PROFESSOR: All right, so today we're going to think about the-- well, also the low end of the polynomial hierarchy. So most of this class is about polynomial versus not polynomial and various notions of hardness, [? NP ?] hardness, and worse. We look briefly at P-completeness, which is about parallel computing.

Today we're going to be thinking about regular good old sequential computing, but trying to distinguish linear time versus non-linear time, and in particular, trying to find problems that are quadratic or cubic in n for sequential running times. And probably the most popular attack on this is called 3SUM, which is the following problem. You're given n integers. Let's say, and you want to know do any three of them sum to 0?

So of course, you can solve this in cubic time by testing all triples, whether they sum to 0. But you can also solve it in quadratic time. So it's not an algorithms class. I won't ask you to come up the algorithm. But they're quite easy.

First order n squared randomized is really easy. You take all pairwise sums. So first of all you, build a dictionary, a hash table of all the integers. And then you look at all pairwise sums. For each pairwise sum, you see whether the negation is in the hash table in constant time. So that gives you n squared randomized, because if a plus b plus c equals 0 is the same thing as a plus b equal minus c. So you look at all pairwise sums, see whether the negation is in the list of integers.

OK, you can also do order n squared deterministic. This a more fun puzzle. But I'll spoil the answer for you just to give you intuition for why this problem is n squared.

For every possible target sum minus c-- so that's going to happen n times-- I'm going to run the following linear time algorithm. So this is just two copies of the

integers in sorted order. I have n log n time to sort. So that's no problem.

I'm going to start from with my left finger here and my right finger here and look at the sum of those two numbers. If it's too big-- I have a particular targets, negative c, in mind-- if the sum of these two numbers is smaller then negative c, then I'm going to advance this one, because if this is in sorted order that will make my sum larger. If the sum is too big, I will advance this one backwards.

So in general, I have my right finger advancing left or my left finger advancing right. In each step, one of the two advances. And this will not miss-- and this will tell you whether that target sum is among the pairwise sums from this thing. In linear time, we do that n times, once for each target sum. So that's the fancy quadratic algorithm. No fancy data structures required.

Cool. So 3SUM is quadratic. And the big conjecture is that you can't solve it any faster. Well, you can. So it's not quite the conjecture. The conjecture is that there is no n to the 2 minus epsilon algorithm in general-- in the worst case.

Let me tell you before we get to-- and this is has led to a whole world of lower bounds of 3SUM hardness. If your problem is 3SUM hard, then you expect it also has no n to the 2 minus epsilon algorithm, because if it did, 3SUM would. And people generally believe that's the case for 3SUM. But there are some exceptions.

So one thing is that the numbers have to be fairly large. If all of the integers are in the range, let's say, minus u to u-- that's the universe size-- then via FFT, you can solve the problem in u log u time plus linear time. So your numbers better be at least larger than n squared. And in fact, we'll see n cube suffices. So they don't have to be huge, but they do have to be bigger than linear or quadratic.

More generally, there are a bunch of sub-quadratic, but only slightly sub-quadratic, algorithms. The first one achieves a roughly log squared savings. So a little bit less is a log log squared in the denominator.

This is a randomized algorithm in a model of computation called the word RAM. So if you're interested in the word RAM, you should take advanced data structures. But

basically you can manipulate, let's say, log n bit words in constant time. You can add them together, multiply them, that sort of thing. And you assume that the numbers you're dealing with fit in a word. Because if you're going to compare them, you'd also need that assumption.

So that's a reasonable model. And it essentially affords a kind of logarithmic amount of parallelism. And so because it's a quadratic problem, roughly speaking you get a quadratic amount in the parallelism of the model. So it's a bit improved. This is by two former MIT students, [? Eli ?] Barron and Mihai Petrescu and myself.

And very recently, this year, there's been another nice improvement. It's actually three algorithms, depending on your model. But all based on a similar idea.

Did I get these backwards? I think so.

So first, these two results-- so these are results by [? Gourmand ?] and Petty. So that's Petty, gave a talk here about it recently. In a real RAM model of computation-- this is a weaker model of computation, so the result is stronger.

In a real RAM, you still assume the numbers you're given, you can add them. I think actually all it needs is the ability to add them and compare them, maybe subtract-- yeah, also subtract. But no multiplication is really useful in that particular model. Because you can't extract bits out of the thing, so you don't assume that they're integers, you just treat them as real numbers. And all you know how to do is add a bunch of them and compare those additions.

So that's a weaker model of computation. And still they're able to get a roughly logarithmic improvement, not quite as strong as the quadratic analog. But one advance is that this is the first deterministic algorithm to be n squared. So randomization isn't necessary to achieve that. Though this is 2/3 power.

But the other advance is that you don't need to manipulate the individual bits. So even in the randomized model that's nice. And then the major thing, and sort of the first thing to call into question the 3SUM conjecture, which is that conjecture, is I

wouldn't really call this an algorithm. But it's a thing which runs n roughly n to the 1.5 time.

But here it's a more powerful model, a super powerful model, called the decision tree model, where the idea is that an algorithm is specified by an entire tree. The depth of the tree is this big. Each node of the tree says, add these five things, compare them to these five things, and see which is bigger, and then branch. There's a left branch, and a right tree.

If you've ever seen a comparison tree, same thing. But the comparisons are more interesting.

But this is not an algorithm, because we don't know how to compute that tree efficiently. We can compute it in probably polynomial time, but we don't know how to compute it in sub-quadratic time or sub-this time. So it's kind of a frustrating situation, because we know that this thing is sort of out there, but actually finding it is hard. And actually several problems in computers since the '80s, I think, where we know better decision trees than we know algorithms.

So my sense would be that decision tree model is strictly more powerful. The 3SUM conjecture is true for regular algorithms if you define it this way or this way. But in the decision to model, obviously, you can do a lot better. Question?

AUDIENCE: What's the reason to believe the decision tree model is some bit that we run polynomial algorithm, look at the bit and it tells us the incident?

PROFESSOR: You mean, why would--

AUDIENCE: Why would you believe this decision tree is there, like a model of computation that we should care about?

PROFESSOR: Oh, no, you shouldn't. Decision tree is not a model you should consider a reasonable computer. But it's interesting in that it suggests-- it gives you this tantalizing feeling that maybe you could turn this into a real algorithm, you could run a computer. But, definitely, yeah, you cannot-- if you don't know what decision tree

is you can't run it on a computer directly. So it's not a model of computation in the strict sense.

It's especially interesting-- I mean, it's always important to see how the model of competition relates to bounds, particularly if you're going to try to prove the lower bound. There are some lower bounds of n squared in restricted forms of the decision tree model. This says you can't extend those lower bounds to arbitrary decision trees.

Decision trees are traditionally used as a lower bound model. If you can prove a lower bound there, because it's a very powerful model, that implies lower bounds in something like the real RAM. So that's why people care in some sense. This says you can't prove a strong lower bound in that model, which is annoying. Another question?

AUDIENCE: What is the tree that they found the structure and the constants of the nodes, does it depend on the values of the integer? Only the number?

PROFESSOR: I'm pretty sure in this model you have a constant number of original integers. You add them together and compare them to a constant number of original integers.

AUDIENCE: But what was is-- I mean I can compute the tree in sum, but what does it depend on? Like if I change the value--

PROFESSOR: The tree, of course, has exponential size. So you never would actually want to compute it explicitly. What you want to compute is after I've done some number of things, what's the next operation that happens.

AUDIENCE: That's still not my question. So my question is what changes the actual structure of the tree? Is it just the number--

PROFESSOR: Oh, yeah, n. The decision tree only depends on n. I mean in some sense the decision tree's encoding an adaptive algorithm, that depending on the results of previous comparisons tells you what to do next about. If you think of it as the entire tree that's only depending on n. But because it's so big, I mean, that doesn't help us

if. You could imagine pre-computing for n, but there's no way to store it. And you couldn't really afford that exponential time.

OK, so that's a short story about the known upper bounds and also the known lower bounds on 3SUM. There are some weak lower bounds in a particular version of the decision tree model when you can only compare I think sums of two items, then you can get an n squared lower bound. But that's not especially interesting given this result anymore.

Let me tell you briefly about k sum, which is the obvious generalization, instead of 3, do any k of them sum to 0? Here, they're actually stronger and lower bounds. So if 3SUM is the most popular thing considered for proving quadratic lower bounds, because a lot of problems we care about are linear or quadratic, so 3SUM gets a lot of the attention.

K sum is a little easier to argue about. In particular, it's NP hard in general, right. This, in particular, encodes something like partition. If you have n integers, and they're overall sum is 0, and you want to know whether any n/2 of them sum to 0 or something like that, that would be roughly partition.

So this is NP hard. So definitely it's got to get hard for some k. In fact, you can show fixed parameter hardness, W1 hardness, with respect to k. So in particular, if you assume the exponential time hypothesis, then you get some lower bounds. And the best lower bound known so far is that there's no n to the little of k algorithm, assuming regular ETH.

For this, we need to assume k is less than or equal to-- is not too giant, because, I guess, if k equals n, for example, this problem is really easy. So it can't go all the way.

So this says, well, maybe we don't get the constant right, but there's some kind of n to the roughly k dependence. So there's a reason that there's a number here larger than 1. Although we don't know how to prove that, the feeling is that 3SUM, k sum, they require roughly n to some constant times k. You can debate about what the

constant is, but we have this theorem.

And on the upper bound side-- and what people believe is the right answer-- is k/2 ceiling-- at least randomized. If you want deterministic, you might get a log factor. But you can definitely achieve this by the same kind of do all k/2y sums twice, and then look for collisions in the hash table.

And so the ceiling is what's making 3SUM into a quadratic thing. But 4 sum is just as easy as 3SUM, because you can still solve it in quadratic time. But 5 sum, the conjecture is that requires n cube time.

AUDIENCE: Sorry, I didn't quite catch that. Is that a known result?

PROFESSOR: This is an upper bound is known. And then the conjecture is that that's tight. There's no end to the ceiling k/2 minus epsilon algorithm. That's what we don't know. But algorithm is easy.

So that's k sum. And this gives you some more intuition for why you should expect these problems are hard. In particular, if you could prove k sum requires-- I mean, if you can prove this conjecture, you prove the exponential time hypothesis. So you prove p does not equal NP. And you may make $1 million and lots of good things happen. So we should try to do this.

Of course, as I mentioned, 3SUM is the one we use the most in this world, because for NP hard problems, usually we use NP hardness and all the stuff we did. You could use k sum, but that's basically partition. But this is sort of motivation for why you should think 3SUM is hard. I'll leave it at that.

So let me talk about 3SUM hardness. So I'm going to call a problem 3SUM hard if that algorithm has an n to the two minus epsilon time algorithm, then so does 3SUM. OK, this is what we want. If I say problem is 3SUM is hard, it means it shouldn't be solvable in less than quadratic time other than this poly log stuff.

So this is the formal meaning. If you could solve it in sub-quadratic, truly sub-quadratic time this is often called the minus epsilon, then 3SUM can be solved in

truly sub-quadratic time, contradicting the 3SUM conjecture. So if you believe this is the 3SUM conjecture that means this is not possible for your problem. And the way we're going to do that usually is with a 3SUM reduction.

There are other ways to do it. You don't have to follow this particular style reduction. But most of them do.

So it's going to be a multi call reduction, but in this world, we have to be careful about polynomial factors. We don't want to call your thing n times and say that was a legitimate reduction. So let's say you can call-- if you're reducing from a to b, then that means you could solve b using a. And you're going to make a constant number of calls to a. Sorry, other way around. That means I can solve a using b.

Usually, we take an instance here, reduce it to an instance here. That's OK. But because we're going to want to solve not just decision problems, we're going to say, OK, you take your instance of a. You can call an oracle for solving b a constant number of times, as long as the thing you call it with is also not much bigger. So the n prime that you call this thing with should be linear in n.

And the running time of the reduction should be sub-quadratic. OK, pretty much all reductions, it's like n, n log n, maybe n log squared n. But it should be strictly less than n squared, otherwise the reduction doesn't tell you much about weather the problem is quadratic or not.

So with this much running time, plausibly you could construct a larger instance. But this constraint says the instance of b that you called should be linear in size, so the quadratic over here is the same thing as quadratic over here. OK, so those are the rules of the game. We're not going to have to worry about these constraints too much. Most of our reductions are constant factor blow up and run in a reasonable amount of time, But we've got to be a little careful here to make sure the running time is not huge. Usually we're allowed polynomial time.

OK, so if you have a 3SUM reduction from a to b, and you know a is 3SUM hard, then b is 3SUM hard.

AUDIENCE: [INAUDIBLE]

PROFESSOR: N prime is the size of the thing that you're-- the instance you're calling with. So if you have an instance x over here, you have some x prime over here. N prime is the size of x prime. There's a constant number of them. But I want all of those instances to be linear size.

OK, so initially a is going to be 3SUM. 3SUM is 3SUM hard. Because if it as a sub-quadratic algorithm, then so does itself. And so that's actually easy. NP harnessed, that's not so easy. If we let b be some other problem, then we'll prove hardness.

In this world, because we don't have any solid, lower bounds to work from, it's also interesting to go in both directions. I'm not going to define a notion of completeness here, just because it hasn't been done. But you could define 3SUM completeness to mean you can reduce from 3SUM and you can reduce to 3SUM. We'll see a few problems in that-- usually people call that equivalence, sub-quadratic equivalence.

OK, so let me start with some base 3SUM hard problems to start from. A lot of this comes from a paper-- this paper-- by Gajentaan and Overmars, 1995. And they had been collecting over the years a whole bunch of mostly computational geometry problems, which are 3SUM hard in that you can reduce 3SUM to all of them.

But in the center here is a bunch of core problems, 3SUM, 3SUM prime, which I would call ABC version of 3SUM-- we've seen a few ABC problems in the past-- and a geometric version of 3SUM. So let me tell you about those.

First of all, 3SUM is 3SUM hard, even when u is order n cubed. So that's nice to know. The integers you're working with don't have to be giant. This is based on a hashing argument, which we won't go into.

OK, so what is 3SUM prime? 3SUM prime you're given three sets of integers, A, B, C-- yeah?

AUDIENCE: For these reductions we're using only deterministic reductions?

PROFESSOR: Let's say we're only using deterministic reductions, although randomized would also be interesting, You just have to weaken this statement. But here I'll say deterministic.

AUDIENCE: Is the hashing argument for why [INAUDIBLE]?

PROFESSOR: Yeah, I'm pretty sure you can derandomize that hashing scheme. I need to double check, but, yeah, that's the claim. Yeah, that is good question.

So this is the ABC version of 3SUM. We just want the three items to come from three particular sets. And traditionally, this one is phrased as a plus b equals c, although you could also say a plus b plus c equals 0. It's the same thing. You're just negating all the c's. And in this world, because you have one item from each set, actually it's really easy to just negate one subset of them. So it doesn't matter whether you put a minus sign here or not. But I will not, because that's how 3SUM prime is usually defined.

So I claim 3SUM prime is 3SUM hard. Why? I let capital A-- if I'm given a 3SUM instance, let's call it S. S is a set of n integers. I'm going to let a equal S. I'm going to let b equal S. I'm going to let c equal negative s. Done. OK, so that's a reduction from 3SUM.

AUDIENCE: [INAUDIBLE] like if 0 is in your list, the way the instance you constructed, you might use some element x of the list from a. The same element x from b, and then y from c. And x plus x might be, I guess, plus y would equal 0. In the original list you didn't have x twice. I'm just getting the example of choosing 0 three times as sort of a convenient example of that.

PROFESSOR: So that you also detect a linear time.

AUDIENCE: What do you mean?

PROFESSOR: You can detect whether there are any such triples. If you allow repetition, then you answer the question. Your goal is to solve 3SUM.

AUDIENCE: So you're solving 3SUM by making oracle calls to 3SUM prime?

PROFESSOR: Right, so if you can 3SUM ahead of time, you're done. In linear time, you can check whether there are any pairs that allow, with some duplication, a solution to 3SUM instance. This is not addressed in the paper, which makes me think that in this definition of 3SUM, we allow the items to-- basically, every item could be used three times, up to three times. There's no requirement that they're distinct items. So then this reduction is fine.

I don't think that's a big deal. And you can get rid of it. But that must be how it's normally defined. I didn't specify whether it was three distinct items. But let's allow multiplicity there. And then this reduction is fine, because that's what the paper does.

OK, with more effort, like adding big integers and so on, you can reduce-- in the other direction reduce from 3SUM prime to 3SUM. I won't cover that, because I just want to prove the 3SUM hardness about things. But in fact, all these three problems are identical to each other. If any one of them is sub-quadratic, then they all are. So that's nice, because 3SUM prime is really a special case of 3SUM.

So our next problem is the geometric problem. They call it geometric base problem.

So here we're in 2D. And we're given endpoints whose y-coordinates are all 0, 1, or 2. You can imagine why-- because there's three lists. And so they all live on three horizontal lines. Here are the points.

And we want to know is there a non-horizontal line that passes through three points like this. OK, so our claim, GeomBase is 3SUM hard. And this is their proof.

On the first line, we put A. On the last line, we put B. And in the middle line, we put every item in C divided by 2.

So if you look at-- sorry, this is a reduction from 3SUM prime. So I have three integers. We want to know whether you can every a plus b equals c. So the idea is if I have two items, A and B, then this point-- I mean, if I just draw the line and intersect it with the y' equals 1 line, that point will be the average of little a and little

b-- so a plus b over 2.

So if there's an item C that matches a plus b, then the C/2 will equal the a plus b over 2. So there's going to be a line through three points, if and only if, 3SUM prime had a yes answer. And you can reduce in the reverse direction-- in fact, just like this. You just multiply all these coordinates by 2. That gives you C.

So those are our starting points. And we're going to use all of them. And I'm just going to run through a bunch of examples of 3SUM hard problems. So all of them shouldn't have sub-quadratic time algorithms unless 3SUM does.

So the obvious starting point here is what's called degeneracy testing in computational geometry. So usually, we like to assume that you have endpoints in the plane. They're in general position, meaning no three are co-linear. So the problem is given endpoints, are any three of them co-linear? Question?

AUDIENCE: With 3SUM prime, because those are integers, how do you deal with them that way?

PROFESSOR: Oh, integers and rational, same thing, you just scale it, multiply. So everything here-- because we're going to go into geometry quite, I will use rationals quite a bit. So I can multiply everything by 2 to make it integers again.

But this problem does not say integers, so that's why I'm allowed to do that. I start with integers. And then I do this. But you can also add integers here. It wouldn't make a big difference.

OK, so given endpoints in the plane, are any three co-linear? I'm guessing this is the original motivation for defining 3SUM. This is really a harder version of the problem. This is kind of a special case.

But in particular, it's not exactly the same, because we forbid horizontal lines. We had to construct a very degenerate instance with lots of points on the horizontal lines in order for this correspondence to work. So the question is can you make something that is only degenerate, only has three points co-linear, when the 3SUM

instance has a solution?

And this reduction is a little unsatisfying. And I don't have a great intuition for it. But it's very simple.

We're going to take-- this is going to be a reduction from regular old 3SUM, not 3SUM prime. So every number x, we're going to map to the point x comma x cubed. Cubed because it's odd and not 1 basically.

And so we take our x values-- probably not a good idea to put 0 in there, but whatever. And we just project them onto this x cubed curve, x 3 is odd, so it has this nice picture. And the claim is if you take any two points here-- so here's an x-coordinate 1/4 and 3/4, and, of course, they would actually be integers. That's OK. You can scale.

Then so the sum of those is 1. And if you look at negative 1, that will-- the cube of negative 1, which is 1, is exactly equal to where these two cubed points would hit if you extend the line. Yes?

AUDIENCE: Now we have the problem that if you want to use the version of 3SUM here, [INAUDIBLE].

PROFESSOR: Yep, so we definitely need that those guys are distinct. I'm sure that those two versions of 3SUM are equivalent up to sub-quadratic reductions. But I don't see how to prove that off hand.

OK, cool, now why is this true? I've checked it. It's true. Off the page of a algebra and prove it. I don't have a great intuition for why this is true. But there you go. It's easy enough to check where this line should go. And it happens to go exactly to the place where the sum goes. So sorry.

I'm guessing it would also work for x to the fifth. But I didn't check that.

OK so those three points on the line.

OK, so an important life lesson about geometry is something called duality. So here

we're interested whether there was one line that goes through three points. A complimentary problem is I give you a bunch of lines, do any three of them pass through a common point? Is there one intersection between three or more lines? Is there a point that is on three lines?

If you know projective geometry, this is totally obvious. It's the same problem as this. You just apply duality.

Now, there are many different dualities. I'll give you two today. First, my favorite is projected duality and the sort of most standard, at least, in math. If you have a point with x-coordinate a and y-coordinate b, you map that to the line ax plus by plus 1 equals 0. And vice versa.

So if I have a line-- almost every line can be written this way. Everyone line that does not go through the origin can be written like this. And then you can convert it into the corresponding point. So if you give me a bunch of lines, just translate so that none of them go through the origin. And then convert into corresponding set of points.

And the nice thing about this duality is it preserves incidence, meaning if before I apply duality, I have a point and a line that are touching, then after apply duality, I will have a line and appointed are touching. So that's great, because in particular, a three-way intersection or a point is on three lines will convert into a line that it goes through three points. And so we get a reduction from here to here.

That's kind of like magic. But it works, essentially because-- well, if you think of a line over here as just a pair of coordinates, usually written a, b, then we're just taking essentially a dot product between those two things. It's a plus 1. But it doesn't matter which one was the point and which one was the line. So that's very convenient. And you can use that to convert a lot of line problems into point problems.

I won't mention k sum very much. But obviously, the d dimensional versions here are d plus 1 sum hard. So that's these are sort of the more geometric versions of k

sum.

So let's do some more problems. Next one's called a separator.

So let's say we're given n line segments in the plane, is there a line that separates them into any two non-empty groups?

And that line is not allowed to intersect any of the segments. So you're not allowed to split a line segment into two parts. You just want to partition the line segments into a left chunk and a right chunk.

So there's actually two versions of this problem. The first version allows half infinite rays as segments. And then you can assume that all the segments are horizontal. So I think that pretty clearly expresses it.

But we can in particular think that we are reducing from GeomBase. We had this setup. We have points on three lines. We want to know whether there's a line that passes through them. So I'm just going to take the complement essentially of those lines, emit tiny intervals wherever I had points before. And now there will be a separating line, if and only if, the original points have a line through them.

If you make these tiny enough, you won't be able to do anything else-- oops, yeah-- or you could just split them into 1/3, 2/3, yeah. I definitely need here that it's a non-horizontal line. Thanks.

Now this requires having these half infinite rays. Otherwise you could make a line like this. So you're not allowed to do that if these goes off to infinity, then you'd be cutting those rays.

OK, so maybe you consider that reasonable. Maybe not. Depends on the application. If you don't consider half infinite things reasonable, you can replace them with some vertical segments. You build this little box, essentially a box like a pinwheel pattern. So there's no way to cut it up except to go through the center.

So two versions-- version one, we allow half infinite things. And every segment is horizontal. Version two, horizontal and vertical segments are all finite length. We'll

use this version to reduce from a bunch of times. Or we will follow this kind of reduction essentially. OK, next problem.

OK, next problem is called strips cover box. I like these names of problems. They're pretty clear.

In fact, they're so clear, here's a figure. We have a box, which means axes align rectangle. And I have strips. Strips are-- I take two parallel lines and take the lines in between them. All these parallel lines between here and here-- that's a strip.

So I'm given n strips. I'm given a box. I want to know whether there's an empty part or whether it covers.

AUDIENCE: Can you [? use ?] angle or just strips?

PROFESSOR: Oh, yeah, sorry, the strips are placed. I give you two lines for each strip. And I mean the region in between them. They're on the plane. You can't slide them around. That would be a coverage problem. It's pretty easy. I just want to compute whether there's any point in here is not hit by any of the given strips.

I should mention, all of these problems, except where I say otherwise, can be solved in quadratic time. And so this is showing that that's essentially tight. So you can solve this problem by computing the arrangement of these lines in quadratic time and checking all the cells, whether they're in all the strips. So that's not hard. But the claim is you can't do any better than n squared if you believe the 3SUM conjecture.

And the reduction is essentially this. But I'm going to modify it a little bit. So remember, I rotated this 90 degrees for a reason, because I wanted to use a particular kind of duality.

The construction is going to be the dual of this. But remember, here, the goal is to find the line that does not hit any of these segments. And the segments are vertical. Or they might be infinite, half infinite rays.

OK, so I'm going to start from there. And then I'm going to dualize, using a different dualization. This is probably the most popular one in computational geometry. I think because everyone remembers y equals mx plus b. And so there's the obvious conversion between a point, which has b and m, to a line, which is mx plus b. You could argue about which is which, but I think this is the more common.

So what this means is I start with a point. The y-coordinate determines the slope of my line. And the x-coordinate determines the y-intercept of my line. It's b.

And you can also convert in the other direction. I don't think we'll need to be here. And that will work for all non-vertical lines. This will not represent vertical lines.

In case you're curious, if you want vertical lines here or if you want lines going through the origin here, you need points in infinity. That's the projective thing here. But we don't need that here.

Because we're just going to take these points and convert each of them to corresponding lines. So this is a segment, I'm going to get an infinite number of points-- sorry, there's an infinite number of points here. So I'm going to convert into an infinite number of lines. That's actually OK, because these points all I have the same-- sorry-- opposite.

I really want the x-coordinate to m and the y-coordinate to be b for this picture to be the right picture. So all these points have the same x-coordinate. So when I convert them into lines, they all have the same slope. And they'll also be right next to each other. Namely, they will be a strip. Isn't that cool?

So when you do this dualization, a vertical segment becomes a strip. I think the fancy word would be here you have a pencil of points. And you convert that into a pencil of parallel lines. A pencil of parallel lines is strip. Pencil just means like a continuous family.

Now here, these guys are infinite. So it's going to be a strip that it goes off to infinity on one end. That's a half plane.

So if we have a ray, vertical ray, that's going to convert into half plane. Half planes aren't allowed, so we're going to have to do something with them. But there's only six of them. There's three down here and three up here.

And we also haven't defined what our target rectangle is. But at this point what we would like to say-- so let's see, what would it correspond to a line here? Notice, the line will never be vertical. What would be a line that happens not to hit any of these things? In the dual, that line maps to a point.

And so that's saying that there is a point that is not covered by any of these strips or half planes. So we want to know whether the union of these things is the entire plane. So it's not quite the problem we wanted to reduce do. But it's not hard to fix it.

We essentially just need to make a retake a really big rectangle. And then there'll be an empty point in there, if and only if, there was a line in this problem.

How big does the rectangle have to be? Well, conveniently, these half planes essentially narrows down to a hexagon. So you have six of them. It might be less than a hexagon. But I'll just draw six.

And we're saying all of this stuff is covered. All the things outside the hexagon are covered by those half planes. So really it's just a matter whether this has any empty points. So take the bounding box of that hexagon. That's my box.

And now I don't have to worry about half planes anymore. I can restrict them. I can just say, oh, well, now this is a strip, which covers-- in particular, I need to cover this part of the rectangle. I no longer need to go off to infinity.

So now, I have a bunch of finite strips and a finite box. And it's just a matter of whether that thing has any empty parts. Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yes, there will be an empty point, if and only if the original thing had a yes answer. So they will cover, if and only if, if you have a no answer. Any questions?

AUDIENCE: Just nomenclature-- can have word [INAUDIBLE] that thing you just put on the right, because it's actually a special case of what you call [INAUDIBLE].

PROFESSOR: Yeah, right, so what they say is this a reduction from GeomBase, mimicking the proof of separator 1. Yeah, I don't have a name for it. Sorry. Yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yes, this duality also preserves incidence. I think it's a slightly perturbed version of the regular projector duality. It is also a projected duality in a sense, but with an extra Mobius transformation thrown in or something. But those also preserve incidence. Yeah, we obviously need that.

OK so to make this a little more usable-- I mean, strips can be nice. We'll use it in some situations. But geometers tend to like to think about triangles. So we can also convert this into whether bunch of triangles cover a given triangle.

Basically, so here we are going to reduce from strips covering a box. We start with a box. And we're going to convert that-- we'll draw on top of it so it's a little clearer-- I'm going to take a really big triangle, which contains that box. But then we'll triangulate the exterior right here. And add those triangles to my covering collection.

So all of this stuff is covered for free. And so now what remains-- in order to cover this triangle, I just need to cover this box. OK so, that's one part of it.

And then the other thing is that we're given strips. So if I have a strip-- actually, do I have a figure for this? No-- if I have a strip in the original problem, which looks something like this, I really only care about the portion of a strip that hits the box here. So I will just triangulate that part and say those triangles are in my set.

And then those triangles are going to cover this triangle. The red triangles will cover the red triangle, the big red triangle, if and only if the white strips cover the white rectangle. Pretty easy-- all we need is that these have constant complexity so we're not blowing up more than a constant factor.

Note here all of these smaller triangles are contained inside the big triangle. So you

can even assume that these guys are contained in this guy. We'll use that at some point-- possibly very soon. So next problem.

Hole in union-- I give you a bunch of triangles. I take their union. I want to know whether that's a simply connected polygon or whether it has a hole in the center.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Almost the same as this problem. I do is to do a little bit of work, because maybe you'd-- actually, pretty much that reduction will work fine, as long as the outer triangle is strictly bigger than the strip, then I'll always have these outer red things, which make a region. And then there'll be a hole in there, if and only if the rectangle is not fully covered.

So I just did to enlarge that outer triangle slightly. Then I have proof that this is 3SUM hard. Done.

Here, we're also using that the red triangles are contained inside the big red triangle. We don't go outside and possibly make a hole in some other way.

OK, another easy one-- triangle measure. I give you a bunch of triangles. What is the area of their union? Well, it's going to be the area of the big triangle, if and only if the big triangle's covered. So this is a reduction from triangle covers triangle. OK, easy.

Here's a somewhat different problem-- point covering. So here I'm given a bunch of half planes, n of them. I want to know is there a k-way intersection?

You can think of this as a version of two-dimensional linear programming. So this is all in 2D. You're given a bunch of linear inequalities, which are half planes. You want to know-- not can I satisfy all them, maybe that's not possible, but can I satisfy at least k of them? So this is for approximating linear programming.

There a lot of algorithms for doing that. You can do this in quadratic time. But the claim is you can't do it better unless if you believe the 3SUM conjecture.

OK, so this is-- I don't have a figure-- no.

So we're going to reduce from strips cover box. So we're given a bunch-- this figure again-- we're given a bunch of strips. We're given a rectangle. We want to convert that whether the whole thing intersects to whether there's a k-way intersection between infinite strips on one side, half planes.

So here's what I'm going to do. If I have a strip-- I should maybe really draw it this way-- bunch of parallel lines, finite segment of them. But the lines are infinite in this is direction and this direction. I'm going to convert that into the complement, a common trick here.

So we have half plane over here. And we have a half plane earlier. Obviously, you cannot be in both of these at once, because they're disjoint. So the best you can hope for is to be in one of them, which means you're not here.

And remember, the whole question here is whether there's a point that is not in any of the strips. So you're going to get one point-- you're going to get a score of 1 if you're here or here. You're going to get a score of 0 locally if in the region that's already covered.

Now we also need to represent the rectangle in some way. So if I'm given a box. Draw this line, this line, this line, and this line. And the half planes I want are the ones that contain the rectangle.

So the idea is you're going to get four bonus points if you're in the rectangle. And now the question is-- I'm going to set k to be n plus 4. I want to know are there any points that are in n plus 4 of the half planes. To do that, you have to be in all four of these-- in other words, in the box-- and in one of these for every strip, because you can't be in both.

It's just to achieve that score of n plus 4. So that means you're exterior to all the strips, but inside the rectangle. So that's the same problem. Cool.

Next problem is a visibility problem. So we've got a couple of visibility problems.

Both of these problems are about something called weak visibility. If you have two geometric objects, a and b, they are strongly visible to each other, if every point over here can see every point over here, meaning the visibility line doesn't cross anything else. But what we're talking about here's weak visibility, which says there's some point over here which can see some point over here.

So I want to know, say, given two segments, whether there's some point here, some point here, where if I draw the connecting visibility line, there's no other segment blocking it. So this is a. And this is b. This would be an example where a and b-- well, they do weakly see each other, because there's that pair.

So given n segments, you can construct what's called a visibility graph, which is all of these things in quadratic time. If I want to know whether this segment can see another segment in this weak sense, that's 3SUM hard. Here's the proof.

It's exactly the same thing, just add two segments. This segment can see that segment, if and only if there's a line. There are so many fun little things you can do. But these are problems that a lot of people have thought about. And they're always wondering, can we do better than quadratic? Now we know they're all sort of in the same bucket-- almost. OK, here's another problem.

So the problem is I give you a bunch of triangles in 3D. And let's see, and maybe I give you a point up here. And I want to know whether that point can weakly see a given triangle. And here all the triangles are horizontal. Make it nice and simple. And it is possible to solve this, I think, in n squared log n time-- maybe n squared time. You can construct the visibility graph here in n squared, because they're all horizontal.

So I'm given this point. I want to know can it see any of t. Or is it completely blocked by these triangles? Sound familiar?

If we put that point way up near or at infinity, depending on what you allow me-- near infinity will be enough-- this will essentially be orthographic projection of these triangles onto this triangle. And it's a question whether these triangles cover this

triangle. So this is a reduction from triangles cover triangle to visible triangle. Pretty easy, just put all these guys really close, slightly different z-coordinates so they're not overlapping, and put the point far away. So visible triangle is 3SUM hard.

All right, let's do some motion planning, robot motion planning-- first in 2D-- so planar motion planning. I give you a segment. That's the robot. And I have-- so this is the special arm, the robot-- then I have various obstacles, which the robot is not allowed to penetrate. I give you a starting position for the robot. And I give you a target position for the robot.

And want to know can I go from here to here by some motion? And motion we'd allow rotations and translations. So can I slide this robot somehow through this obstacle course, in here, just some parallel parking, whatever. You can do lots of tricks to get from A to B. This problem can be solved in quadratic time. Most 2D motion planning problems with just rotation and translation can be sold in n squared time.

And that's tight, because of this. We build these frames, which are large enough that it doesn't concern the robot. The robot's big enough that it's going to have to simultaneously pierce all three lines here. And we're done. OK, great. All this build up for very simple proofs. Cool, one more.

AUDIENCE: Is that one actually quadratic time?

PROFESSOR: Yeah, you can solve this in quadratic time. Cubic is obvious. There are three degrees of freedom. But I think, in fact, quadratic time. That's claim in the paper. I haven't studied motion planning algorithms for a while. So I don't know exactly how it goes. But in general, constant dimensional motion planning with constant numbers of objects can solved in polynomial time. But you can debate about the constants, which do matter here. I think the claim is tightness for that one.

Here's another one which can be solved in-- their claim here, they say n squared log n they say. We can probably get n squared, but n squared log n is enough. It's a particular version of 3D motion planning.

So we are given a vertical segment. That's our robot. We're going to have triangles. Those are our obstacles, because triangles we can make polyhedra. In this case, all the triangles will lie in horizontal planes. And the segment position will be vertical. And here you're only allowed translation.

With translation and rotation, you could also do this. But they restrict to translation only, because there they can get a almost quadratic time algorithm. With rotation you have to add a couple of factors of n probably. But this version, they can solve in n squared log n.

And it's 3SUM hard by similar kind of structure to the triangles covering triangle. Mainly, we build this cage-- the triangles aren't shaded in. But there's basically a triangle of triangles here, and a bunch of them. The segment is a unit length. And so there's this many to subdivide the space.

They call this a cage. All of these triangles live in horizontal planes. There's no way for this vertical robot with translation only to get out. It's too long.

And so the starting configuration is going to be up here. The destination is going to be down here. And in the middle triangle here, we're going to put a whole bunch of triangles like this.

So just squish this down. Put them all in here. And you're going to be able to penetrate if and only if there's a blank spot.

End of this paper. I think we covered almost all of these proofs. We started with the base problems, 3SUM, 3SUM prime, GeomBase, two versions of separator, which we weren't directly reducing from but we mimicked a zillion times. We had the degeneracy over here, strips covering a box, triangles covering a triangle, hole in the union, visible triangle. Those are actually all identical to each other. You could reduce in all directions.

We talked about point cup. This was the linear programming. Motion planning, 3D motion planning happen to reduce here instead of there, and visibility. Cool. Lot of

fun, little reductions, and it gives you a flavor for n squared hard-ish problems.

AUDIENCE: Are there restrictions at all?

PROFESSOR: I think all the restrictions are open. I mean, you'd have to check all the paper since 1995. But definitely in that paper, they're open. And I haven't heard of any-- there isn't a ton of work going the other direction. But it would differently nice to, especially if 3SUM-- it builds more if 3SUM is the right problem, if you can do reductions in both directions.

I want to show you a couple more-- I'll show you one more reduction, which relates to a problem to be proved was strongly NP complete at some point way back when. This is about fixed angle chains.

You have this kind of linkage structure. These are rigid bars. These are rigid angles.

But you can still twist one segment around another. So it preserves the angles and the edge lengths. And so if I give you a structure like this, I want to know if I spin along this edge, if I take all this stuff and rotate it out of plane around this edge, does it hit anything? Or can it go all the way around? What about this edge? What about this edge? What about this edge?

This is a polynomial time fixed angle chain problem. I want to know for every edge, which one's spinning causes a collision. In fact, if I just want to know whether these guys cause a collision, because if they do, a plus b equals c. I think that's maybe clear enough with your negative B/2 in the middle. In this case, we've shifted-- the A, B, C is the x-coordinate in this structure. And these are candidate foldings here where we miss, here we collide.

We separated things out by taking every item of a, subtracting a huge number from it to put it over here, and every item of c, adding a huge number. And because we're in the 3SUM prime problem, we know we get one item for each, so adding and subtracting and matching huge number will preserve all 3SUM pairs, 3SUM triples.

So that let's us separate out this picture. And then you just have to check this reflection corresponds to adding in the right way, because of negation we divided by 2. That's it. So that's another.

And there are a bunch of other problems, like if I give you two polygons, I want to know whether I can translate this polygon to fit inside that polygon. That's also 3SUM hard. Proof is a little bit messy, so I don't have it here.

Let me mention me another more recent use of 3SUM conjecture is some non-quadratic lower bounds. So we're still going to assume the 3SUM conjecture, but we're going to prove that a problem requires some time other than n squared.

So here are two problems where this has been done. These are graph problems, which is cool, because everything we've seen so far has been a geometric problem.

So the weighted, undirected graph, I want to know whether there's a three cycle, also called a triangle, of given weight. So I want to know, for example, is there triangle weight of 0. This can be done in polynomial time, obviously.

And the lower bound says is the number of edges to the 1.5, instead of 2 or however you want to think about this, minus epsilon. This is Mihai Petrescu. What he says is this problem, finding this in this much time, is 3SUM hard, meaning if this is possible, then 3SUM could be solved in sub-quadratic time. So there's a gap between this bound and 3SUM bound, introduced by the reduction.

Here's another fun problem. Here we're given an unweighted graph, undirected graph. And we just want to find e triangles. Just list them form me please or tell me there aren't that many.

You cannot do that in any better than e to the 4/3 minus epsilon times if you believe the 3SUM conjecture. So these are both 3SUM hard in a different sense from what we were using before.

Before it was quadratic quadratic. With graphs, also there's v verses e. But neither of these are quadratic no matter how you slice them. So there are different.

So there are a small number of pounds of that form. And that's kind of interesting and relatively hot area. So a few people are thinking about that.

In particular, there's been a recent surge of interest in thinking about graph problems. Now for graph problems, there isn't a ton of work relating 3SUM, which is a very arithmetic problem to graph problems, but this is the beginning of that. But there are some other problems which people think are hard. So let me give you some of them.

Diameter-- so here we're given a weighted, undirected graph. I want to know-- so delta v, w-- this is like CRLS notation. This is the weight of the minimum weight path from v to w. And I want to know the max overall v to w. What is the longest shortest path? That's diameter.

Conjecture-- no V to the 3 minus epsilon. So here there's a lot of interest around cubic problems, because this problem seems cubic.

Another closely related problem is all pair shortest paths. I want to know delta of v, w for all v and w. This is, of course, a harder problem than diameter. Also, this conjecture implies conjecture over here.

This one's a little more famous. The all pairs shortest path conjecture is that you cannot solve this in truly sub-cubic time. There are, again, poly log improvements. But you cannot beat-- we don't know how to beat by a v to the epsilon factor, n to the epsilon factor over the standard algorithm, which should be Floyd Warshall, the triply nested loop. Relax every edge n times is the standard cubed algorithm. For sparse graphs you could do better. For dense graphs, the claim is that's the best you can do.

And so there's a bunch of problems that are all pair shortest path hard. There are some problems that are diameter hard. Being diameter hard is a little bit stronger.

Obviously, diameter can reduce to all pairs shortest paths via now we want a sub-cubic reduction, something n to the 3 minus epsilon time instead of n to the 2 minus

epsilon. You can reduce diameter to all pairs shortest paths. Big open problem is whether you could reduce all pairs shortest paths to diameter.

But you can reduce all pairs shortest paths to some cool problems. I have an image of some reductions. So negative triangle-- is there a triangle of negative weight?

Over here we wanted a triangle of weight exactly 0. That helps you find things faster. Over here we just want a triangle of negative weight. There's more options for those. Finding that is all pairs shortest paths hard. You can reduce all pairs shortest path that problem-- kind of crazy.

And here, the reduction is not a single-- or a constant number of call reductions like we've been doing. This has a huge output. This only has a yes or no output. So this reduction's a little bit crazy. But basically you take the sum over all calls for this reduction. And it just has to work out that if you could solve negative triangle and sub-cubic time, then you could also solve all pairs shortest path in sub-cubic time.

So if you take the sum of the n primes to the power 3 minus epsilon, this should work out to n to the 3 minus epsilon over the different the different calls you make to the negative triangle. So it's a weaker notion of reduction, which lets you prove these things.

Kind of a big innovation from just-- this one hasn't even published yet. It will be in [? Sodo ?] 2015. It's on the archive now. And there been a few papers over the last few years doing these kinds of reductions. Actually the negative triangle one is from an older paper by Vassilevska Williams and Williams. It's a husband and wife team, Stanford.

OK, some more problems, radius-- for vertex v, the radius around v is how big a ball do need to grow in order to cover all the vertices. I want to know what's the farthest vertex w from v. And then the radius of the graph is what is the best such vertex that minimizes the radius, the vertex.

So closely related to this. But it is equivalent to negative triangle. Both of these are-- this is a-- well, OK, of course, you could reduce negative triangle to-- well, it's not so

obvious. You can reduce radius to all pairs shortest paths, because you have all the deltas. You can compute that max-min in quadratic time. So that means all these problems are equivalent to each other up to sub-cubic reductions.

OK, median is another problem-- very similar. I just replace this max with a sum. So that's some other kind of central located vertex. That's median. And that's also equivalent to all these problems.

So there's a growing list of problems that are in this sort of cubic space. All the problems are conjectured to be cubic. And a lot of them are equivalent to each other, though there's this divide between diameter and all pairs shortest path.

There are some bounds, assuming strong ETH. So if you have strong ETH, then there's no e to the 2 minus epsilon algorithm. This is not quite what we want. We want to v to the 3 minus epsilon. This is a statement about sparse graphs.

You can beat this for dense graphs. For sparse graphs this is interesting. This is for the worst case relationship between v and E. So you get somethings like this. You get this even if you allow some approximatability. But we still don't have a way to actually prove this conjecture. Yeah.

AUDIENCE: Since you haven't said anything about it, I assume that 3SUM and this world, there is no bridge between them.

PROFESSOR: I believe there's no bridge currently. This is the closest thing. And they do seem similar. And also here we have cubic. There we have quadratic.

So maybe you can do it. Maybe with 5 SUM you could show some relation. I don't know. But there's no such theorem yet.

These problems sound very similar to these problems, in particular listing a bunch of negative triangles is just as hard as finding one. And so there's some similarity to over here. But the constraint on the triangles is different. Here we want negative ones. Here any triangle or triangle of 0 weight.

So it's an interesting space. It's still very ongoing. All of these things-- this stuff, and this stuff-- is all within the last four years. So it'll be interesting to see how it develops. But these are the current approaches to understanding n squared, n cubed, and sort of the low end of the polynomial spectra. I think it's pretty interesting, but also where we know the least and have the fewest general techniques for proving things.

3SUM is a little more established. And there's a bunch of proofs like the one you've seen-- not too many out there actually. But they're quite accessible. This stuff is still, I think, converging. But very exciting things, relating all these algorithmic problems to each other and kind of a nice way for us to end.

This is my last lecture for 6890. And the next two classes are lectures by [? Acosis ?] about how [? arithmic ?] game theory in a class called PPAD and PPAD hardness, which is its own universe, but around economic game theory and very cool stuff. He is the expert on it and he's a professor here, so he graciously agreed to do two lectures on it. Should be fun. I'm looking forward to it. Thanks.