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

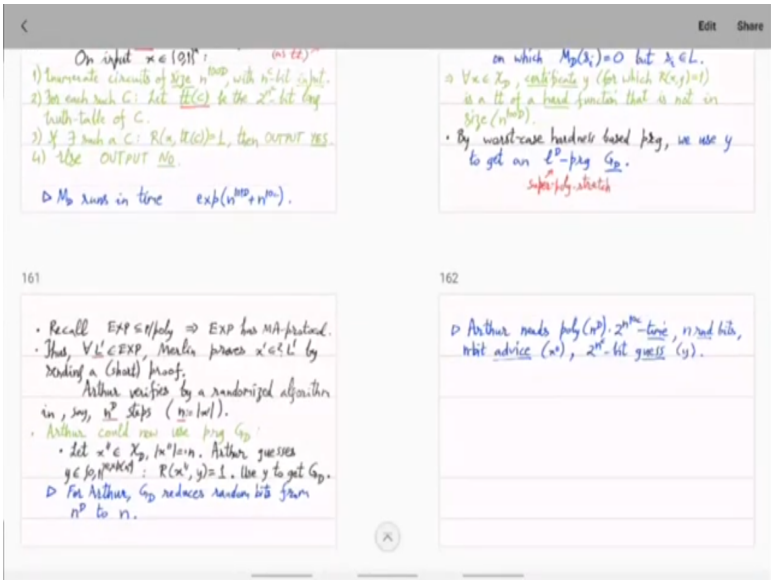
Lecture -21
Introduction to Error Correcting Codes (ECC)

Last time we almost finished this theorem by Impagliazzo Kabanets Wigderson which says that if NEXP has small circuits then NEXP = EXP.

(Refer Slide Time: 00:32)

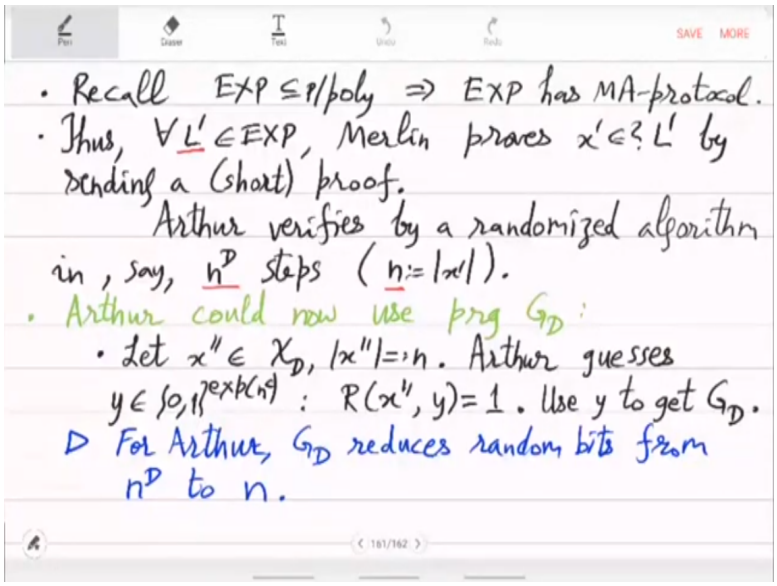
This seems to be a result of the type NP = P, but instead of P you have EXP. So, the way this is done is we will prove the contrapositive. You assume that NEXP is not EXP and when you assume that then you will get a hard problem and that hard problem then you use to create a prg. Now, it is not clear why prg will help because in the theorem statement, there is no randomization. But, actually last time, we saw that what we are trying to do is de-randomize, the Arthur Merlin or Merlin Arthur protocol.

(Refer Slide Time: 01:16)



Actually assuming NEXP in P / poly, we get that EXP is P / poly and which gives you a MA protocol for EXP and that protocol de-randomize which means we de-randomize Arthur, the verifier. So, we have already identified this hard problem and we created a prg G_D, this stretches l to l^D bits which will look pseudo random to you had to limited computation.

(Refer Slide Time: 01:55)

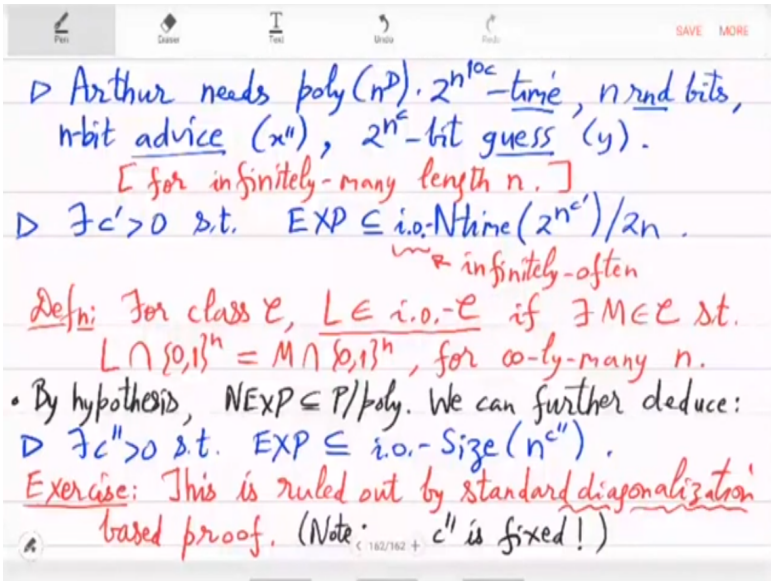


So, we were here. Arthur will now use the prg G_D, so for G_D to run you need this hard string hard input instance x'' which we defined in a set called χ_D . x'' is n bit string if somebody gives, so basically think of x'' as advice if this advice is given then the certificate y that will be hard. And

what is this relation R this is coming from the assumption that NEXP is different from EXP. So, based on that there is a language L which is hard.

The corresponding relation R(x, y) or hard strings x or for x in χ_D this certificate y will be hard and n^D is tries to it is Turing machine that tries to search for y. So, Arthur will guess this exponentially long string y and test whether R on x'', y is 1. And if this is true, then y gives the prg G_D and G_D will reduce the random bit in any algorithm from n to the power D^n.

(Refer Slide Time: 03:15)



After all this, the point where we stopped is we observe that Arthur who is the verifying algorithm for Marlin, he now needs exponential time $2^{n^{10c}}$ random bits only n and advice only n bits which is really for x'' and then he has to guess this y which will be used to devise the prg G_D. So, this guess is also exponentially large. It is exponential time verifier and the non-deterministic bits are exponential.

And remember that this χ_D has strings for infinitely many length n. What you get is the following result. You get that there exists a constant c' such that x, remember that Arthur is trying to solve x. Let x = MA is the protocol that Arthur is trying to deal and device. What you have shown is that any problem in x by the above blue statement any problem in x is infinitely often let me first write the class which is Ntimes, so non-deterministic time is exponential $2^{n^{c'}}$.

This includes in particular time and also the guess, time is exponential guess is exponential. This is n time exponential. But, this is not enough Arthur also needed n random bits and n advice, so that you can think of as 2n advice. We write it like this, the advice string is of length 2n and this is not true for all n, it is only true for certain n infinitely many. We will write infinitely often that is what we get.

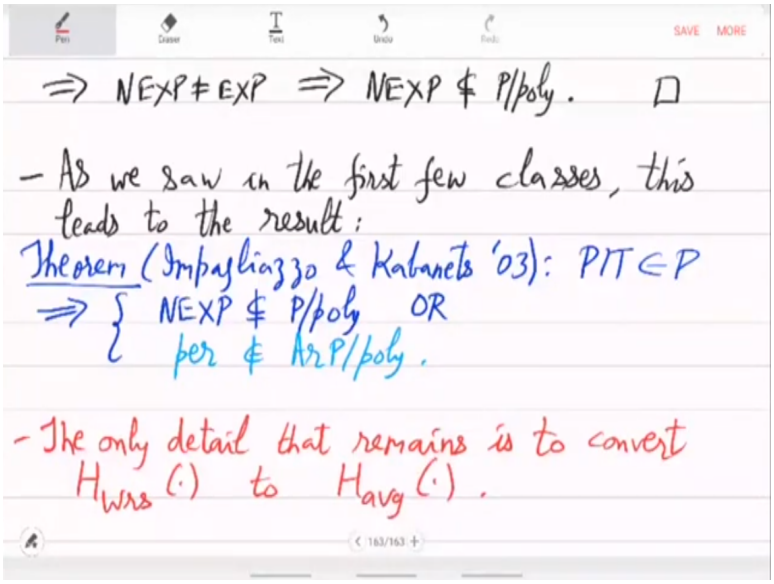
Let us define that for complexity class C we say that L is in infinitely often see if there exists a problem M in the complexity class C such that L follows M for infinitely many n. In that sense you have shown that EXP any problem in EXP because of this de-randomized MA protocol you have put it in infinitely often non-deterministic exponential time with twin advice string of length 2n that is what we have achieved in this long proof. Now, remember that we wanted to get to a contradiction. So, are we there yet?

Since, we have also assumed that NEXP in P / poly. We can further write that there exists a constant C'' such that EXP is N. Replace this Ntime with circuit size with a circuit right and that will be since NEXP is in poly, so it will be actually poly sized. You will get infinitely often size n to the c'' that is what we get from the previous Ntime statement. Now, I claim that this is a contradiction.

Why this is a contradiction because basically you are seeing that every exponential time solvable problem has a small circuit, this you can contradict by your hierarchy theorem or by a diagonalization argument. That I leave as an exercise, this is ruled out by standard diagonalization argument. EXP you can show independently that EXP cannot be in size in poly size. This is much stronger than P / poly this is actually C'' is fixed constant.

Note that C'' is fixed. Use this, it is not P / poly, it is actually our particular size and because of this, you can actually give a diagonalization based proof that EXP cannot be contained there. So, this contradiction means that our original assumption that NEXP and EXP are different that was false.

(Refer Slide Time: 10:35)

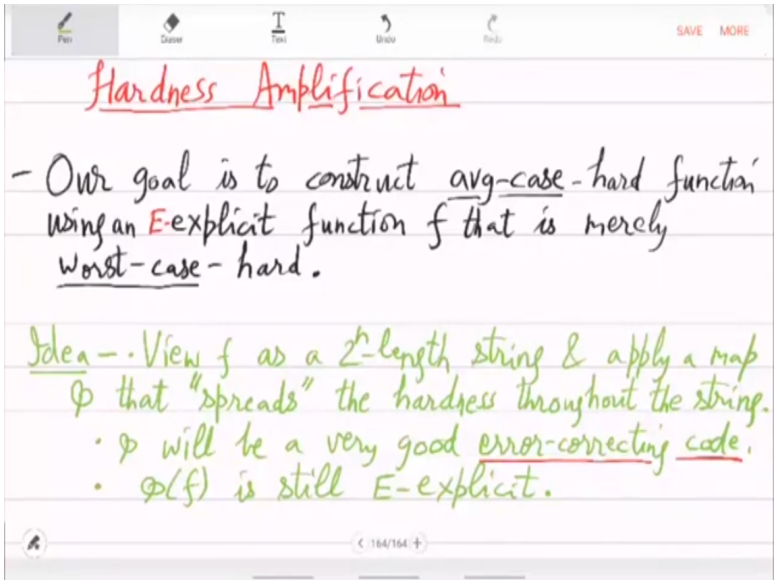


This means that if you assume NEXP and EXP different then, NEXP cannot be in P / poly. That is what we wanted to show in this theorem by Impagliazzo Kabanets Wigderson and the key thing here was this prg construction G_D which de-randomized EXP = MA result that is how you got the contradiction. So, this is a very unexpected proof technique for this theorem, because the theorem statement did not ask for any prg or randomization.

As we saw in the beginning, this leads to the result by Impagliazzo and Kabanets that PIT is in P, if you can find a deterministic polynomial time algorithm for identity testing then either NEXP is not in P / poly or so, either NEXP does not have Boolean circuits or permanent does not have arithmetic circuits. So, it is mixed this, result mixed result either Boolean circuit lower bound or arithmetic circuit lower bound.

Here we critically used this assumption NEXP in P / poly and which needed NEXP = EXP result. So, that is now completely proved. What we are missing in that proof still we are missing detailed proof of worst case to average case hardness. This will need we have used this many times before in other results. So, the only thing remaining is to convert let me write worst case hardness to average case hardness. So, this is called hardness amplification.

(Refer Slide Time: 14:12)



You have worse case hardness some function that is let us say E time computable 2^n time computable and from that function you want to get another function which is computable in similar time but it should be average case hard. So, almost all the inputs that should be hard. So, this is a new topic and it will force us to develop many new techniques. In fact a whole area that is called error correcting codes.

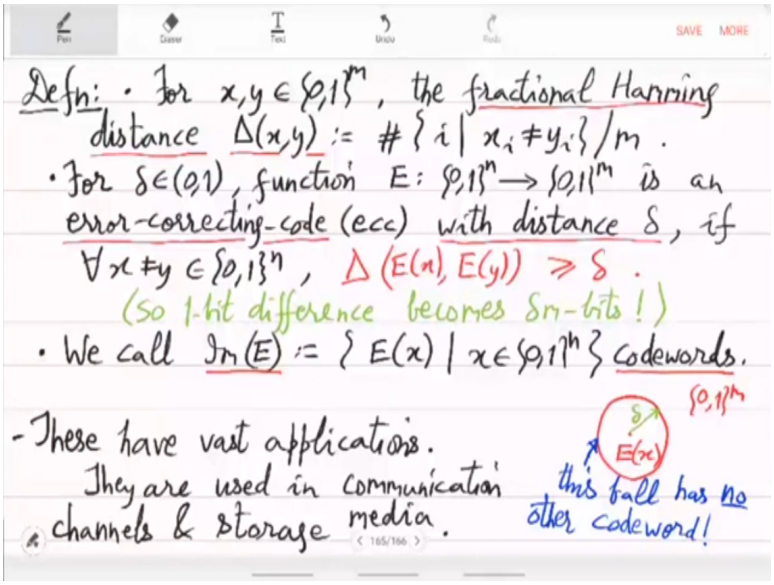
We will develop a lot of error correcting codes in this chapter and this will likely be the last chapter in this course. What is our goal here? Again our goal is to construct average case hard function using a function f that is only worst case hard that is our goal. So, how can this thing the conceivably done? How do you even start to convert a worst case problem to an average case problem, worst case hard function to average case hard function and remaining explicit you want E-explicitness.

So, the idea is that you look at the truth table of the function f as a string is a long string length 2^n and what you can do so, truth table as a string means that each location corresponds to an input instance and what is the answer there. So, the hardness is you can think of the hardness as being in one location only, it is not everywhere you want to spread it almost everywhere you want to diffuse it. So, that diffusion will be done by a great mathematical object called a code, error correcting code.

That is how you diffuse the information which is present in 1 bit to almost all the bits. So, view f as a 2^n length string and apply a map Φ that diffuses the hardness or spreads the hardness throughout the string and this map Φ will be a very good error correcting code. So, we have to first define what is an error correcting code? But I should remark here that the explicitness will not be lost.

This $\Phi(f)$ will still be because of these very good properties of the error correcting code, it will be efficient to encode and then very efficient to decode. So; which is why actually $\Phi(f)$ will remain E-computable E-explicit. Moreover, it will become from worst case it will become average case hard.

(Refer Slide Time: 20:07)



Let us define an error correcting code. So, that will need a notion of distance between strings. So, for x, y m bit strings the fractional hamming distance denoted Δ. This is the number of locations where x and y the strings differ. What we want is we want to increase the distances in terms of this fraction hamming distance. So, if when you start with 2 strings let us say x and y which are different at one location you want to design a map or devise a map Φ such that $\Phi(x)$, $\Phi(y)$ they will be different in a large fraction of the locations the bits.

You want to blow up the distance from 1 bit through almost all the bits whichever map does that we call it error correcting code. For Δ 0, 1 function E is an error correcting code will shortened to

ECC with distance Δ, if for every distinct strings x, y on n bit, so as I said what should happen is this E(x) and E(y) they should be far apart. The hamming distance between E(x), E(y) there should be at least Δ because remember originally it was what for x, y the distance was 1 / m.

It was they were different possibly only in 1 place, but way after applying either in the image they are different in Δ m locations which is fractional hamming distance delta. So, from 1 it blows up to Δ m. So, 1 bit difference becomes delta m bits. Now, to achieve this obviously, some resources have they have to be used or you have to spend some resources. So, m will be greater than n in fact, polynomially greater than n.

So, E is actually stretching the string but then the advantages that in the stretch previously x and y differing by 1 bit they are now differing by a good fraction of the bits. And this definition should suggest to you why this function E will help in amplification of the hardness worst case to average case. So, we call image of E naturally E(x) for all x, we call this either image of E or code words. So, these are the code words and as I have said from the domain when you go to code words the code words are far away.

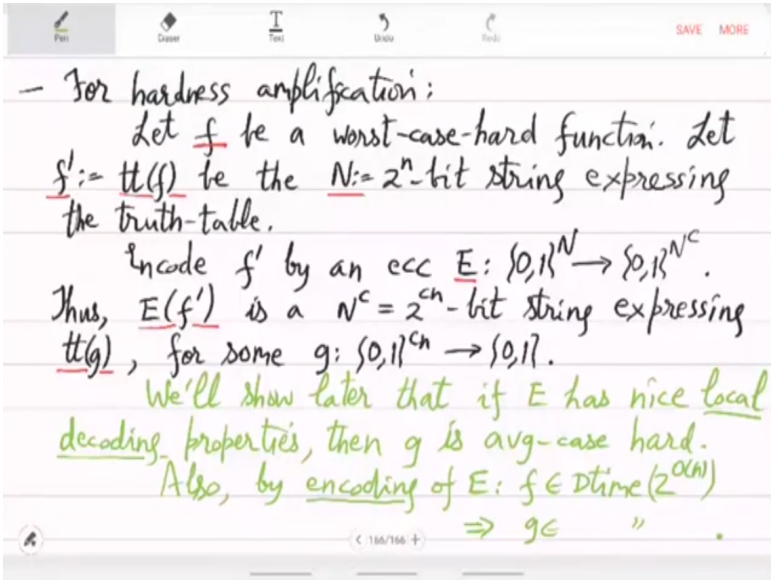
The code words I mean this fractional hamming distance or hamming distance being large means that if you draw balls around code words these even big balls will not intersect they are far apart in the space you can look at this. So, you draw these balls in the space $\{0, 1\}^m$, this you can say is the E(x), this you can say is the E(y) and between them the distance is more than delta. So; if you draw Δ / 2, so if you draw balls over radius Δ / 2 around E(x) around E(y).

These balls will not meet there will be no code word inside these walls or to put it simply let me just draw 1 ball and the radius is delta. So, what the claim is that this ball has no other code word good that is the simple claim actually. This follows from the definition of error correcting code. So, if E is in error correcting code then you pick a code word you draw ball and ball means that you are looking at strings which are at a fractional hamming distance delta or less none of these strings are code words.

Except E(x) that is the main amazing property of error-correcting code. So, these error-correcting codes have vast applications. So, they are used in physical communication channels which can be wired, it can be wireless, it can even be satellite communication any kind of communication channel you want your message to be relayed or broadcasted with errors, there will be errors, but then the receiver should be able to correct them that is the main reason why communication channels will not work without error-correcting code.

And the same reason for storage media, so they are using communication and storage because storage again hard disk or DVD or CD whatever there will be errors some bits will get flipped because of the physical device error and still you want the information to be recoverable. So, you have to actually store the information in some coded form and that is done by error-correcting code. The reason why this will be good also in hardness amplification, we can talk about it in a bit more detail than before.

(Refer Slide Time: 30:05)



For hardness amplification, we will discuss now how exactly E will act and time complexity. We will quickly go through the parameters for hardness amplification, how this error-correcting code E will act, so let f be a worst-case hard function, of course, we will only look at Boolean functions, so, let f' be the truth table be the 2^n bit string we will call it N expressing the truth table, so remember this string is a very long string.

It is 2^n bit and on this, you will apply the error-correcting code. So, encode f' by an error-correcting code that stretches N bits. Now, since we will only look at efficient error-correcting codes obviously the stretch will only be polynomial. So, N will stretch to N^c. Thus if you look at E of f' this is an equally long string which you can again think of as a truth table.

This f' is a N^c which is 2^{cn} bit string expressing the truth table of another function right this is a long Boolean string you can again think of it as a function we are calling it g. So, for some g, cn bit to 1 bit. So, f was the worst-case hard function f' is its truth table it is a N bit long string using error-correcting code E for polynomial stretch and E(f') is now seen as a truth table of another Boolean function g.

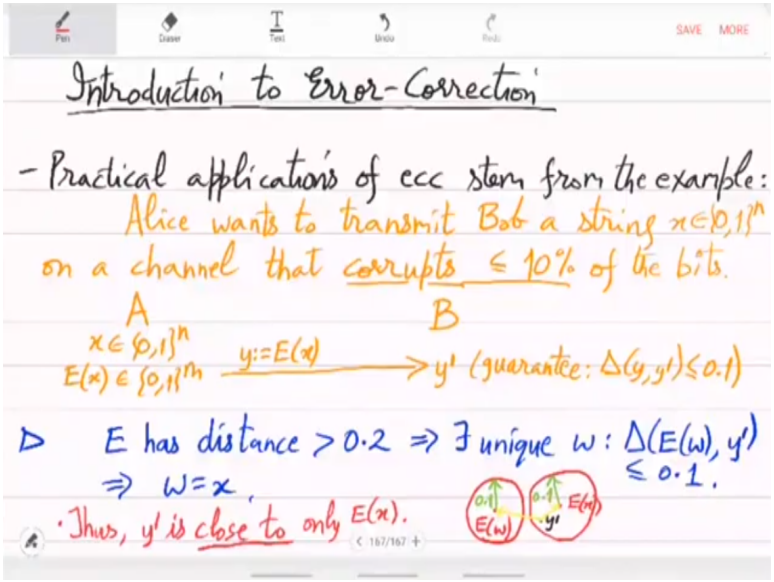
So, in this topic we will show that if E has so encoding you have already assumed will be polynomial time, it is polynomial stretch polynomial time. But decoding has actually many possibilities. We will gradually make it better and better. So, the ultimate thing that we will need for this amplification to work is nice local decoding properties. It has those then g is average-case hard. So, this local decoding is needed because the string we are talking about is very long, we cannot work with that as a whole.

At this point, you may recall this log-space algorithm for undirected graph connectivity. So, there also we had the same problem that the graph was too large for our workspace, but we solved it. Similarly here also the string is too large for us to look at the whole string. So, we actually want decoding by just querying some locations logarithmically many locations. And from those locations, we should be able to construct the original bit of the plain text or the string f' that we started with.

So that is why we will need local decoding and if this local decoding is there, then we will immediately get the g is average-case hard. Also by encoding, we will get if f is in Dtime(2^n) then g as well. So, encoding will imply that g is explicit, which is 2^n time computable. And local decoding will imply that g is very hard, much harder than what f was. Now, what we will do is we will go deeper into error-correcting codes.

Some properties of it, then how to construct them, how to do those encodings and what is the distance and finally, all kinds of decoding that we will need. So, it will be standard decoding or list decoding and finally, local list decoding it these are the topics that we will now cover.

(Refer Slide Time: 38:04)



Let us start that. So, what is error correction? Why is it useful? And what kind of codes do we need. Practical applications of error-correcting codes stem from the following example. What happens in this example is Alice wants to transmit Bob string x or some channel on a channel that will corrupt a fraction of the bits. Let us say that corrupts at most 10% of the bits. This corruption is involved in the channel.

But it is limited by some percentage in general it is limited it is something below 50%, let us say 49% and below. So, we model it in the following picture. So, Alice A has a message x n bit and the encoding the code word is E(x) That is m bits, this is now what is sent to Bob. So, y send but then corruption happens and what Bob gets is y'. Some of the bits of y have been flipped by the channel because of the error.

Now, the question is how will Bob recover y and then recover x from y', the guarantee is the distance fractional hamming distance is at most 10% which is 0.1. So, what you need is an error-correcting code E whose distances double this corruption. For 0.1 corruptions the distance

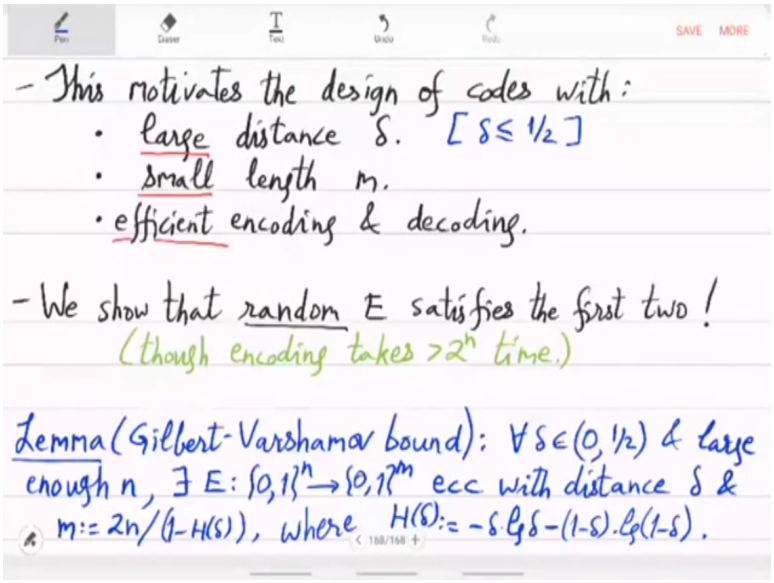
should be 0.2, if that is the case then in the ball that you will draw around y' there will be only 1 code word and that will be y. So, Bob can get x uniquely.

If E has a distance greater than 0.2 then there exists a unique w such that the distance of this code word E(w) from this y' is less than equal to 0.1. This means that w has to be x so, the proof of this is you look at again as I was trying before look at E(w) get E(x) and look at E(w) you know that y' is as so, these first of all these balls have radius 0.1 and what you are told is that in this ball around x of strings with fractional hamming distance less than or equal to 0.1, y' is there.

Now you have this information that y' is close to E(x). If you also assume that y' is close to E(w) that will be a contradiction. This is E(x), so the distance that you are looking at is this. So, y' if you assume is close to E(x) and if you assume that y' is close to E(w) then that would mean that both E(x) and E(w) are also close. But since both are code words you know that these two code words are far away, they are further away than 0.2.

So, which means that E(w) has to be far away from y'. So, that is the basic idea if you take code words far away then even if you perturb one of the code word E(x) to y' still it will not come close to any other code word it will remain far away. So, this y' is close to only E(x) and this is at least existentially Bob will be able to recover x but that there is an algorithm or not fast algorithm or not that is still not clear from this picture, but you can see the relationship between the distance of a code and the amount of error that can be tolerated.

(Refer Slide Time: 46:06)



This motivates the design of codes, to tolerate maximum possible error you should pick a code with a large distance. Now, one thing to note is the distance between two codewords the maximum distance that you can get is only half right you cannot have all the code words each of the pair having a distance of more than half that is not possible. So, note that δ will be less than equal to half more than this is impossible, but you want to get as close as possible to δ.

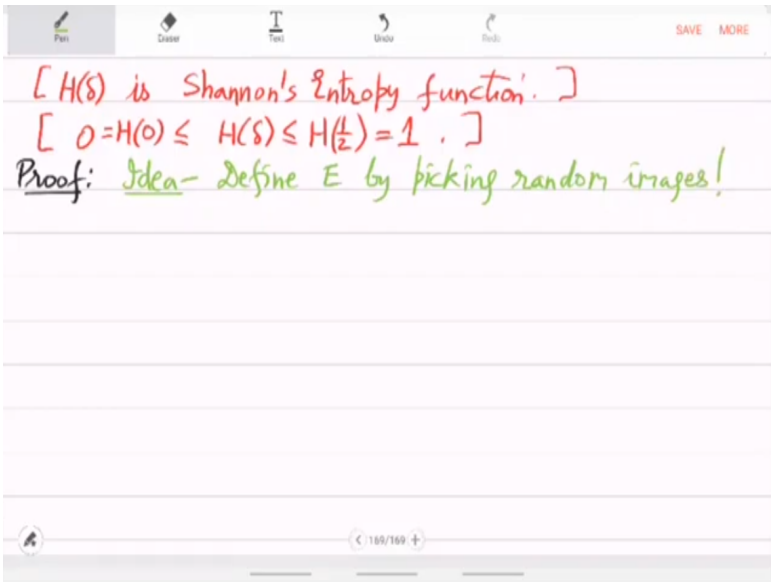
And hence you will be able to then handle errors or Bob will be able to handle errors up to δ / 2 you want the stretch to be not too much you want the code to be efficient. Small length m and goes without saying that you want everything efficient, so efficient encoding and decoding. We want to design codes with these properties. We showed that random codes does satisfy the first two properties which means that random code has δ close to half; distance close to half.

And it does not need much of a stretch m small it will not be efficient because in your space you have 2^n strings and you are assigning to every string you are assigning a random m bit string so, it is obviously the time is 2^n already for encoding. But if you are willing to pay this price then this much of time if you have then you can have the best code which is simply the random code. How do you prove this? So, let us first state it formally. How big is δ and how small is m? This is called Gilbert-Varshamov bound.

For any δ in fact, positive less than half for any delta and large enough n there exists a map E which is an error-correcting code with distance δ and what is m? It is 2n / 1 – H(delta) where H is the entropy function it is $-\delta \log \delta - (1-\delta) \cdot \log(1-\delta)$. So, δ is an absolute constant between 0 and half strictly in that range. So, each delta is also a fraction and m then is just a constant multiple of n.

And we will show actually that just a random choice works for a random map will already be an error correcting code with δ distance which you have picked anything between 0 and half and the stretch that it will do is only a linear it is only by a constant multiple.

(Refer Slide Time: 52:26)



This H(δ) is Shannon's entropy function and you will check that H(0) is 0 and H(½) is 1 and everything else is in between. So, H(δ) is just a fraction between 0 and 1 and hence, so, this m is just order n. So, what is the idea? Idea is that define E by picking random strings in the space. We have already fixed the space image space $\{0, 1\}^m$, m is this number order n and just if you just pick random strings from that space and assign to any string or to the respective corresponding string is $\{0, 1\}^n$ that will satisfy the δ property. We will prove this next time.