PROFESSOR: OK. Just to review where we are, we've been talking about source coding as one of the two major parts of digital communication. Remember, you take sources, you turn them into bits. Then you take bits and you transmit them over channels. And that sums up the whole course. This is the part where you transmit over channels. This is the part where you process the sources. We're concentrating now on the source side of things.

Partly because by concentrating on the source side of things, we will build up the machinery we need to look at the channel side of things. The channel side is really more interesting, I think. Although there's been a great deal of work on both of them. They're both important.

And we said that we could separate source coding into three pieces. If you start out with a waveform source, the typical thing to do, and almost the only thing to do, is to turn those waveforms into sequences of numbers. Those sequences might be samples. They might be numbers in an expansion. They might be whatever. But the first thing you almost always do is turn waveforms into a sequence of numbers. Because waveforms are just too complicated to deal with.

The next thing we do with a sequence of numbers is quantize them. After we quantize them we wind up with a finite set of symbols. And the next thing we do is, we take that sequence of symbols. And -- and what we do at that point is to do data compression. So that we try to represent those symbols with as small as possible a number of binary digits per source symbol. We want to do that in such a way that we can receive it the other end.

So let's review a little bit about what we've done in the last couple of lectures. We talked about the Kraft inequality. And the Kraft inequality, you remember, says that the lengths of the codewords in any prefix-free code have to satisfy this funny inequality here. And this funny inequality, in some sense, says if you want to make one codeword short, by making that one codeword short, it eats up a large part of this fraction. Since this sum has to be less than or equal to 1. If, for example, you make l sub 1 equal to 1, then that uses up a half of this sum right there. And all the other codewords have to be much longer. So what this is saying, essentially, and we proved it, and we did a bunch of things with it, and your homework you worked with it, we have shown that that inequality has to hold.

The next thing we did is, given a set of probabilities on a source, for example, p1 up to p sub m, by this time you should feel very comfortable in realizing that what you call these symbols doesn't make any difference whatsoever as far as any matter of encoding sources is concerned. The first thing you can do, if you like to, is take whatever name somebody has given to a set of symbols, replace them with your own symbols, and the easiest set of symbols to use is the integers 1 to m. And that's what we will usually do.

So, given this set of probabilities, and they have to add up to 1, the Huffman algorithm is this ingenious algorithm, very, very, clever, which constructs a prefix-free code of minimum expected length. And that minimum expected length is just defined as the sum over i, of p sub i times l sub i. And the trick in the algorithm is to find that set of l sub i's that satisfy this inequality but minimize this expected value.

Next thing we started to talk about was a discrete memoryless source. A discrete memoryless source is really a toy source. It's a toy source where you assume that each letter in the sequence is independent, and equally independent, and identically distributed. In other words, every letter is the same. Every letter is independent of every other letter. That's more appropriate for a gambling game than it is for real sources. But, on the other hand, by understanding this problem, we're starting to see that we understand the whole problem of source coding. So we'll get on with that as we go.

But, anyway, when we have a discrete memoryless source, what we found -- first we defined the entropy of such a source as h of x, which is the sum over i of minus p sub i, of logarithm of p sub i. And that was just something that came out of trying to do this optimization not the way that Huffman did it but the way that Shannon did it. Namely, Shannon looked at this optimization in terms of dealing with entropy and things like that. Huffman dealt with it in terms of finding the optimal code.

One of the very surprising things is that the way Huffman looked at it, in terms of entropy, is the way this really valuable, even though it doesn't come up with an optimal code. I mean, here was poor Huffman, who generated this beautiful algorithm, which is extraordinarily simple, which solved what looked like a hard problem. But yet, as far as information theory is concerned, he used that algorithm, sure. But as far as all the generalizations are concerned, it has almost nothing to do with anything.

But, anyway, when you look at this entropy, what comes out of it is the fact that the entropy of the source is less than or equal to the minimum number of bits per source symbol that you can come up with in a prefix-free code, which is less than h of x plus 1. And the way we did that was just to try to look at this minimization. And by looking at the minimization, we usually showed it had to be greater than or equal to H of x. And by looking at any code which satisfied this inequality with any set of length -- well, after we looked at this, this said that what we really wanted to do was to make the length of each codeword minus log of p sub i. That's not an integer. So the thing we did to get this inequality is said, OK, if it's not an integer, we'll raise it up the next value. Make it an integer. And as soon as we do that, the Kraft inequality is satisfied. And you can generate a code with that set of lengths. So that's where you get these two bounds.

This bound here is usually very, very weak. Can anybody suggest the almost unique example where this is almost tight? It's a simplistic sample you can think of. It's a binary source. And what binary source has the property that this is almost equal to this? Anybody out there?

AUDIENCE: [UNINTELLIGIBLE]

PROFESSOR: Make it almost probability 0 and probability 1. You can't quite do that because as soon as you make the probability of the 0, 0, then you don't have to represent it. And you just know that it's a sequence of all 1's. So you're all done. And you don't need any bits to represent it. But if there's just some very tiny epsilon probability of a 0 and a big probability of a 1, then the entropy is almost equal to 0. And this 1 here is needed because l bar min is 1. You can't make it any smaller than that. So, that's where that comes from.

Let's talk about entropy just a little bit. If we have an alphabet which has size m, which is the symbol we'll usually use for the alphabet, x. And the probability that x equals i, is p sub i. In other words, again, we're using this convention so we're going to call the symbols the integers 1 to capital M. Then the entropy is equal to this. And a nice way of representing this is that the entropy is equal to the expected value of minus the logarithm. Logarithms are always to the base 2, in this course. When we want natural logarithms we'll write ln, in other words it's log to the base 2. So it's log to the base 2 of the probability of the symbol x.

We call this the log pmf random variable. We started out with x being a chance variable. I mean, we happen to have turned it into a random variable because we've given it numbers. But that's irrelevant. We really want to think of it as a chance variable. But now this quantity is a numerical function of the symbol which comes out of the source. And, therefore, this quantity is a well-defined random variable. And we call it the log pmf random variable. Some people call it self-information. We'll find out why later. I don't particularly like that word. One, because what we're dealing with here has nothing to do with information. Probably the thought that this all has something to do with information, namely, that information theory has something to do with information, probably held up the field for about five years. Because everyone tried to figure out, what does this have to do with information. And, of course, it had nothing to do with information. It really only had to do with data. And with probabilities of various things in the data. So, anyway. Some people call it self-information and we'll see why later. But this is the quantity we're interested in.

And we call it log pmf, we sort of forget about the minus sign. It's not good to forget about the minus sign, but I always do it. So I sort of expect other people to do it, too. One of the properties of entropy is, it has to be greater than or equal to 0. Why is it greater than or equal to 0? Well, because these probabilities here have to be less than or equal to 1. And the logarithm of something less than or equal to 1 is negative. And therefore minus the logarithm has to be greater than or equal to 0.

This entropy is also less than or equal to log M, log capital M. I'm not going to prove that here. It's proven in the notes, it's a trivial thing to do. Or maybe it's proven in one of the problems, I forget. But, anyway, you can do it using the inequality log of x is less than or equal to x minus 1. Just like all inequalities can be proven with that inequality.

So there's a quality here if x is equiprobable. Which is pretty clear, because if all of these probabilities are equal to 1 over M, and you take the expected value of logarithm of M, you get logarithm of M. So there's nothing very surprising here.

Now, the next thing -- and here's where what we're going to do is, on one hand very simple, and on the other hand very confusing. After you get the picture of it, it becomes very simple. Before that, it all looks rather strange. If you have two independent chance variables, say x and y, then the choice where the sample value of the chance variable x, and the choice of the sample value y, together that's a pair of sample values which we can view as one sample value. In other words, we can view xy as a chance variable all in its own right. This isn't the sequence x, followed by y, where you can think of it that way. But we want to think of this here as a chance variable itself. Which takes on different values. And the values it takes on are pairs of sample values, 1 from x, 1 from ensemble y, and xy takes on the sample value of little xy with the probability p of x times p of y.

As we move one with this course, we'll become much less careful about putting these subscripts down, which talk about random variables. And the arguments which talk about sample values of those random variables. I want to keep doing it for a while because most courses in probability, even 6.041, which is the first course in probability, almost deliberately fudges the difference between sample values and random variables. And most people who work with probability do this all the time. And you never know when you're talking about a random variable and when you're talking about a sample value of a random variable. And that's convenient for getting insight about thing. And you do it for a while and then pretty soon you wonder, what the heck is going on. Because you have no idea what's a random variable any more, and what's a sample value of it.

So, this entropy, H, of the chance variable, xy, is then expected value of minus the logarithm of the probability of the chance variable, xy. Mainly, when you take the expected value, you're taking the expected value of a random variable. And for the random variable here, in the chance variables, xy and here. This is the expected value of minus the logarithm of p of x times the probability p of x. Which is the expected value. Since they're independent of each other it's the sum. And that gives you H of x y is equal to H of x plus H of y. This is really the reason why you're interested in these chance variables which are logarithms of probabilities. Because when you have independent chance variables then you have the situation that probabilities multiply and therefore log probabilities add.

All of the major theorems in probability theory, in particular the law of large numbers, which is the most important result in probability, simple though it might be, that's the key to everything in probability. That particular result talks about sums of random variables and not about products of random variables. So that's why Shannon did everything in terms of these log PMF. And we will soon be doing everything in terms of log PMF also.

So now let's get back to discrete memoryless sources. If you now have a block of n chance variables, x1 to xn, and they're all IID, again we can do this whole block as one big monster chance variable. If each one of these takes on m possible values, then this monster chance variable can take on m to the n possible values. Namely, each possible string of symbols, each possible string of n symbols where each one is a choice from the integers 1 to capital M. So we're talking about tuples of numbers now. And those are the values that this particular chance variable, x sub n, takes on.

So it takes on these probabilities, takes on these values with the probability p of x n is the product from i equals 1 to n, of the individual probabilities. In other words, again, when you have independent chance variables, the probabilities multiply. Which is all I'm saying here.

So the chance variable x sub n has the entropy H of x n, which is the expected value of minus the logarithm of that probability. Which is the expected value of minus the logarithm of the product of probabilities, which is the expected value of the sum of minus the log of the probabilities, which is n times H of x.

If you compare this with the previous slide, you'll see I haven't said anything new. This argument and this argument are really exactly the same. All I did was do it for two chance variables first. And then observe. But it generalizes, to an arbitrary number of chance variables. You can say that it generalizes to an infinite number of chance variables also. And in some sense it does. And I would advise you not to go there. Because you just get tangled up with a lot of mathematics that you don't need.

So the next thing is, how do we fix the variable prefix-free codes and what do we gain by it? So the thing we're going to do now, instead of trying to compress one symbol at a time from the source, we're going to segment the source into blocks of n symbols each. And after we segment it into blocks of n symbols each, we're going to encode the block of n symbols. Now, what's new there? Nothing whatsoever is new. A block of n symbols is just a chance variable. We know how to do optimal encoding of chance variables. Namely, we use the Huffman algorithm. You can do that here on these n blocks. We also have this nice theorem, which says that the entropy -- well, first the entropy of x n as n times the entropy of x. So, in other words, when you have independent identically distributed chance variables, this entropy is just n times this entropy. But the important thing is this result of doing the encoding. Which is the same result we had before. Namely, this is the result of what happens when you take a set -- when you take an alphabet, and the alphabet can be anything whatsoever. And you encode that alphabet in an optimal way, according to the probabilities of each symbol within the alphabet. And the result that you get is the entropy of this big chance variable x sub n is less than or equal to the minimum -- well, it's less than or equal to the expected value of the length of a code, whatever code you have. But when you put the minimum on here, this is less than the entropy of the chance variable x super n plus 1. That's the same theorem that we proved before. There's nothing new here.

Now, if you divide this by n, this by n, and this by n, you still have a valid inequality. When you divide this by n, what do you get? You get H of x. When you divide this by n, by definition L bar -- what we mean is the number of bits per source symbol. We have n source symbols here. So when we divide by n, we get the number of bits per source symbol in this monster symbol. So l min is equal to this. When we divide this by n, we get this. When we divide this by n, we get H of x. And now the whole reason for doing this is, this silly little 1 here, which we're trying very hard to think of as being negligible or unimportant, has suddenly become 1 over n. And by making n big enough, this truly is unimportant.

If you're thinking in terms of encoding a binary source, this 1 here is very important. In other words, when you're trying to encode a binary source, if you're encoding one letter at a time, there's nothing you can do. You're absolutely stuck. Because if both of those letters have non-zero probabilities, and you want a uniquely decodable code, and you want to find codewords for each of those two symbols, the best you can do is to have an expected length of 1. Namely, you need 1 to encode 1, and you need 0 to encode 0. And there's nothing else, there's no freedom at all that you have. So you say, OK, in that situation, I really have to go to longer blocks. And when I go to longer blocks, I can get this resolved. And I know how to do it. I use Huffman's algorithm or whatever. So, suddenly, I can start to get the expected number of bits per source symbol. Down as close to h of x as I want to make it. And I can't make it any smaller. Which says that H of x now has a very clear interpretation, at least for prefix-free codes, of being the number of bits you need for encoding prefix-free codes when you allow the possibility of encoding them a block at a time.

We're going to find later today that the significance of this is far greater than that, even. Because why use prefix-free codes, we could use any old kind of code. When we study the Lempel-Ziv codes tomorrow, we'll find out they aren't prefix-free codes at all. They're really variable length of variable length codes. So they aren't fixed to variable length. And they do some pretty fancy and tricky things. But they're still limited to this same inequality. You can never beat the entropy bound. If you want something to be uniquely decodable, you're stuck with this bound. And we'll see why in a very straightforward way, later. It's a very straightforward way which I can guarantee all of you are going to look at it and say, yes, that's obvious. And tomorrow you will look at it and say, I don't understand that at all. And the next day you'll look at it again and say, well, of course. And the day after that you'll say, I don't understand that. And you'll go back and forth like that for about two weeks. Don't be frustrated, because it is simple. But at the same time it's very sophisticated.

So, let's now review the weak law of large numbers. I usually just call it the law of large numbers. I bridle a little bit when people call it weak because, in fact it's the centerpiece of probability theory. But there is this other thing called the strong law of large numbers, which mathematicians love because it lets them look at all kinds of mathematical minutiae. It's also important, I shouldn't play it down too much. And there are places where you truly need it. For what we'll be doing, we don't need it at all. And the weak law of large numbers does in fact hold in many places where the strong law doesn't hold. So if you know what the strong law, is temporarily forget it for the for the rest of the term. And just focus on the weak law. And the weak law is not terribly complicated.

We have a sequence of random variables. And each of them has a mean y bar. And each of them has a variance sigma sub y squared. And let's assume that they're independent and identically distributed for the time being. Just to avoid worrying about anything. If we look at the sum of those random variables, namely a is equal to y1 up to y sub n. Then the expected value of a is the expected value of this plus the expected valuable of y2, and so forth. So the expected value of a is n times y bar. And I think in one of the homework problems, you found the variance of a. And the variance of a, well, the easiest thing to do is to reduce this to its fluctuation. Reduce all of these to their fluctuation. Then look at the variance of the fluctuation, which is just the expected value of this squared. Which is the expected value of this squared plus the expected value of this squared, and so forth.

So the variance of a is n times sigma sub y squared. I want all of you know that. That's sort of day two of a probability course. As soon as you start talking about random variables, that's one of the key things that you do. One of the most important things you do.

The thing that we're interested in here is more the sample average of y1 up to y sub n. And the sample average, by definition, is the sum divided by n. So in other words, the thing that you're interested in here is to add all of these random variables up. Take one over n times it. Which is a thing we do all the time. I mean, we sum up a lot of events, we divide by n, and we hope by doing that to get some sort of typical value. And, usually there is some sort of typical value that arises from that. What the law of large numbers says is that there in fact is a typical value that arises. So this sample value is a over n, which is the sum divided by n. And we call that the sample average.

The mean of the sample average is just the mean of a, which is n times y bar, divided by n. So the mean of the sample average is y bar itself. The variance of the sample variance, -- the variance of the sample average, OK, that's, -- I'm talking too fast.

The sample average here, you would like to think of it as something which is known and specific, like the expected value. It, in fact, is a random variable. It changes with different sample values. It can change from almost nothing to very large quantities. And what we're interested in saying is that most of the time, it's close to the expected value. And that's what we're aiming at here. And that's what the law of large numbers says.

The sample average here, the variance of this, is now equal to the variance of a divided by n squared. In other words, we're trying to take the expected value of this quantity squared. So there's a 1 over n squared that comes in here. When you take the 1 over n squared here, this variance then becomes sigma -- I don't know why I have the n there. Just take that n out, if you will. I don't have my red pen with me. And so it's the variance of the random variable y, divided by n. In other words, the limit as n goes to infinity of the of the variance of the sum is equal to infinity. And the variance of the sample average as n goes to infinity is equal to 0. And that's because of this 1 over n squared effect here. When you add up a lot of independent random variables, what you wind up with is the sample average has a variance, which is going to 0. And the sum has a variance which is going to infinity. That's important. Aside from all of the theorems you've ever heard, this is sort of the gross, simple-minded thing which you always ought to keep foremost in your mind. This is what's happening in probability theory. When you talk about sample averages, this variance is getting small.

Let's look at a picture of this. Let's look at the distribution function of this random variable. The sample average as a random variable. And what we're finding here is that this distribution function, if we look at it for some modest value of n, we get something which looks like this upper curve here. Which is then the lower curve here. It's spread out more, so it has a larger variance. Namely, the sample average has a larger variance.

When you make n bigger, what's happening to the variance? The variance is getting smaller. The variance is getting smaller by a factor of 1/2. So this quantity is supposed to have a variance which is equal to 1/2 of the variance of this. How you find a variance in a distribution function is your problem. But you know that if something has a small variance, it's very closely tucked in around this. In other words, as the variance goes to 0, and the mean is y bar, you have a distribution function which approaches a unit step.

And all that just comes from this very, very simple argument that says, when you have a sum of IID random variables and you take the sample average of it, namely, you divide by n, the variance goes to 0. Which says, no matter how you look at it, you wind up with something that looks like a unit step.

Now, the Chebyshev inequality, which is one of the simpler inequalities in probability theory, and I don't prove it because it's something you've all seen. I don't know of any course in probability which avoids the Chebyshev inequality. And what it says is, for any epsilon greater than 0, the probability that the difference between the sample average and the true mean, the probability that that quantity and magnitude is greater than or equal to epsilon, is less than or equal to sigma sub y squared divided by n epsilon squared.

Oh, incidentally that thing that was called sigma sub n before was really sigma squared. That's mainly the variance of y. I hope it's right in the notes. Might not be. It is? Good.

So, that's what this inequality says. There's an easy way to derive this on the fly. Namely, if you're wondering what all these constants are here, here's a way to do it. What we're looking at, in this curve here, is we're trying to say, how much probability is there outside of these plus and minus epsilon limits. And the Chebyshev inequality says there can't be too much probability out here. And there can't be too much probability out here.

So, one way to get at this is to say, OK, suppose I have some given probability out here. And some given probability out here. And suppose I want to minimize the variance of a random variable which has that much probability out here and that much probability out here. How do I do it? Well, the variance deals with the square of how far you were away from the mean. So if I want to have a certain amount of probability out here, I minimize my variance by making this come straight, come up here with a little step, then go across here. Go up here. And then, oops. Go up here. I wish I had my red pencil. Does anybody have a red pen? That will write on this stuff? Yes? No? Oh, great. Thank you. I will return it.

So what we want is something which goes over here. Comes up here. Goes across here. Goes up here. Goes across here, and goes up again. That's the smallest you can make the variance. It's squeezing everything in as far as it can be squeezed in.

Namely, everything out here gets squeezed in to y minus epsilon. Everything in here gets squeezed into 0. And everything out here gets squeezed into epsilon. OK, calculate the variance there. And that satisfies the Chebyshev inequality with equality.

So that's all the Chebyshev inequality is. And it's a loose inequality usually, because usually these curves look very nice. Usually this looks like a Gaussian distribution function, and the central limit theorem says that we don't need the central limit theorem here, and we don't want the central limit theorem here, because this thing is an inequality that says, life can't be any worse than this. And all the central limit theorem is, is an approximation. And then we have to worry about when it's a good approximation and when it's not a good approximation.

So this says, when we carry it one piece further, it's for any epsilon and delta greater than 0. If we make n large enough -- in other words, substitute delta for this. And then, when you make n small enough, this quantity is smaller than delta. And that says that the probability that s and y differ by more than epsilon is less than or equal to delta when we make n big enough. So it says, you can make this as small as you want. You can make this as small as you want. And all you need to do is make a sequence which is long enough.

Now, the thing which is mystifying about the law of large numbers is, you need both the epsilon and the delta. You can't get rid of either of them. In other words, you can't say -- you can't reduce this to 0. Because it won't make any sense. In other words, this curve here tends to spread out on its tails. And therefore, there's always a probability of error there. You can't move epsilon into 0 because, for no finite end, do you really get a step function here. So you need some wiggle room on both end. You need wiggle room here, and you need wiggle room here. And once you recognize that you need those two pieces of wiggle room. Namely, you cannot talk about the probability that this is equal to y bar, because that's usually 0. And you cannot talk about reducing this to 0 either. So both of those are needed.

Why did I go through all of this? Well, partly because it's important. But partly because I want to talk about something which is called the asymptotic equipartition property. And because of those long words you believe this has to be very complicated. I hope to convince you that what the asymptotic equipartition property is, is simply the week law of large numbers applied to the log pmf. Because that, in fact, is all it is. But it says some unusual and fascinating things.

So let's suppose that x1, x2, and so forth is the output from a discrete memoryless source. Look at the log pmf of each of these. Namely, they each have the same distribution function. So w of f x is going to be equal to minus the logarithm of p sub x of x, for each of these chance variables. w of x maps source symbols into real numbers. So there's a random variable, capital W of x sub j, which is a random variable. We have a random variable for each one of these symbols to come out of the source. So, for each one of these symbols, there's this log pmf random variable, which takes on different values.

So the expected value of this log pmf, for the j'th symbol out of the source is the sum of p sub x of x, namely the probability that the source takes on the value x, times minus the logarithm of p sub x. And that's just the entropy. So, the strange feeling about this log pmf random variable is its expected value is entropy. And w of x1, w of x2, and so forth, are a sequence of IID random variables, each one of them which has a mean, which is the entropy.

So it's just exactly the situation we had before. Instead of y bar, we have the entropy of x. And instead of the random variable y sub j, we have this random variable w of x sub j, which is defined by the symbol in an alphabet.

And just to review this, but it's what we said before, if capital X1, this little x1, namely, if little x1 is the sample value for the chance variable x1 and if x2 is a sample value for the chance variable X2, then the outcome for w of x1 plus w of x2 -- very hard to keep all these little letters and capital letters straight. Is w of x1 plus w of x2 is minus the logarithm of the probability of x1 minus the logarithm of the probability of x2, which is minus the logarithm of the product, which is minus the logarithm of the joint probability of x1 and x2, which is the random variable w of x1 x2. So the sum here is the random variable corresponding to log pmf of the joint outputs x1 and x2.

So w of x1 x2 is the log pmf of the event, but this joint chance variable takes on the value x1 x2. And the random variable x1 x2 is the sum of x1 and x2. So, what have I done here? I said this at the end of the last slide, and you won't believe me. So, again this is one of these things where tomorrow you won't believe me. And you'll have to go back and look at that. But, anyway, x1 x2 is a chance variable. And probabilities multiply in log pmf's add, which is what we've been saying for a couple of days now.

So. If I look at the sum of n of these random variables, the sum of these log probabilities is the sum of the log of pmf's, which is minus the logarithm of the probability of the entire sequence. That's just saying the same thing we said before, for two random variables. The sample average of a log pmf's is the sum of the w's divided by n, which is minus the logarithm of the probability divided by n. The weak law of large numbers applies, and the probability that this sample average minus the expected value of w of x is greater than or equal to epsilon is less than or equal to this quantity here.

This quantity is minus the logarithm of the probability of x sub n, divided by n, minus H of x, greater than or equal to epsilon. So this is the thing that we really want.

I'm going to spend a few slides trying to say what this means. But let's try to just look at it now, and see what it must mean. It says that with high probability, this quantity is almost the same as this quantity. It says that with high probability, the thing which comes out of the source is going to have a probability, a log probability, divided by n, which is close to the entropy. It says in some sense that with high probability, the probability of what comes out of the source is almost a constant. And that's amazing. That's what you'll wake up in the morning and say, I don't believe that. But it's true. But you have to be careful to interpret it right.

So, we're going to define the typical set. Namely, this is the typical set of x's, which come out of the source. Namely, the typical set of blocks of n symbols out of the source. And when you talk about a typical set, you want something which includes most of the probability. So what I'm going to include in this typical set is all of these things that we talked about before. Namely, we showed that the probability that this quantity is greater than or equal to epsilon is very small. So, with high probability what comes out of the source satisfies this inequality here. So I can write down the distribution function of this random variable here. It's just this w -- this is a random variable w. I'm looking at the distribution of that random variable w. And this quantity in here is the probability of this typical set. In other words, when I draw this distribution function for this combined random variable, I've defined this typical set to be all the sequences which lie between this point and this point. Namely, this is H minus epsilon. And this is H plus epsilon, moving H out here. So these are all the sequences which satisfy this inequality here. So that's what I mean by the typical set. It's all things which are clustered around H in this distribution function.

And as n approaches infinity, this typical set approaches probability 1. In the same way that the law of large numbers behaves that way. The probability that x sub n is in this typical set is greater than or equal to 1 minus sigma squared divided by n epsilon squared. Let's try to express that in a bunch of other ways.

If you're getting lost, please ask questions. But I hope to come back to this in a little bit, after we finish a little more of the story. So, another way of expressing this typical set -- let me look at that as the typical set. If I take this inequality here and I rewrite this, namely, this is the set of x's for which this is less than epsilon plus H of x and greater than H of x minus epsilon. So that's what I've expressed here. It's the set of x's for which n times H of x minus epsilon is less than this logarithm of probability is great less than n times H of x plus epsilon. Namely, I'm looking at this range of epsilon around H, which is this and this.

If I write it again, if I exponentiate all of this, it's the set of x's for which 2 to the minus n, H of x, plus epsilon, that's this term exponentiated, is less than this is less than this term exponentiated. And what's going on here is, I've taken care of the minus sign also. And if you can follow that in your head, you're a better person than I am. But, anyway, it works. And if you fiddle around with that, you'll see that that's what it is.

So this typical set is a bound on the probabilities of all these typical sequences. The typical sequences all are enclosed in this range of probabilities. So the typical elements are approximately equiprobable, in this strange sense above. Why do I say this is a strange sense? Well, as n gets large, what happens here? This is 2 to the minus n times H of x. Which is the important part of this. This epsilon here is multiplied by n. And we're trying to say, as n gets very, very big, we can make epsilon very, very small. But we really can't make n times epsilon very negligible. But the point is, the important thing here is, 2 to the minus n times H of x. So, in some sense, this is close to 2 to the minus n H of x. In what sense is it true?

Well, it's true in that sense. Where that, in fact is, a valid inequality. Namely in terms of sample averages, these things are close. When I do the exponentiation and get rid of the n and all that stuff, they aren't very close. But saying this sort of thing is sort of like saying that 10 to the minus 23 is approximately equal to 10 to the minus 25. And they're approximately equal because they're both very, very small. And that's the kind of thing that's going on here. And you're trying to distinguish 10 to the minus 23 and 10 to the minus 25 from 10 to the minus 60th and from 10 to the minus three. So that's the kind of approximations we're using. Namely, we're using approximations on a log scale, instead of approximations of ordinary numbers.

But, still it's convenient to think of these typical x's, typical sequences, as being sequences which are constrained in probability in this way. And this is the thing which is easy to work with. The atypical set of strings, namely, the compliment to this set, the thing we know about that is the entire set doesn't have much probability. Namely, if you fix epsilon and you let n get bigger and bigger, this atypical set becomes totally negligible. And you can ignore it. So let's plow ahead. Stop for an example pretty soon, but --

If I have a sequence which is in the typical set, we then know that its probability is greater than 2 to the minus n times H of x plus epsilon. That's what we said before. And, therefore, when I use this inequality, the probability of x to the n, for something in the typical set, is greater than this quantity here. In other words, this is greater than that. For everything in a typical set. So now I'm heading over things in a typical set. So I need to include the number of things in a typical set. So what I have is this sum. And what is this sum?

This is the probability of the typical set. Because I'm adding overall elements in the typical set. And it's greater than or equal to the number of elements in a typical set times these small probabilities. If I turn this around, it says that the number of elements in a typical set is less than 2 to the n times H of x plus epsilon. For any epsilon, no matter how small I want to make it. Which says that the elements in a typical set have probabilities which are about 2 to the minus n times H of x. And the number of them is approximately 2 to the n times H of x. In other words, what it says is that this typical set is a bunch of essentially uniform probabilities.

So what I've done is to take this very complicated source. And when I look at these very humongous chance variables, which are very large sequences out of the source, what I find is that there's a bunch of things which collectively have zilch probability. There's a bunch of other things which all have equal probability. And a number of them is enough to add up to y. So I have turned this source, when I look at it over a long enough sequence, into a source of equiprobable events. And each of those events has this probability here.

Now, we know how to encode equiprobable events. And that's the whole point of this. So, this is less than or equal to that. On the other side, we know that 1 minus delta is less than or equal to this probability of a typical set. And this is less than the number of elements in a typical set times 2 to the minus n h of x minus epsilon. This is an upper bound on this. This is less than this.

So I just add all these things up and I get this bound. So it says, the size of the typical set is greater than 1 minus delta, times this quantity. In other words, this is a pretty exact sort of thing. If you don't mind dealing with this 2 to the n epsilon factor here. If you agree that that's negligible in some strange sense, the all of this makes good sense. And if it is negligible, let me start talking about source coding, which is why this all works out.

So the summary is that the probability of the complement of the typical set is essentially 0. The number of elements in a typical set is approximately 2 to the n times h of x. I'm getting rid of all the deltas and epsilons here, to get sort of the broad view of what's important here. Each of the elements in a typical set has probability 2 to the minus n times H of x. So I've turned a source into a source of equiprobable elements. And there are 2 to the n times h of x of them.

Let's do an example of this. It's an example that you'll work on more in the homework and do it a little more cleanly. Let's look at a binary discrete memoryless source, where the probability that x is equal to 1 is p, which is less than 1/2. And the probability of 0 is greater than 1/2. So, this is what you get when you have a biased coin. And the biased coin has a 1 on one side and a 0 on the other side. And it's more likely to come up 0's than 1's.

I always used to wonder how to make a biased coin. And I can give you a little experiment which shows you you can make a biased coin. I mean, a biased is a little round thing which is flat on the top and bottom. Suppose instead of that you make a triangular coin. And instead of making it flat on top and bottom, you turn it into a tetrahedron. So in fact, what this is now is a coin which is built up on one side into a very massive thing. And is flat on the other side. Since it's a tetrahedron and it's an equilateral tetrahedron, the probability of 1 is going to be 1/4, and the probability of 0 is going to be 3/4. So you can make biased coins. So when you get into coin-tossing games with people, watch the coin that they're using. It probably won't be a tetrahedron, but anyway.

So the entropy here, the log pmf random variable, takes on the value of minus log p with probability p. And it takes on the value minus log 1 minus p, with probability 1 minus p. This is a probability of a 1. This is a probability of a 0. So, the entropy is equal to this.

Used to be that in information theory courses, people would almost memorize what this curve looked like. And they'd draw pictures of it. There were famous curves of this function, which looks like this. 0, 1, 1.

Turns out, that's not all that important a distribution. It's a nice example to talk about. The typical set, t epsilon n, is the set of strings with about p n1's and about 1 minus p times n 0's. In other words, that's the typical thing to happen. And it's the typical thing in terms of this law of large numbers here. Because you get 1's with probability p. And therefore in a long sequence, you're going to get about pn 1's and 1 minus p 0's.

The probability of a typical string is, if you get a string with this many 1's and this many 0's, it's probability is p. Namely, the probability of a 1 times the number of 1's you get, which is pn. Times the probability of a 0, times the number of 0's you get. And if you look at what this is, if you take p up in the exponent and 1 minus the p up in the exponent, this becomes 2 to the minus n times h of x, just like what it should be. So these typical strings, with about pn 1's and 1 minus pn 0's, are in fact typical in the sense we've been talking about.

The number of n strings with pn 1's is n factorial divided by pn factorial divided by n times 1 minus p factorial. I mean I hope you learned that a long time ago, but you should learn it in probability anyway. It's just very simple combinatorics.

So you have that many different strings. So what I'm trying to get across here is, there are a bunch of different things going on here. We can talk about the random variable which is the number of 1's that occur in this long sequence. And with high probability, the number of 1's that occur is close to pn. But if pn 1's occur, there's still an awful lot of randomness left. Because we have to worry about where those pn 1's appear. And those are the sequences we're talking about. So, there are this many sequences, all of which have that many 1's in them. And there's a similar number of sequences for all similar numbers of 1's. Namely, if you take pn plus 1 and pn plus 2, pn minus 1, pn minus 2, you get similar numbers here. So those are the typical sequences.

Now, the important thing to observe here is that you really have 2 to the n binary strings altogether. And what this result is saying is that collectively those don't make any difference. The law of large numbers says, OK, there's just a humongous number of strings. You get the largest number strings which have about half 1's and half 0's. But their probability is zilch. So the thing which is probable is getting pn 1's and 1 minus pn 0's.

Now, we have this typical set. What is the most likely sequence of all, in this experiment? How do I maximize the probability of a particular sequence? The probability of the sequence is p to the i times 1 minus p to the n minus i. And 1 minus p is the probability of 0. And p is the probability of a 1. How do I choose i to maximize this? Yeah?

AUDIENCE: [UNINTELLIGIBLE] all 0's.

PROFESSOR: You make them all 0's. So the most likely sequence is all 0's. But that's not a typical sequence. Why isn't it a typical sequence? Because we chose to define typical sequence in a different way. Namely is only one of those, and there are only n of them with only a single one. So, in other words, what's going on is that we have an enormous number of sequences which have around half 1's and half 0's. But they don't have any probability. And collectively they don't have any probability. We have a very small number of sequences which have a very large number of 0's. But there aren't enough of those to make any difference. And, therefore, the things that make a difference are these typical things which have about np 1's and 1 minus pn 0's.

And that all sounds very strange. But if I phrase this a different way, you would all say that's exactly the way you ought to do things. Because, in fact, when we look at very, very long sequences, you know with extraordinarily high probability what's going to come out of the source is something with about pn 1's and about 1 minus p times n 0's. So that's the likely set of things to have happen. And it's just that there are an enormous number of those things. There are this many of them. So, here what we're dealing with is a balance between the number of elements of a particular type, and the probability of them. And it turns out that this number and its probability balance out to say that usually what you get is about pn 1's and 1 minus p times n 0's. Which is what the law of large numbers said to begin with. All we're doing is interpreting that here. But the thing that you see from this example is, all of these things with exactly pn 1's in them, assuming that pn is an integer, are all equiprobable. They're all exactly equiprobable. So what we're doing when we're talking about this typical set, is first throwing out all the things which have to many 1's are or too few 1's in them. We're keeping only the ones which are typical in the sense that they obey the law of large numbers. And in this case, they obey the law of large numbers for log pmf's also. And then all of those things are about equally probable.

So the idea in source coding is, one of the ways to deal with source coding is, you want to assign codewords to only these typical things. Now, maybe you might want to assign codewords to something like all 0's also. Because it hardly costs anything. And a Huffman code would certainly do that. But it's not very important whether you do or not. The important thing is, you assign codewords to all of these typical sequences.

So let's go back to fixed-to-fixed length source codes. We talked a little bit about fixed-to-fixed length source codes before. Do you remember what we did with fixed-to-fixed length source codes before? We said we have an alphabet of size m. We want something which is uniquely decodable. And since we want something which is uniquely decodable, we have to provide codewords for everything. And, therefore, if we want to choose a block length of n, we've got to generate m to the n codewords.

Here we say, wow, maybe we don't have to provide codewords for everything. Maybe we're willing to tolerate a certain small probability that the whole thing fails and falls on its face. Now, does that make any sense? Well, view things the following way. We said, when we started out all of this, that we were going to look at prefix-free codes. Where some codewords had a longer length and some codewords had a shorter length. And we were thinking of encoding either single letters at a time, or a small block of letters at a time. So think of encoding, say, 10 letters at a time. And think of doing this for 10 to the 20th letters.

So you have the source here which is pumping out letters at a regular rate. You're blocking them into n letters at a time. You're encoding in a prefix-free code. Out comes something. What comes is not coming out at a regular right. What is coming out, sometimes you get a lot of bits out. Sometimes a small number of bits out. So, in other words, if you want to send things over a channel, you need a buffer there to save things. If, in fact, we decide that the expected number of bits per source letter is, say, five bits per source letter, then we expect over a very long time to be producing five bits per source letter. And if we turn our channel on for one year, to transmit all of these things, what's going to happen is this very unlikely sequence occurs. Which in fact requires not one year to transmit, but two years to transmit.

In fact, what do we do if it takes one year and five minutes to transmit instead of one year? Well, we've got a failure. Somehow or other, the network is going to fail us. I mean we all know that networks fail all the time despite what engineers say. I mean, all of us who use networks know that they do crazy things. And one of those crazy things is that unusual things sometimes happen. So, we develop this very nice theory of prefix-free codes. But prefix-free codes, in fact, fail also. And they fail also because buffers overflow. In other words, we are counting on encoding things with a certain number of bits per source symbol. And if these unusual things occur, and we have too many bits per source symbol, then we fail.

So the idea that we're trying to get at now is that prefix-free codes and fixed-to-fixed length source codes which only encode typical things. In fact, are sort of the same if you look at them over a very, very large sequence length. In other words, if you look at a prefix-free code which is dealing with blocks of 10 letters, and you look at a fixed-to-fixed length code which is only dealing with typical things but is looking at a length of 10 to the 20th, then over that length of 10 to the 20th, your variable length code is going to have a bunch of things which are about the length they ought to be. And a bunch of other things which are extraordinarily long. The bunch of things which are extraordinarily long are extraordinarily unpopular, but there are an extraordinarily large number of them. Just like with a fixed-to-fixed length code, you are going to fail. And you're going to fail on an extraordinary number of different sequences. But, collectively, that set of sequences don't have any probability.

So the point that I'm trying to get across is that, really, these two situations come together when we look very long lengths. Namely, prefix-free codes are just a way of generating codes that work for typical sequences and over a very large, long period of time, will generate about the right number of symbols. And that's what I'm trying to get at here. Or what I'm trying to get at in the next slide.

So the fixed-to-fixed length source code, I'm going to pick some epsilon and some delta. Namely, that epsilon and delta which appeared in the law of large numbers. I'm going to make n as big as I have to make it for that epsilon and that delta. And calculate how large it has to be, but we won't. Then I'm going to assign fixed length codewords to each sequence in the typical set.

Now, am I going to really build something which does this? Of course not. I mean, I'm talking about truly humongous lengths. So, this is really a conceptual tool to understand what's going on. It's not something we're going to implement. So I'm going to assign codewords to all these typical elements. And then what I find is that since the typical set, since the number of elements in it is less than 2 to the n times H of x plus epsilon, if I choose L bar, namely, the number of bits I'm going to use for encoding these things, it's going to have to be H of x plus epsilon in length. Because I need to provide codewords for each of these things. And it needs to be an extra 1 over n because of this integer constraint that we've been dealing with all along, which doesn't make any difference.

So if I choose L bar, that big, in other words, if I make it just a little bit bigger than the entropy, the probability of failure is going to be less than or equal to delta. And I can make delta -- and I can make the probability of failure as small as I want. So I can make this epsilon here which is the extra bits per source symbol as small as I want. So it says I can come as close to the entropy bound in doing this, and come as close to unique decodability as I want in doing this. And I have a fixed-to-fixed length code, which after one year is going to stop. And I can turn my decoder off. I can turn my encoder off. I can go buy a new encoder and a new decoder, which presumably works a little bit better. And there isn't any problem about when to turn it off. Because I know I can turn it off. Because everything will have come in by then.

Here's a more interesting story. Suppose I choose the number of bits per source symbol that I'm going to use to be less than or equal to the entropy minus 2 epsilon. Why 2 epsilon? Well, just wait a second. I mean, 2 epsilon is small and epsilon is small. But I want to compare with this other epsilon and my law of large numbers. And I'm going to pick n large enough. The number of typical sequences, we said before, was greater than 1 minus delta times 2 to the n times h of x minus epsilon. I'm going to make this epsilon the same as that epsilon, which is why I wanted this to be 2 epsilon.

So my typical set is this big when I choose n large enough. And this says that most of the typical set can't be assigned codewords. In other words, this number here is humongously larger then 2 to the l bar, which is in the order of 2 to the nh of x minus 2 epsilon n.

So the fraction of typical elements that I can provide codewords for, between this and this, I can only provide codewords for a fraction 2 to the minus epsilon n of the codewords. We have this big sea of codewords, which are all essentially equally likely. And I can't provide codewords for even a small fraction of them. So the probability of failure is going to be 1 minus delta. The 1 minus delta's the probability that I get something atypical. Plus, well, minus in this case, 2 to the minus epsilon n, which is the probability that I can't encode a typical codeword that comes out. And this quantity goes to 1. So this says that if I'm willing to use a number of bits bigger than the entropy, I can succeed with probability very close to 1. And if I want to use a smaller number of bits, I fail with probability 1. Which is the same as saying that I'm using a prefix-free code, I'm going to run out of buffer space eventually if I run long enough. If I have something that I'm encoding -- well, just erase that. I'll say it more carefully later.

I do want to talk a little bit about this Kraft inequality for unique decodability. You remember we proved the Kraft inequality for prefix-free codes. I now want to talk about the Kraft inequality for uniquely decodable codes. And you might think that I've done all of this development of the AEP, the asymptotic equipartition property. Incidentally, you now know where those words come from. It's asymptotic because this result is valid asymptotically as n goes to infinity. It's equipartition because everything is equally likely. And its property, because it's a property. So it's the asymptotic equipartition property. And I didn't do it so I could prove the Kraft inequality. It's just that that's an extra bonus that we get. And by understanding why the Kraft inequality has to hold for uniquely decodable codes, if is one application for AEP which lets you see a little bit about how to use it.

OK, so the argument is an argument by contradiction. Suppose you generate a set of lengths for codewords. And you want this -- yeah? And the thing you would like to do is to assign codewords of these lengths. And what we want to do is to set this equal to some quantity b. In other words, suppose we beat the Kraft inequality. Suppose we can make the lengths even shorter than Kraft says we can make them. I mean, he was only a graduate student, so we've got to be able to beat his inequality somehow. So we're going to try to make this equal to b. We're going to assume that b is greater than 1. And then what we're going to do is to show that we get a contradiction here. And this same argument can work whether we have a discrete memoryless source or a source with memory, or anything else. It can work with blocks, it can work with variable length to variable length codes. It's all essentially the same argument.

So what I want to do is to get a contradiction. I'm going to choose a discrete memoryless source. And I'm going to make the probabilities equal to 1 over b times 2 to the minus li. In other words, I can generate a discrete memoryless source for talking about it with any probabilities I want to give it. So I'm going to generate one with these probabilities. So the lengths are going to be equal to minus log of b times p sub i. Which says that the expected length of the codewords is equal to the sum of p sub i l sub i, which is equal to the entropy minus the logarithm of b. Which means I can get an expected length which is a little bit less than the entropy.

So now what I'm going to do is to consider strings of n source letters. I'm going to make these string very, very long. When I concatenate all these codewords, I'm going to wind up with a length that's less than n times H of x minus b over 2, minus log b over 2 with high probability. And as a fixed-length code of this length it's going to have a low failure probability. And, therefore, what this says is I can, using this remarkable code with unique decodability, and generating very long strings from it, I can generate a fixed-length code which has a low failure probability. And I just showed you in the last slide that I can't do that. The probability of failure with such a code has to be essentially 1. So that's a contradiction that says you can't have these unique decodable codes.

If you didn't get that in what I said, don't be surprised. Because all I'm trying to do is to steer you towards how to look at the section in the notes that does that. It was a little too fast and a little too late. But, anyway, that is the Kraft inequality for unique decodability. OK, thanks.