PROFESSOR: So you guys know the quiz is cumulative, right? Everything all the way back from lecture one, so I would look at all the lectures and all the P sets, and look at all the stuff that we taught you, so data structures, algorithms, everything. And at least be able to know, for every one of them, what's the name, what it does, and wants the running time. Proofs and how it does it might be harder, but these be able to call it as a black box and argue about the running times. So I have a dp problem, and I have a non-dp problem. Which problem would you like me to start with? OK.

Do you guys know the saying, if a woodchucker would chuck wood, how much wood would a woodchucker chuck? Today we're going to chuck wood. So you have a piece of wood that is l meters long, and they have n markings. So say the first mark is at 3 meters, the second mark is at 5 meters, so on, so forth. And 3, and 4, all the way up to mn. So we want to cut this piece of wood at all the markings. The thing is the woodchucker doesn't work for free. If you give it a piece of wood of length l, and you ask it to cut it at some marking, you're going to get two pieces of wood, length l1 and l2. The price for this is l1 times l2. So we like woodchucker, but woodchuckers would also like our wallets. So we want to cut this up by paying the minimum amount of money. Rings a bell? So I'll let you guys think for a minute, then I'll give you the running time, then we'll start talking.

So we usually give you running times on quizzes. The running time is why you should know all the problems in their matching running times, because the moment we give you a running time you can automatically eliminate all the things that don't match, and just focus on a few things.

So you're going to have to cut it at all the markings, eventually, but the order in which you cut is important. So if I cut here first, then I'm going to pay three times l

minus 3, whereas if I cut in the middle first, I'm going to pay whatever this is, and 3 times l minus m3. So we're trying to decide the order. Does this look like any familiar problem?

AUDIENCE: [INAUDIBLE] using dp, right?

PROFESSOR: dp, that is good. I did say that we're going to start with a dp problem, so this is dp. It's a good start.

AUDIENCE: [INAUDIBLE]

PROFESSOR: What? Not exactly.

AUDIENCE: Yeah.

PROFESSOR: So, it is not like any problems on the recitations. So far recitations did prefixes and suffixes. We're going to solve this using a running time of n cubed, which is like the parenthesis problem. It should be what you said, but I don't know how to spell that, so we're going to go for this instead. So running n cubed-- the moment I said this you guys should know that this is the n cubed problem that we have in lecture notes. So make sure to have those on the cheat sheets, and try to understand them, right? OK, so given that I've said this, you should know the solution now. To make sure everyone is with me, we're going to go through the solution, whole. So what is a subproblem?

AUDIENCE: Smaller piece of wood.

PROFESSOR: OK.

AUDIENCE: Like how to cut it up.

PROFESSOR: OK. So this is how you think of it informally. When you write it up, I want to see this. I want to see dp of something means something. So how you fill out your dp table. It's really useful to write this up on your exam before, because one, this will help you write the recursion correctly, and two, if the grader sees this they might skim over the recursion completely. And then you might have bugs there. We might not see

them. Good for you.

So this says how you're going to fill out the table. Right? dp of something equals something. What's in a dp table? Numbers. It's never how to do something. It's always the numbers, so it's always the maximum profit, or the minimum cost, or the shortest distance, or the longest something. So it's always a number. So what we do here?

AUDIENCE: Start and dp, start location to the end location is

PROFESSOR: OK, so we're going to get the mean distance, right? We usually do i j k and whatever else it takes. So start to end is?

AUDIENCE: The minimum cost of cutting that up.

PROFESSOR: Minimum cost of cutting up the wood board from marking i, all the way to marking j. There's a tiny problem here, that the initial-- there's no problem for this big piece of wood, right? If I can only consider the board from i to j, so if I can only consider the board from marking 1 to marking n, then I get to this. So this part and this part get left out.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Exactly. We add fake markings. Then 0 is 0, and mn plus 1 equals l. Very good.

AUDIENCE: [INAUDIBLE] equally spaced?

PROFESSOR: No. So these are numbers. If they were evenly spaced, I think there's an algorithm. You might come up with a math and say, you always cut it up like this.

So while we solve this, you guys have candy, right? You should eat the candy and be energetic and everything.

So min cost of cutting up the board from marking i to marking j. I like this. Have this on your exam if possible, because this will make our life easier, and it's going to make your life easier when you get to the next step, which is how do we compute dp

of i j? So suppose I'm looking at the subboard from m1 to m4 so I'm looking at only this. How do I compute the best way to cut the board from m1 to m4? What are my options?

AUDIENCE: The locations you can cut it.

PROFESSOR: Exactly. So in order to cut this up, I can either make a first cut at m2. So say I make my first cut here, and then I recursively cut this, and cut this. Or the other alternative is take the same guy-- m1, m2, m3, m4-- cut it at m3, and then recursively cut this, and recursively cut this. So I'm iterating over all the markings inside the board. Now suppose I'm cutting it-- yes?

AUDIENCE: [INAUDIBLE] cutting both, or actually, never mind.

PROFESSOR: Yeah, when I recursed, that takes care of it. So suppose I'm looking at m1 through m4, and I'm cutting it at m2. What's the total cost? So what's the best way to cut, given that then I know I'm going to cut there?

AUDIENCE: The sum of the dp's.

PROFESSOR: OK, so it's the best way to cut m1 through m2, plus best way to cut m2 through m4, plus the price I'm paying for this cut, right? Not just the sum of the dp's. One more term. What's this term?

AUDIENCE: 4 minus 1? Or the location of 4 minus the location of 1.

PROFESSOR: So, not quite, almost. So if I'm cutting a board into two pieces, the cost is the product of the length of the two pieces. m2 minus m1, times yes. OK, why did I bother doing this? Some people think better with concrete numbers. If that's the case, then give yourself an example. Write some numbers on your sheet of paper, then see what letters match to what numbers, and copy it up using letters. And there you go, you've solved the problem.

So where are i and j here?

AUDIENCE: i would be 1.

PROFESSOR: OK, so this is i.

AUDIENCE: That's j.

PROFESSOR: Cool, so let's try to write it up, now. So in order to cut the board from i to j, what am I doing? So what am I computing? Usually the first word in your subproblem definition is the function that you're going to use. So it's minimum, and I'm going iterate over something.

AUDIENCE: dp of i to-- it has to be all of j. dp of i, j, and you're looking to--

PROFESSOR: So I'm computing dp of i j.

AUDIENCE: I know, of j minus [INAUDIBLE].

AUDIENCE: j minus i, then k j minus [INAUDIBLE].

PROFESSOR: There's a k, right? I need a new variable for where I'm going to cut up, right? So fortunately, we have a lot of letters in the alphabet, i, j, k, so on and so forth, l, m.

AUDIENCE: i plus k.

PROFESSOR: So let's say that k is the place where we cut, to make our life easy. So I'm going to have dp of

AUDIENCE: Well i is the starting point.

PROFESSOR: OK

AUDIENCE: And then, the endpoint is i plus k, right?

PROFESSOR: So what's k here?

AUDIENCE: k is an actual number. It's not the offset, it's the actual number, so it should be i to k. It depends how you define k.

PROFESSOR: So I'm going to make my life easy, and define k as exactly the marking at which I cut. k is this 2 here. And this is easier, trust me. OK, plus?

AUDIENCE: k j?

PROFESSOR: OK, and?

AUDIENCE: Cost of m--

AUDIENCE: j minus i, m of k minus m of i times m of j minus m of k.

PROFESSOR: Cool. Yeah, other way around-- doesn't matter. So now where does k go? We have to come up with numbers for the loop, right?

AUDIENCE: Between i and j.

AUDIENCE: j minus i.

AUDIENCE: Just for k in i to j.

PROFESSOR: So if I have the board from 1 to 4, do I cut at 1? I can, but that's kind of weird. Because I'm recursing on the same subproblem. By the way, if you recurse to the same subproblem, what are you going to get as your running time? Infinite. So let's not do that. So we're going to go from?

AUDIENCE: [INAUDIBLE]

PROFESSOR: So going from i would be bad. So i plus 1. 2?

AUDIENCE: j minus 1.

PROFESSOR: Very good.

AUDIENCE: Would it be m over i plus 1, because [INAUDIBLE].

PROFESSOR: So k is which marking I'm cutting at. I never want to cut inside a marking. However, I don't even know these are integers.

AUDIENCE: They wouldn't be called [INAUDIBLE].

PROFESSOR: So k is which marking, i, j, and k are which marking I'm cutting at. These are the

only discrete things I have. This board is all filled with real numbers. So if I want to cut somewhere here, that's a real number-- I don't like that. I want to have integers. So my markings help me get integers. I only want to cut at the marking, so I always look at my problem in terms of which marking I'm cutting it. So this always iterates over markings.

So this looks very much like the parentheses problem, right? Same subproblems, roughly the same recursion. Turns out that these problems, where you're not considering suffixes or prefixes, but rather you're considering substrings, are reasonably hard to come by, and reasonably hard to solve. So if we give these to you, chances are they're going to be exactly like the parentheses problem, except for the cost function. This isn't what we had in the parentheses problem, right? So you should be prepared to solve problems that look exactly like the paren problem, but might have a different cost function. And this is how we solve it. OK.

AUDIENCE: When you say that the complexity determines which type of dp example we use, does that mean that a problem can be solved using any of dp examples? It's just that the only thing that changes is the complexity.

PROFESSOR: I don't think you can map every approach onto every problem. For example, if you tried to map prefixes onto this, you'd come up with a solution that doesn't look at all the possible choices, so your answer would be sub-optimal. So you'd come up with a fast, but incorrect algorithm. However, if you take the problem of find the longest increasing sub-sequence, you can definitely apply this technique to it. It's more general than suffixes or prefixes. So it's going to work, but it's going to be slower. So in theory, what you should do is, you have all these techniques. Given a problem, you try all the techniques. You see which ones apply, and out of those, you see which one gives you the best running time. In practice, if we give you the running time, you match it to the techniques that match the running time. You start backwards from the stuff that you know.

OK. Does this problem make sense? Sweet. Now let's do a hard problem. Do people remember hashing? You have one minute to remember hashing while I

erase the board.

[LAUGHING]

So suppose we want to implement the set. The way we're going to implement the set is, we have n elements. We're going to put them into the set, so for i goes from 1 through n, we're going to insert element i, so first we're going to insert all the elements into the set. And then after that, given a random number, we want to see is it in the set, or not. So for some other number-- I used n before, so let's use-- for some other number f, we want to see is f in the set, or is f not in the set? What data structure would you use normally for this? A hash table, right? You stick everything into a hash table, then you try to find the elements. If you find them, then you say yes. If not, then you say no. Well, it turns out that this would take more memory than what we have. So instead, we're going to do this.

We're going to have a hash table of m bits. So these are m bits. And say we have a hash function that satisfies with uniform hashing, so given any element, the value is anywhere from 0 to m minus 1, and they're all independent. So the way we're going to insert an element is-- this table is T-- we're going to say that T of h of ai equals 1. So this is a table of bits. For every element we hash the element, and we set the corresponding bit to 1. So we're going to have some 1s, and some zeros in the table. Say if this is ai, it hashes somewhere here.

OK so the question is, we inserted n elements into a table of size n. Given a new element, f, where f stands for false positive-- f is not one of the elements that we inserted. I want to know what's the probability that the set will say that the element is in the set, so basically, the probability of a false positive.

AUDIENCE: So what are we doing about [INAUDIBLE]?

PROFESSOR: Nothing.

AUDIENCE: Is it chaining, is it open addressing? Does it even matter?

PROFESSOR: So we're not inserting the elements into the table. This table only has bits. The

elements are lost completely after we insert them. So the tradeoff is uses a lot less memory. Instead of having to store entire elements, you just store bits. On the downside you're going to have false positives. Because if I have a different element, say f, if it hashes to the same location, then the set is going to say, yeah, it's in the set. So you get false positives. Would you get false negatives? No, right? Because you start out with a table of 0's, and you only set the table to ones for the numbers that match to hashes of elements that are in the set. Did you have a question? OK. OK, do we understand the problem, before we attempt to solve it?

AUDIENCE: Is it probably 1/m?

PROFESSOR: You'd wish, but no.

AUDIENCE: It's less than n/m.

PROFESSOR: OK, I like that. So what are you thinking?

AUDIENCE: If there are no collisions previously, then it would equal to n/m, but there are collisions, probably collisions.

PROFESSOR: OK, I'm going to open up a window in your head and tell everyone else the small steps you took to get here. So we have this new number f. How are we going to check if it's in the set or not? We're going to compute h of f, and we're going to check if t of h of f is 0 or 1. f is different from all the other elements. So its hash value is independent from all the other hash values we had before. We don't really care about this anymore, after we have the independence assumption. So h of f is just some random position in the table. So the question is, given some random position in the table, will that be a 0 or a 1? How do you know? If I knew how many 1's I have in the table-- if I have k 1's in the table, and automatically this means n minus k 0's-- then what's the probability that h of f will point to a 1?

AUDIENCE: k/m.

PROFESSOR: Yes. So the hash takes m possible values. k of them are 1's. So the probability that the hash is going to guess a 1 is k/m. So if we knew how many 1's we have, then

this is the answer. We know that we're going to have at most n 1's-- that's what you're thinking, right? So k is definitely smaller or equal to n, so the answer definitely has to be smaller or equal than n/m. Now if you're in a rush, you might say, well, we inserted n elements, so we're definitely going to have n 1's here. That is not true. The hashes of all the elements are independent. So there is some probability that two elements will hash to the same value, and as the number of elements grows, that probability also grows.

OK, so now by looking at this, we got rid of this part of the problem. We don't care that there's a new element. We don't care that it's a false positive. All that we care about is how many 1's do we have in the table after inserting n values. Well, what's that? That's m times the probability that a slot in the table is 1. Right, the probability that the slot in the table is 1 is k/m. So if we know this probability, and we multiply it by m, then we get k. People still with me?

AUDIENCE: And what does that variable represent, h?

PROFESSOR: This is k. Represents that my handwriting sucks, basically.

AUDIENCE: I mean, why do we do m times the probability. That's the the expected number of 1's in the table?

PROFESSOR: Yeah. Yeah, this is E of k, I guess. So then our final answer is this thing divided by m. So the answer is the expected value of k, or you can just think of it as the average value of k, divided by m. So this is m times this probability, divided by m. So it is exactly this probability. So the thing that we want to focus on is, what's the probability that a random slot in the table is a 1?

AUDIENCE: It's equal to 1 minus the probability that it was never fixed.

PROFESSOR: Exactly, the first thing we do. 1 minus the probability that a slot is 0. This is easy, right, like it looks easy. But this makes a huge difference, because once we're here, well, a slot is zero if none of the insertions made it a one. And the insertions are all independent. So this is like, you're flipping a coin. What's the probability that after you flip it n times, you never get a head? So this is 1 minus

AUDIENCE: 1 over m to the something.

PROFESSOR: That-- So a slot is 0 means that no number was inserted in it. We're inserting n numbers, so it's the probability that a single number was not necessarily in the slot, raised to the power of n. So we have n independent experiments, right? Every time you insert a number into the hash function, that's one experiment. The hash function gives you independent values for all the elements. So all the insertions are independent of each other. If, in a single insertion, you've hit that slot, then you've made it a 1-- game over. So the slot is only a zero if none of the insertions make it the 1. So you take the probability that the insertion doesn't make it a one, and you raise it to the power n, because that has to happen n times in order for the whole thing to be successful.

And the probability that the number was not inserted in a slot is 1 minus the probability that it was inserted. Right, we're doing this again. 1 minus probability that a number hit. Well what this probability? Uniform hashing.

AUDIENCE: 1/m

PROFESSOR: 1/m. So this whole thing is 1 minus 1 minus 1, over m to the power n. 1 minus m minus 1, over m to the power n.

AUDIENCE: Can we go through this again. From 1 minus probability of a slot is 0, to 1 minus probability of a number was not inserted in a slot?

PROFESSOR: OK. So first off, the point of the problem. It's our problem, right? Don't panic, don't be angry. You're not going to have some this hard on the exam. The point of this is, I want to go through probabilities a little bit, and I want to go through hashing and the math behind hashing. Because remembering that will be useful. OK, so now you said you're having trouble with this step? OK, so let's see. Let's do this here. So we have this table here, right? And we have n elements-- e1, e2, e3, all the way through en. How do we put them in the table? We hash each of them, and each of them maps to a random slot in the table. If these are the slots, then e1 might map here, e2 might map here, e3 might map here, e4 might map here, so on and so

forth. So I have arrows, right? Every time I do a hash, that's going to set something to a 1. The numbers don't necessarily map to different slots, because each number, on its own, maps to a random slot. So these are all going to be ones. And everything else becomes zero. If no number maps to a slot, it is 0. OK, let's look at one slot, any slot. So let's say I'm looking at this slot over here. Can you guys see, by the way? OK, so let's look at this guy here. What's the probability that it's a 0? So the probability that the slot is a 0 is the probability that the first number didn't map to it-- otherwise it would be a 1-- e1 didn't hash to that slot. e2 also couldn't match to that slot, right? So it's the probability that e1 didn't hash to the slot, and e2 didn't hash into slot, and e3 didn't hash into the slot, so on so forth, right? All the way up until en didn't hash to the slot. This makes sense?

Now these are all independent events, because all the hashes are independent, by the uniform hashing assumption. So then I can turn ands into products. So I can say that this equals to the probability that e1 didn't hash into the slot, times the probability that e2 didn't hash into the slot, times the probability that e3 didn't hash into the slot, so on and so forth, all the way to the probability that en didn't hash. So since I'm dealing with the same hash function, turns out that all the probabilities are the same. So there, the probability that some fixed number didn't hash, to the power n. So this is how I got from here to here. Probabilities and the properties of hashes and hashing assumptions. So you guys should have those on your cheat sheet, and maybe if you have time, review probabilities a bit.

AUDIENCE: What is the probability that any given one doesn't hash, 1/m? So if e1 doesn't hash in that spot, isn't that probability 1/m?

PROFESSOR: Not quite. You're close, but not quite. So you're saying that the probability that e1 doesn't hash to this slot is 1/m?

AUDIENCE: I guess it's 1 minus 1/m.

PROFESSOR: Exactly. The probability that it would hash here is 1/m, because it has to pick that one slot out of n possible slots. But if you're just saying, all I want is that it doesn't hash here, well, it means it can hash anywhere else. So it has m minus 1 options. It

can go to any of those m minus 1 places, just not to that one place. So m minus 1 over m.

AUDIENCE: It's interesting it went the other direction. Instead of saying, it's 1, it's 1 minus it. Wouldn't it have been just as easy to go the other direction, or no?

PROFESSOR: No. Not doing this makes the problem hard, so that's why we're doing it. This kind of flipping is easy to do conceptually, but it might make a hard problem into a really easy problem, or at least into a do-able problem.

AUDIENCE: Isn't it this the same thing? I guess maybe not totally.

PROFESSOR: So it is exactly the same in terms of the math, but computing this without turning it into this is really hard.

AUDIENCE: Any given slot is 1, isn't it kind of like what we just said, except if the probability of any one mapping is 1/m, mapping to a 1, right? And then you take 1 over m raised to the n, that's the probability of it being a 1 at that one place, right?

PROFESSOR: No, not quite. Yeah. OK, so are we getting this? Somewhat? Yes?

AUDIENCE: So the probability of a false positive, you're saying that's what's the probability that you get the 1, if you actually should [INAUDIBLE] the 0. It's because multiple things mapped to that one slot, right?

PROFESSOR: So the probability of a false positive is the probability that, given a new element, when we hash it we get the 1. The hash of that new element is independent of all the other hashes.

AUDIENCE: Then why is it simple in probability that you get the 1?

PROFESSOR: So if I have a new element, I'm going to compute its hash, and I'm going to look in the table. If I see a 1, I'm going to say, oh.

AUDIENCE: Oh, it's a new element. OK.

PROFESSOR: Yeah, so it's something that was not in the set.

AUDIENCE: OK.

PROFESSOR: Okay, cool. OK, so let's see if we can do a harder version of this. So this probability isn't great, but if we do one trick, we can make this really nice. And this puts together a trick called bloom filters that is used in all sorts of situations.

So for Bloom filters, we still have n elements, and we still have a table of m bits. What changes this time is instead of having one function, we have k hash functions. So when we take an element and insert it, we're taking element i. The way to insert it is we're going to compute its hash value using all the hash functions, and set all the corresponding bits to 1. So insert ei becomes, for j in 1 through k, the table bit corresponding to the hash function, j, of the element is 1. So each element sets k bits to 1. Now how do we check if an element is in the table?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Since, for every element, we set all the corresponding k bits to 1, now when we have a new element, we're going to compute to the k positions, and if any of them is a 0, then we couldn't have possibly put that in the table. So all T of h j of f have to be 1. So for every element, we hashed it k times, and set the corresponding bits. If we have a new element, and by hashing we get here and here, but we also get here, and this guy was a zero, we know we definitely didn't put this in. So now what's the probability of a false positive?

AUDIENCE: My first intuition is just raising that to a power.

AUDIENCE: The probability that when you check--

PROFESSOR: Oh, I forgot to say something, by the way. The k hash functions-- I think they satisfy simple uniform hashing. I'm not sure if that's the right thing, but they all have independent values from each other. So they're all independent. So for any number you give, any hash function returns a value that's independent of all the other hash functions, and they're all 0 through n minus 1.

AUDIENCE: Why is not that just raised to something? Because we know the probability-- OK, actually we need to recalculate that.

AUDIENCE: Because it's the probability that all of them are 1, even though you haven't hashed yet.

PROFESSOR: So the false positive, the probability of false positives is the probability that all the k slots that correspond to f are 1's, right? So, since the hash functions are all independent, this is the probability that one slot is the 1, raised to the power k. Right, because they're all independent slots. So it's the probability that one slot is a 1, raised to the power k.

OK, so now what's the probability that one slot is a 1? It looks a lot like this problem, right? Except there's a tweak. How many times did we put the 1 in the table? So here, we put a 1 in the table for every element. So we have n sets, right? So n times we're going to set t of something to 1. Right? For every element, we have one set. We set one bit to 1. It might have been said before-- that's something else. Yes?

AUDIENCE: So here it's raised to the m k?

PROFESSOR: Yeah, pretty much. So here, for every element we hash it through all the k functions, and set the corresponding bits to 1. So one element generates k set operations, and we have n elements, so we set n k bits to 1. Does this make sense?

AUDIENCE: Can two hash functions point to the same slot?

PROFESSOR: Sure. But they're all independent, and that's the only thing that matters. So every time we set the bit, which bit was set is independent of all the other bits we set, because all the hash functions are independent, and all the values are independent of each other. So this time, the table size is still m, so that didn't change. This time we set n bits to 1, this time we set n k bits to 1. So then the right thing to do is copy this answer, and replace n with n k. And if you have to write the proof, you'd copy-paste the proof and replace n with n k.

So this is 1 minus m minus 1, over m, times n k. And of course you should go

through the whole thing in your head and convince yourselves that this is true.

AUDIENCE: Does that say one of the elements is what? k, something?

AUDIENCE: Sets.

PROFESSOR: Bit sets. So one element sets k bits in the table, not necessarily different bits, just independent bits. So if you have n elements altogether, they set n times k bits. This thing gets run n times k times, whereas here, the set operation gets run n times in total. That's the difference in the two problems. Right here you have one function for each element, here you have k hash functions. This is hard, right? Well, it's the hardest hashing problem that I could think about and that makes us go through probabilities and through all the hash stuff. The problems on the exam will be easier, so one, don't panic. Two, review hashing, review probabilities. When I said, from the theory, this is what you get, if you didn't understand that then please review the theory.

AUDIENCE: Why is it raised to the k? Because we did down there, if we replace n with n k, then we'd just get everything except.

PROFESSOR: So this thing in here is the answer to the previous problem, except you take an n and you replace it with an n k. So this is the probability that one bit is set to 1. But here, when you're given an element, you're going to hash it through the k functions-- you take this guy-- you're going to hash it through the k functions, and you're going to check all the bits. So you're going to check k bits. So as long as any of the k bits is a zero, not a false positive. So we need all the k bits to be a 1.

AUDIENCE: Oh, I see.

AUDIENCE: What if the hash functions are dependent?

PROFESSOR: Then become intractable.

AUDIENCE: And what if they are? I think the in this problem, the way they are being hashed, that becomes dependent, because I think there were some problems where, if something is being hashed somewhere, then the probability-- there could be hash

functions that would put the other thing in the next slot.

PROFESSOR: Yes, so you want to reduce these problems to independent hashing. If you look at the proofs, all the proofs assume uniform hashing, simple uniform, whatever it takes to get the math down to independence. Because this is the only thing that we know how to solve with probabilities. If everything is independent, then things multiply and add up in the right places, and everything is easy. If things are dependent, then proofs become really, really hard. So whenever you have dependent things, you want to find a way to reduce that to independent things.

Is everyone tired, or do you guys really not like this problem? By the way, really cool trick-- so this turns out to be a lot better than that, and I think the optimal value of k is around square roots of log n. And that gives you some filters with a really low false positive rate.

AUDIENCE: What do you mean by optimal?

PROFESSOR: Minimize the false positives. So given n and m, pick a case so that this thing is minimized.

AUDIENCE: What was the answer again? Or actually, regardless of that, what's the percentage of false positives?

PROFESSOR: It depends on what your n and m are, right? The more bits you can afford

AUDIENCE: But if maximize your k, you said you came up with some k that's maximized

PROFESSOR: I think k is

AUDIENCE: Square root of log n.

AUDIENCE: So then if you use that.

PROFESSOR: Let's not do the math.

[LAUGHTER]

It's really, really good. So these are used for all sorts of practical problems, all the way from branch predictors in processors, to databases.

AUDIENCE: So is it better than 1%? Do you know that, at least?

PROFESSOR: Oh yeah, for practical uses, this gets you, I think to 1% of 1% of 1%. So usually, put a Bloom filter before a really expensive check, and the Bloom filter gets rid of most of the false positives. And then you have a few more where you do the more expensive check. Okay, does this make sense? Any questions?

AUDIENCE: Do you more optimal if you repeated this Bloom filter independently of the other one, with more hash functions in that memory structure?

PROFESSOR: I think doubling the memory size is better. So two filters is the same as having two n bits. I think doubling gives you better results, always.

OK, so general stuff. We're going to have a lot of conceptual questions, so please make sure, again, make sure that for everything that we did, go through the problem. Understand the problem, know that there is a solution. Know the running time, maybe know how to implement the solution. Don't worry so much about the proof. We're going to have some problems where you have to come up with new things on your own, so get a good night's sleep before the exam. Really, if you have five hours left, then you have to choose between sleeping five hours or reading notes for five hours--

AUDIENCE: Drink caffeine.

PROFESSOR: It's not going to help, so caffeine actually helps you stay up, but it decreases your performance. And so if you're on caffeine, you're not going to think. You can regurgitate stuff, but you can't think. So caffeinating yourself is a--

AUDIENCE: I thought it was like it gives you concentration.

PROFESSOR: So there's an optimum amount of sleep and caffeine combination. If you don't sleep and caffeinate yourself, I guarantee that you will not solve any of the problems that require new algorithms.

AUDIENCE: Caffeine just squirts adrenaline in your brain. It doesn't do anything else.

PROFESSOR: So the thing is the memory is going to be better. If all you're doing is memorization stuff, then it's going to be better. So you're going to do well on the pattern matching stuff. But when your brain is panicking, you're not going to come up with new solutions, right? Usually, you have a problem, a hard problem. You're thinking about it, and then at some point when you're relaxed, like when you're in the shower or when you wake up you're like, crap, I found a solution. So the brain finds solutions when it's relaxed, not when it's like, holy shit, holy shit, holy shit. And adrenaline gets it in that mood. That's what it does. And that's what caffeine does in the end. So a little bit of caffeine might help you get up and get you running, but don't caffeinate yourself to not sleep the entire night. That's probably going to make you bomb the hard questions. Good luck on Friday. Eat candy.