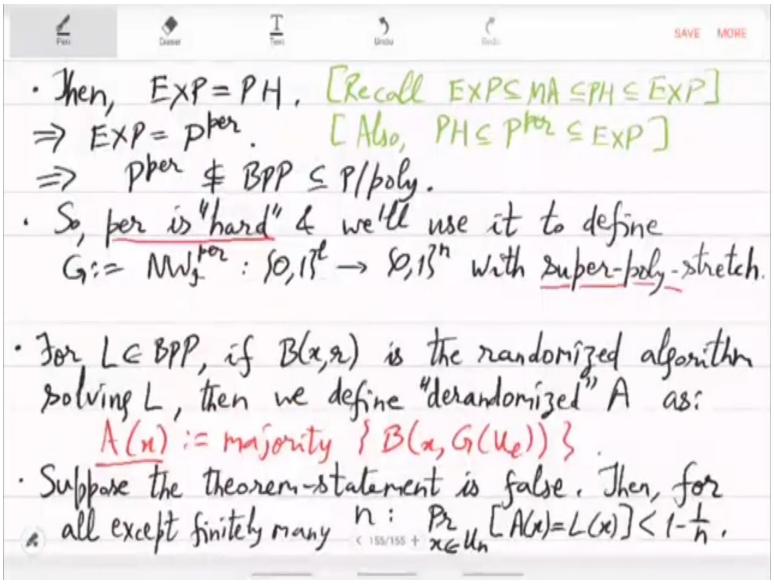
So, this is a partial derandomization of BPP and mathematically we will define this average as, for infinitely many n's .$Pr_{x\epsilon\{0,1\}^n}[A(x) = L(x)] \geq 1 - 1/n$ So, there are 2 things one is that, it may not work for all n but it will work for infinitely many n's, second is even when it works for a n, it works for a large majority of the inputs in$\{0,1\}^n$, it will not work for all $\{0,1\}^n$ strings.

So, let us prove this. So, here we will break into 2 cases. First is whether EXP $\not\subset$ P/poly So, when it is not in p / poly then that is kind of the easier case. Then, $\exists f \in$ EXP with $H_{wrs}(f)>n^{w(1)}$ large. So, if EXP is not in p / poly that mean that, there is a problem which is solvable in exponential time but does not have polynomial size circuits.

, this little w(1) means function that is growing function, diverges to infinity. So, later we will see how to amplify this to get $f'\epsilon$EXP $with\ H_{avg}(f') > n^{w(1)}$. This amplification procedure we will see in the next chapter. We will see it in great detail; it will be using error correcting codes.

So, let us assume all that for now and finish this part of the proof. So, you have from f you have f' and then from f' you have a prg. So, Nisan Wigderson theorem gives then BPP in super poly hardness, average case means, sub exponential time derandomization, so that is done. That was kind of the easy case when EXP is not in p / poly itself that gives you hardness and hardness gives you derandomization. Now, what happens when EXP is in p / poly then there is no hardness, what do you do then?

(Refer Slide Time: 09:31)



So then, EXP = PH. So, recall that we had shown when we were doing initial lectures on identity testing that there were these sequence of results, EXP in p / poly meant that,

$EXP \subseteq MA \subseteq PA \subseteq EXP$we had the sequence. And from this sequence it means that, these 3 classes are equal. So, you have EXP = MA = PH.

So, recall that, we immediately deduce that EXP is equals, actually polynomial hierarchy and moreover, $PH \subseteq P^{per} \subseteq EXP$, this also you recall which is again in EXP. So, this means that, actually all these 4 classes are equal, so EXP in particular is P^{per}. So this, by the way is called Toda's theorem, polynomial hierarchy in P^{per} and permanent, obviously is solvable in exponential time. So, you get all these 4 classes equal, in particular focus on EXP = P^{per}.

Now, you have assumed that BPP $\not\subset$ EXP in the hypothesis. So, which means that,$P^{per}\not\subset$BPP $\subseteq P/poly$. So, we do have some hardness, in particular if you look at permanent, this does not have poly size circuits, it requires super poly circuits. So, it makes sense to use permanent in Nisan Wigderson map and get a prg. So, permanent is hard and we will use it to define this potential prg map, $G: = NW_I^{per}$ where I is your Nisan Wigderson design, permanent is the hard function, it should stretch $\{0, 1\}l \to \{0, 1\}^n$ In the previous lecture, note we used m but here we will continue to use n, just appropriately rename the design parameters with a super poly stretch. That is the idea. What is not clear is how to use this NW map permanent is hard but it is not really average case hard.

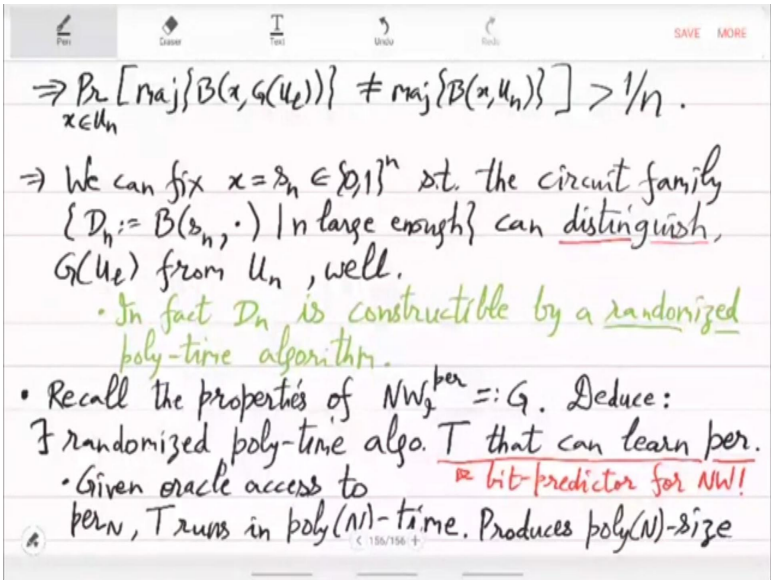
So, we have to see how to go around the conditions required for this Nisan Wigdersons theorem. That is what we will do next. Let us look at it in detail, this implementation. So, for L ϵ BPP, if B (x, r)is the randomized algorithm, solving L then we define a kind of derandomized algorithm using G. So then, we define derandomized A as, so that definition is simply as you can guess, as you have seen before, P will use the output of or the image of G as a pseudo random string.

So, A (x): majority $\{B(x, G(U_l))\}$, guess this is just the majority of all these values of $B(x, G)$ image, over all the image elements of G. So that reduces the random string from requirement of

n to requirement of l. And you can think of this as derandomization because now you can do this in 2^l. So that is the new algorithm A. So, now, suppose the theorem is wrong then, for all except finitely many, what you will get is that $Pr_{x \in U_n}[A(x) = L(x)] < 1 - 1/n$.

So, this algorithm A that we have defined, suppose, this does not satisfy the theorem statement, so which means that only finitely many n's, this inequality holds. And for the rest, infinite minus finite, for those n's, what is happening is that, A and L they are values on x they match with a very low probability. This is the negation of the theorem statement. So, for all except finitely many n's, $Pr_{x \in U_n}[A(x) = L(x)] < 1 - 1/n$ that is the negation. So, what will this give you?

(Refer Slide Time: 17:42)



So, this implies that $Pr_{x \in U_n}[maj\{B, (x, G(U_l))\} \neq maj\{B(x, U_n)\}]$. So, $maj\{B, (x, G(U_l))\}$ that is, A on x and the other thing $maj\{B(x, U_n)\}$ this is really L of x whether x is in L or not that is really this majority when you look at n bit random strings being used by B then the majority answer is of course, correct by assumption because L is in BPP.

So,$Pr_{x \in U_n}[maj\{B, (x, G(U_l))\} \neq maj\{B(x, U_n)\}] > 1/n$ So, you can see that B is kind of discriminator, it is able to distinguish between the image of G and the uniform distribution. So,

you can already feel that, this should lead to some kind of a contradiction to prg definition or NW definition. So, we can fix $x = s_n \epsilon \{0,1\}^n$ to be these strings, one of these strings where B is a distinguisher or discriminator, such that the circuit family $\{D_n := B(s_n, \ . \ )|$n large enough$\}$

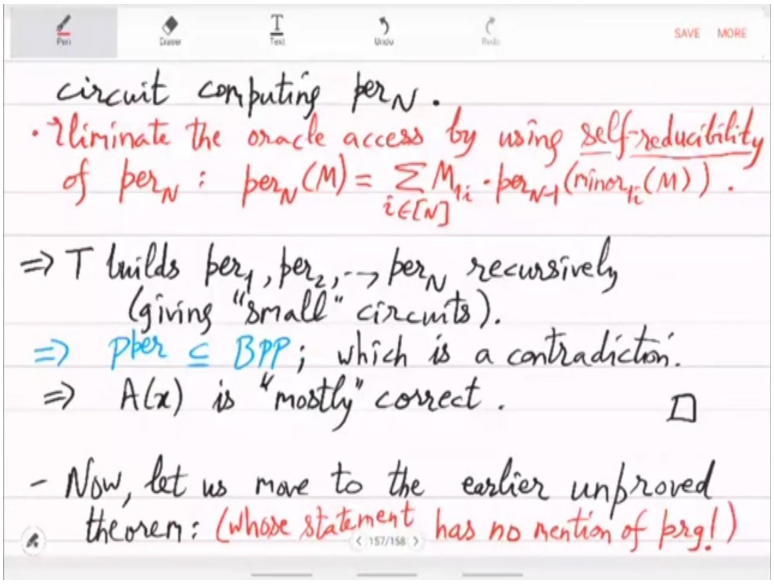
And the second argument we keep free, this can be G image or this can be U_n. So, this circuit, so this can distinguish $G(U_l)$ from U_n very well, D_n are these algorithms and hence circuits, you can also see them as circuits, they are able to distinguish the image of G from the uniform distribution. In fact,D_n is constructible by a randomized poly time algorithm that is because D_n is essentially B which was a given algorithm, explicit algorithm.

You only have to fix s_n string but you can see above that, this string you can just randomly pick because the probability of being a distinguisher is pretty high. It is more than 1 / n. So, you can easily pick it. So, D_n is a easy circuit also explicitly constructible. So, next question is what can you do with this distinguisher? So, for this you go back to the NW map properties we saw last time.

So, recall the properties of $NW_I^{per} = G$ So, deduce that, there exists a randomized poly time algorithm T that can learn permanent. So, this is essentially the idea of bit predictor that we saw, while analyzing the NW map. Any algorithm like, in this case, D_n any circuit family that is able to distinguish NW from the uniform distribution will actually give a bit predictor and the bit predictor in this case would mean that it will give you permanent computation.

So, this algorithm we are calling T, this you get from the bit prediction basically. So, you have learnt permanent and which means that, given oracle access to permanent, T runs let us say permanent on N by N matrix, N by N matrix, so, T runs in polynomial in that much time and produces a circuit.

(Refer Slide Time: 24:55)



Size circuit computing permanent: So this learner for the permanent is there, but it requires oracle access to permanent. It basically requires it to compute permanent on smaller instances. So, how will you get the oracle or how do you eliminate the oracle? You eliminate it by using the identity that permanent satisfies. So, Eliminate the oracle access by using the self reducibility of

$$per_N : per_N(m) = \sum_{i \in [N]} M_{1i} \cdot per_{N-1}(minor_{1i}(M))$$

So that is how it does not matter whether you have the oracle or not because once you have this algorithm T that reduces permanent to smaller permanent, it will be enough. So, basically you can build now, T can build using these circuits for permanent N - 1, T can build a circuit for permanent N and the size bound will be small because T is after all a randomized polynomial time algorithm.

So, the size bound cannot be bigger than the time complexity, so, T builds $per_1, per_2, \ldots, per_N$ recursively, giving "small" circuits because of its time complexity that we know. So, this is how you will learn permanent, this actually goes into the depths of this NW map and the bit prediction that we learnt in the last class. So, using this distinguisher D_n, you get algorithm T.

Using T you get circuits for permanent and circuits for permanent would give you, in fact, these are actually algorithms to compute permanent, so, you get that p^{per} has randomized polynomial time algorithm. Any problem that you can solve using permanent, you can also solve using T in randomized polynomial time which means that, $p^{per} \subseteq BPP$ which is a contradiction.

Why is it a contradiction? Well because in the very beginning you had assumed that P raised to permanent is not in BPP. That was the hypothesis we started with. So that contradiction happens. So, this contradiction shows that our assumption that B can discriminate was wrong. B cannot discriminate G from uniform distribution and which means that, A is the correct algorithm. So, Ax is mostly correct, so, this finishes the proof.

Again, the proof may look tricky but it is really just using the NW map and the way a distinguisher can be used to predict bits for NW map which in this case meant that, you are solving permanents that was the contradiction. So, if BPP $\neq x$ which we all believe then, any randomized polynomial time algorithm can be derandomized in sub exponential time with this infinitely often solution in the average case.

And we can also add infinitely often here, in case where it is infinitely often n, for each such n average case, derandomization and sub exponential time. So that is one application it is very non trivial. Second thing we will do, second application, is the theorem that we could not prove before. Now, let us move to the earlier unproved theorem.

(Refer Slide Time: 31:57)

Theorem(Impagliazzo, Kabanets and Wigderson 2001) if $NEXP \subseteq p/poly \Rightarrow NEXP = EXP$. So, this is a theorem statement where there is no mention of randomization or prg's. But believe it or not, we will prove it using prg's all these advanced methods that we have invented, whose statement has no prg.

So, the proof here is, again it will be by contradiction. So, let us assume $EXP \subsetneq NEXP \subseteq p/poly$ So, there is some opportunity here for using this Nisan Wigderson map and bit predictor and derandomization connection because, you know that there is a problem in EXP that is not in p / poly.

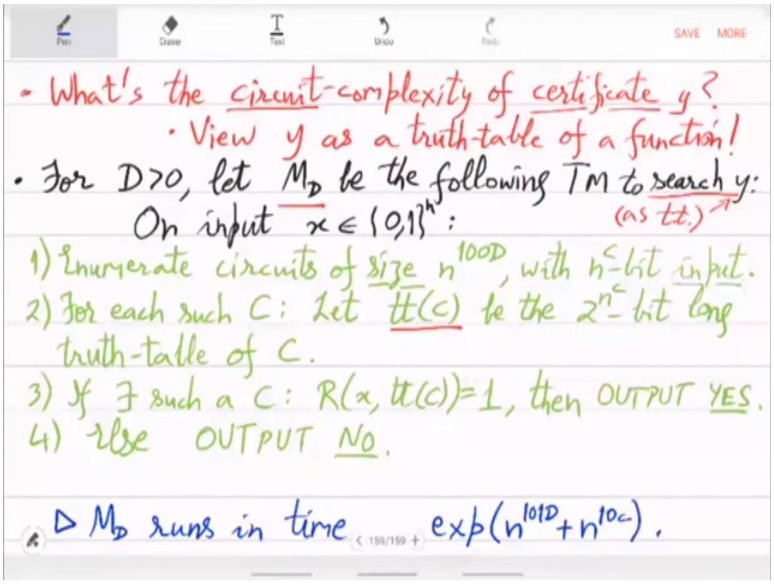
So, it is worst case hard. So, the idea is, there exists a problem in NEXP which is not in EXP it will actually be already because of NEXP different form EXP. So, since we are assuming NEXP different from EXP there is a problem L in NEXP which is not an EXP, so, it is harder than exponential time which can be used. Actually this NEXP, since NEXP is in p / poly, it also means that, EXP⊆ p / poly.
So, there is no circuit hardness, neither for NEXP nor for EXP, but since we are assuming NEXP different from EXP, there is hardness of a NEXP problem L with respect to exponential time computation and we will use that, so which can be used to get a hard function. Now, by hardness versus prg connection, we get a poly stretch, the containment of EXP ⊆ MA.

Remember that EXP when you assume NEXP in p / poly, like we recalled before, you immediately get that EXP = MA and since EXP and MA are equal, MA has some, MA obviously is this Merlin Arthur protocol. So, Arthur has random bits, so that derandomizes Arthur in this result, in this MA protocol for EXP. So, Arthur will be made, more or less it will be, I mean, the random bits which Arthur was using in this protocol, MA protocol, those bits will be eliminated that is the goal.

And then it will be like EXP is in NP that kind of a result we want to reach and then ultimately, finally contradict the time hierarchy. You finally will contradict the, let me not say which hierarchy will contradict some hierarchy that is a very rough goal. So, use L which is not an EXP to derandomize the Merlin Arthur protocols for EXP. So, let us see this in action now.
$Pick\ L \epsilon NEXP \backslash EXP. \exists c > 0\ \&\ relation\ R(x,y)\ testable\ in\ exp(|x|^{10c}) - time\ s.t\ x \epsilon L\ iff\ .$
$\exists y \epsilon \{0,1\}^{exp(|x|^c)}, [R(x,y) = 1]$So, we are assuming it to be2^{x^c} and the verifier R will also be exponential time which we are assuming to be $2^{x^{10c}}$ c time.

This is just the setting or the notation. Now, the question is, this very long string y, what can you say about its circuit complexity? Is this a hard string? Is there a compact representation for y?

(Refer Slide Time: 40:07)



So, what is the complexity of the certificate y? Now, this circuit complexity cannot be very small because if it was small then we could actually guess it faster and then that would put L in EXP that cannot happen. So, this suggests strongly that y has a large circuit complexity it is actually encoding a hard function. So, view y as a truth table. So, think of y this exponentially long string as a truth table of some function and look at the circuit complexity of that Boolean function.

So, for parameter D>0, let us define M_D be the following TM: on input $x \in \{0,1\}^n$. So, essentially we are asking the question, whether this M_D will be solving the question, whether there is a small circuit whose truth table is y. So that is what this M_D will be checking. So, 1)enumerate circuits of size n^{100D} with n^c bit input, why n^c ? Because length of y is 2^{n^c}, so, we want to look at y as a true table of n^c bit input function and it is a Boolean circuited output, so, it is only 1 bit.

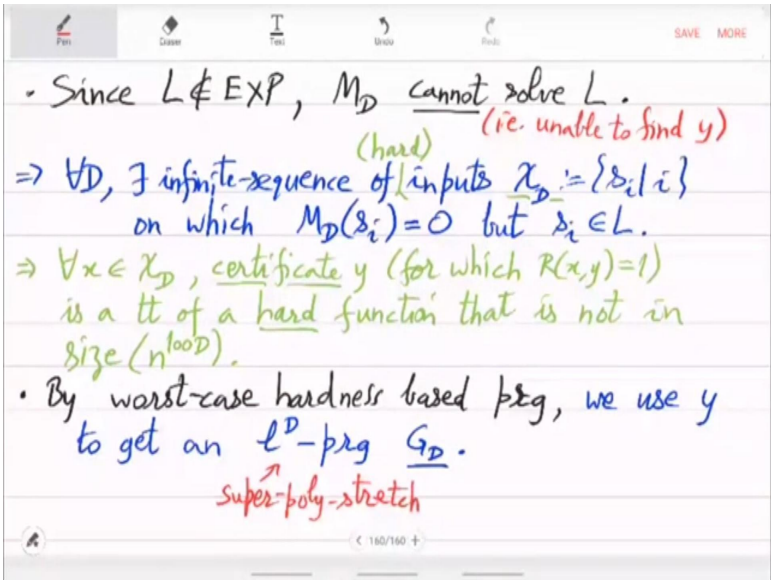
So, 2)for each such circuits c, let tt(c) be the true table, be the 2^{n^c} bit. What is a truth table? So, truth table is just evaluation of c on every possible input. There are 22^{n^c} possible inputs. So, look

at this, these evaluations as a single string that is your tt(c). So,3) if ∃ a c with R (x, tt(c)) = 1 then OUTPUT YES. So, if there is essentially what this Turing machine M D is doing there?

It is just looking at all circuits of some size n^{100D}, poly size circuits and trying to see whether the truth table of such a circuit could certify x. If it does it will say yes, otherwise it will say no.4 So, this is the Turing machine to test, to see basically it is searching y. But it is searching not all these y's, y is a very long string, so that space will be doubly exponential instead of searching for all y's, it is searching y which are truth tables, special y's as truth tables.

If it finds it says yes otherwise no. What is the time complexity of this algorithm or this Turing machine? So, M_D runs in time $\exp(n^{101D} + n^{10C})$This is the time complexity of the algorithm M_D

(Refer Slide Time: 46:46)



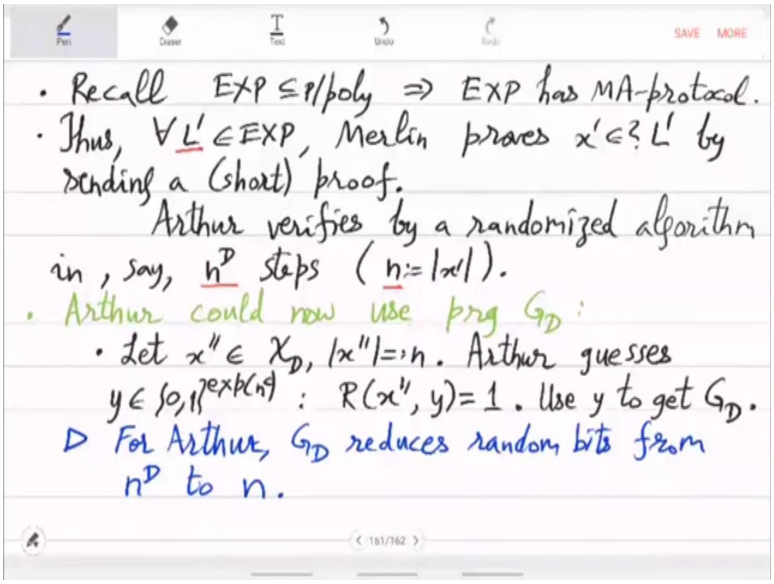
So, since L ∉ EXP, M_D cannot solve L, for any constant D, M_D will not be able to solve L which means that, M_D **will not be able to find y that is unable to find y. So, which means that,** ∀ D ∃an infinite sequence of inputs, let me call it, $\mathcal{X}_D := \{S_i | i\}$ on which $M_D(S_i) = 0$. But, input was x and what it was searching is actually a certificate. So, this is actually $S_i \in L$ Basically a certificate y exists for this S_i but it is not of this small circuit, truth table type is of a different type.

Its circuit complexity as a truth table is, the function is actually very complicated. It cannot be written as a circuit of size n^{100D}. So, remember this, there is an infinite sequence of inputs that are fooling M_D. So, $\forall x \epsilon \mathcal{X}_D$ certificate y for which R (x,y)= 1 is a truth table of a hard function. That cannot be computed that is not in size ,n^{100D}.

So, it is a function certificate for these x's, is a true table of a function that requires circuit size more than n^{100D} this is clear. So, by this worst case hardness based prg, remember that worst case implies average case, this we will prove in the next chapter and from that already we can invoke that theorem and from that what you will get is, these hard functions which y encode, these hard functions can give you a super poly stretch prg, we use y to get an l^D prg which we call G_D.

So, this hardness actually is given us G_D and how did this hardness come about? Because we assumed NEXP different from EXP and then we pick the problem, L in NEXP and based on that we have a super poly stretch, so, remember this. This much we have from the hardness to prg connection, what next? How do we get a contradiction?

(Refer Slide Time: 52:07)

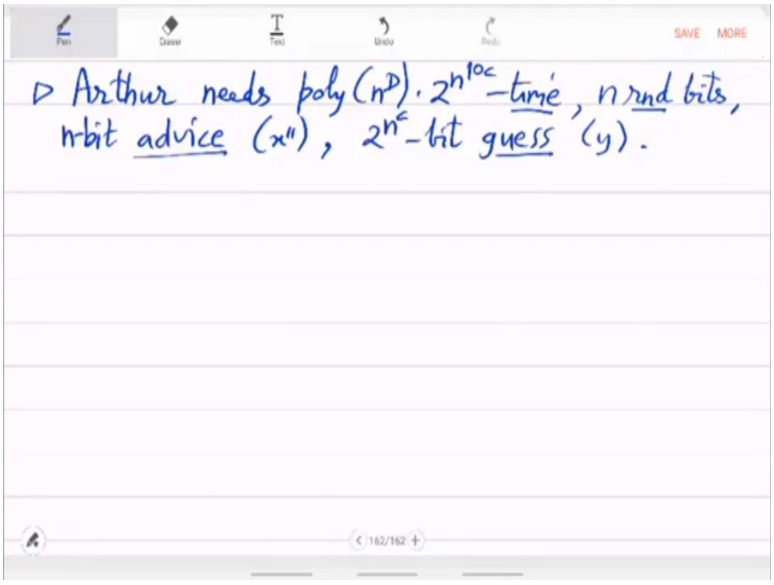


So, recall that EXP ⊆ p / poly implies that EXP has MA protocol, so, thus ∀ L'∈EXP, Merlin proves that a string x'∈$L'$by sending a proof and this short proof is verified by Arthur. Arthur

verifies it by a randomized algorithm in n^D steps. So, $(n := |x'|)$. So, every problem L' in x, Merlin will send a proof basically to show, to actually prove that x' is in L', x' is a s string.

Merlin will send a proof to Arthur and then Arthur is a randomized verifier, so, it will run an algorithm, let us say, time complexity is n^D, n is the length of the input, supposedly, s string x'. So, idea here is Arthur could use this prg G_D, so, Arthur can use G_D to remove the random bits that he needs. So, let $x'' \epsilon \mathcal{X}_D, |x''| = n$, so, Arthur guesses y $\epsilon \{0,1\}^{exp(n^c)}: R(x'', y) = 1$, use y to get G_D So, basically what is happening is, Arthur can use a hard input $\mathcal{X}_D$ and then it is known that the certificate will be truth table of a hard function, so that y Arthur can guess and once guessed it is a truth table, so, it is a function and using NW map construction G_D is the prg. So now, for Arthur G_D reduces the random bits from n^D to n. So, now fewer random bits are needed because thanks to this prg. So, what can you say about Arthur, as a verifier, what are the parameters? What is the time complexity random bit? We have also introduced non deterministic bits in Arthur. So, let us now collect all that information.

(Refer Slide Time: 57:26)



So, Arthur needs poly (n^D)- $2^{n^{10c}}$ time.Why exponential in n^{10c} time? well that is again to compute R. And Arthur was already his complexity was into the D times, so, we keep that. So, Arthur is this much time algorithm but it needs random and nondeterministic bits. So, random

bits T needs n many and x''has to be guessed that is another n bits. That is the advice needed, n advice bits that is x''.

And there is a guess to be made which is y, 2^{n^c} bit guess which is for y. So, these are the parameters. Advice means that, somehow this x'' will be needed. That is n bits, random bits are n which then can be stretched by G_D, time is $2^{n^{10c}}$ and the nondeterministic bits are 2^{n^c}, this is the parameter and we will finish the proof next time.