AMARTYA SHANKHA BISWAS: Let's start. So today we're going to do some NP hardness reductions. So let's just do a quick recap of P and NP. So let's see. So P is-- so you have a decision problem. So I have an input x and you have some algorithm, A, and it spits out an answer, which is either 0 or 1. So that's a decision problem. And if a problem isn't P, this algorithm A runs in polynomial time.

So what about NP? So NP is when the solution is verifiable in polynomial time. So let's say you have an input x and some oracle, which is [INAUDIBLE] run exponential time or is has infinite computation time gives you an answer, so you get x and you get an answer which is either 0 or 1. And you also get a certificate. So let's call that a certificate. And given these three values, you can verify whether the solution was correct or not in polynomial time. Make sense?

So clearly if you're going to compute the answer in polynomial time, you can also verify in polynomial time. So P is a subset of NP. And so that's it. So now, NP hard problems are problems that are at least as hard to solve as any problem in NP. And so now today we're going to be doing some reductions. So I think we did some of them in class, Nintendo games or something. Is that what we did? So today we are going to do some less interesting examples, but let's see.

So how do we do reductions? So if we know that we have a problem A, which is a hard problem, and we want to show that problem B is hard. So we want to draw this implication. So if we want to draw this implication-- so this is equivalent to saying that if B is easy, then A is easy. Sorry, other way. This is the counter positive.

So assuming that B has a polynomial time solution, A has a polynomial time solution. And that statement is equivalent to saying if A is hard, B is hard. So if you know that A is an NP hard problem, we can say that B is an NP hard problem. So if we want to show that B is an NP hard problem, first we show that B is an NP, and then we show that B is hard.

So the way do this step is-- so your A looks like this. You have your algorithm, and it spits out an answer in 0 to 1. And B looks like this. Let's say you have an input y and it spits out an answer in 0 to 1.

And you want to find a function R which takes your x and sends it to y. So basically, if you know how to solve B very fast, then you can take an input to A. So you take x, you transform it, and then you apply B. And the condition is that A applied to x is the same as B applied to R of x.

So basically, what you are doing is you're showing that A is easy by showing that you can use-- so let's say B is easy. We can use B to compute A, so then A must be easy. But since we know that A is hard, that there's something wrong in our logic, so B must be hard. Does that makes sense? Yes? Sort of?

So let's move onto an actual problem. So the first problem we're going to reduce is the Hamiltonian path. So a well-known NP hard problem is the Hamiltonian cycle. So here our A is-- so it's a Hamiltonian cycle. So what's a Hamiltonian cycle? So what's a Hamiltonian cycle? So a Hamiltonian cycle-- so let's say you have a graph. So we have this graph. Let me draw this out. That's it.

So a Hamiltonian cycle is a cycle in the graph which starts at some vertex, visits all the other vertices, and comes back to the starting vertex. So in this case, we could do something like go here, and then take this vertex, take this vertex, take this vertex, and come back here. So that is a valid Hamiltonian cycle. So this graph is a Hamiltonian cycle.

So the decision problem is here that given the graph, does it have a Hamiltonian cycle? And that problem is NP-hard, so you can [INAUDIBLE] polynomial [INAUDIBLE]. So now the new polynomial shows NP-hard, which is B is Hamiltonian path.

So the Hamiltonian path is a very similar problem. Instead of a cycle, you remove the requirement that you have to come back to the starting point. You can just start anywhere and [INAUDIBLE] all the vertices and stop. So for example, if you remove this edge, this graph no longer has Hamiltonian cycle, but it has a Hamiltonian path, which is just this line. Simple.

So this is a simple reduction because the problems are very similar. So the first step is, of course, showing that Hamiltonian path is an NP. So that should be pretty clear because-- so what is our certificate here?

So if someone says, OK, I have solved the Hamiltonian path and this is my Hamiltonian path. And he gives you a certificate which is the actual path. So you can always look at the certificate, check the path, and see if it's a valid path. And then you can verify the answer in polynomial time. So that's the linear time verification.

So this is true. So now what we're going to show is B is hard. So the way we do that is we do a reduction. So the reduction is given an input to the original problem, A. So let's say an input to A is in the form of a graph. And now you have transformed this graph somehow to G dash.

And then you have to argue that-- so this is the transformation R-- and you have to argue that the solution, the answer that this will spit out, is the same amount that this would spit out. So let's look at the transformation first. Anyone have any ideas?

So you have a graph and you're using that to solve the Hamiltonian cycle problem. So how do you transform it in a way that lets you cast into Hamiltonian path formulation? Yes.

AUDIENCE: Remove an edge?

AMARTYA SHANKHA BISWAS: Not exactly. So which edge would you remove? You can't find the Hamiltonian cycle. OK, important point. So there's no point doing this reduction unless this reduction is itself polynomial because otherwise, this whole strategy of transforming and then using B to find A doesn't work. Because if the reduction is exponential time, that doesn't help you. So it--

AUDIENCE: Try removing every edge.

AMARTYA SHANKHA BISWAS: So if you remove some edges, you'll see that there is still a Hamiltonian cycle. But you remove some edges-- you can't tell. You don't know which edge to remove.

So a better way to do it is this. So let's say this is the rest of your graph. And you just look at one vertex. So look at one vertex, V. And let's say this is a directed graph. If it's an undirected graph, you can just like add one edge there and one edge back for everything.

So now you add a directed edge along this. I'm sorry. You look at all the directed edges. So let's say you have some edges coming in and you have some edges going out. So this is just a vertex and you just look at the rest of the graph and look at all the edges coming in and all the edges going out.

So this is in A. This is the original problem. And you transform this into-- you split the vertex into two, so V dash and V double-dash, let's say. And in one of them, you keep all the incoming edges and the other one contains all the outgoing edges.

Does that transformation make sense intuitively? So what do you have here? So here you had-- let's say this graph had a Hamiltonian cycle. So this graph had some cycle which went up here, did something, something, and came back.

So it would go like this. It would do the cycle and come back. So there was some cycle. And since the cycle, it contains V, so now what you're doing is you're splitting apart V and disconnecting them. So you'll still have-- if you look at the original path, it's still there, but it's been split up into a path now. It's no longer a cycle.

Make sense? So now let's argue this more rigorously. So what we want to say here is that let's say there was a cycle here. If there was a cycle here, then is it clear that there is a path here? Because just take the same edges that you had before.

If you take the same edges, they will now form a path instead of a cycle. So cycle implies path. Does that makes sense? So the other way is a little more tricky.

So let's say you have a path. So let's say you had a path. So that means that-- let's redraw this so it's more clear. So you have this new graph where you have two vertices, V dash and V double-dash. This has a bunch of incoming edges and this is a bunch of outgoing edges.

So now let's say you have a Hamiltonian path in this graph. So what does that mean? So where can the Hamiltonian path start? Can it start anywhere? Where can it start?

AUDIENCE: V double-dash.

AMARTYA SHANKHA BISWAS: Right, because V double point doesn't have any incoming edges, so it can't be in the middle of the path. So it has a start here. So it starts. It does something in there.

And where can it end? It can only end, similarly, in V dash. Because V dash doesn't have any outgoing edges, so it can't be in the middle of the path. So it has to end in V dash.

So now, if you have a path like that, and you go back to this graph-- so V dash and V double-dash are now on the same vertex. And just that path just becomes a cycle now. So path implies cycle.

So now what we have is previously what we had, right? So now we know that A of G here is equal to B of G dash. So G dash was transformation.

Also notice that the transformation was just splitting apart a vertex. So depending on your representation of your graph, it'll take something, like constant time or linear time or something polynomial, essentially. So you get a polynomial-time reduction. After reduction, you show that the answer to the reduced problem is the same as the answer to the original problem. And that means that this is also an NP-hard problem by the argument given here.

Questions? Does that make sense? Yes.

AUDIENCE: So are you creating two vertices for every vertex in the graph?

AMARTYA SHANKHA BISWAS: No, just this one. Just pick any vertex-- doesn't matter-- because you have a cycle. So if you take any vertex and split it apart, you get a path.

AUDIENCE: Oh, perfect.

AMARTYA SHANKHA BISWAS: Anything else? OK, let's move on to the next one. Grab a new board. So the next problem is-- so given the graph, is there a k-clique? Do people know what a clique is?

So a clique is this. So a clique is a set of vertices. So let's say C subset of V for the set of vertices, such that C is a complete graph. Let's just draw a diagonal. That's probably easier.

OK. So in this example, so you have this graph. This one. So look at this set of vertices. Every pair of them is connected to each other.

What that means is that if you just look at the graph with these vertices, it's a complete graph. That's what's called a clique. And in this case, this is a 4-clique. So you have four vertices, this is a 4-clique.

So the position problem is, given the graph, does there exist a k-clique? So this is, again, known to be an NP-hard problem. So now we will use this to show that this problem is NP-hard.

So this problem is independent set. So again, so given the graph, what is an independent set? Anyone want to explain? So what an independent set is this. So let's say you have a graph which looks like-- so kind of complementary to the definition of a clique.

An independent set is a set of vertices, such that no pair of them has an edge between them. So in this case, if you took this vertex, you took this vertex, this vertex, and this vertex-- so you can see, none of these vertices have an edge between them, so that is an independent set.

So in this case, you're taking a set of vertices which is a complete graph, so all of them have edges between them. And in this case, you're taking vertices which are completely disconnected. So now we're going to find a reduction from this problem to this problem.

So first of all, independent set is an NP. Is that clear? How would you show that? So how would you create a certificate which would tell you that-- so how would someone create a certificate which would convince you, in polynomial-time, that this is correct? So the certificate would be just give you the independent set.

And you can check if something is independent set. So given I-- let's call the independent set, I. So given the set of vertices, you can verify that it is an independent set in polynomial-time. So just look at all pairs and check if there's an edge-- just an n squared and Q whatever is polynomial. That's important.

So now let's look at our transformation. So again, as before, you have A, which is given by a graph, and you want to transform into something. So the important note here is that in the clique, you have-- so for your clique C, all the pairs of vertices are connected. In I, no pair of vertices are connected.

So what should be a logical transformation that would map a clique to an independent set? Anyone? What do you think you should do to the graph so that the clique becomes-- yeah.

AUDIENCE: Invert the existence of edges so they're [INAUDIBLE].

AMARTYA SHANKHA BISWAS: Precisely.

AUDIENCE: [INAUDIBLE].

AMARTYA SHANKHA BISWAS: Yup. Exactly. Great. So all you have to do is if you want to turn a clique into independent set, you just-- what is it-- complement the adjacency matrix. So every edge does not exist now exists and every edge that did exist is gone.

So you create a graph, G dash, with the same [INAUDIBLE] vertices, except the edges are now complemented. So the E bar just means the edges that were not. So let's just draw an example. So let's say you had-- and what does this become? So let's draw the vertices first.

Let's go one by one. So let's take this vertex. Let's take this vertex. And what edges does it have? So it goes here. It goes here.

It doesn't go here, so we draw an edge here. It doesn't go there, so we draw another edge there. And it doesn't go there, so we draw another edge there.

Now, this vertex. It's connected to all of these except this one. So that means that there's an edge there. Let's take this vertex. It's connected to everything. Oh, it's connected to everything.

Let's take this one. It's connected to these two and nothing else, so we need to connect it to this guy. And I think that's it. Yeah.

Similarly proceeding, this is connected to everything except that, so I guess this goes there. And I think that's it, right? Or is there more? Three. No, OK. So that is a complementary graph I think. So you can probably verify that.

So now let's look at the clique in this graph. So this is the clique. This is the largest clique, rather, but this is a clique. You could have other cliques, like, for example, this these three things are also a clique.

So now look at what this is mapping to. This is mapping to this vertex, this vertex, this vertex, and this vertex. And you can see that's an independent set. So does that transformation make sense?

So now the proof should be intuitively clear. So let's just go through it moderately rigorously. So let's say you have a clique here. So your clique here, for every pair of vertices in the clique, there's an edge between them.

And so if that maps to-- so let's say clique C maps to-- let's say this maps to I. So for all U, V element of C, you have U, V, element of E. So if this is a clique, for every pair of vertices, that edge is in the original graph, which means that U, V is not an element of E for all U, V element of I.

So for every U, V element of I, we have this, which means that-- does that make sense? So that means that that's the independent set criterion. So you reduced clique to independent set. And that means that independent set is now NP-hard.

OK. How are we doing on time? OK. So now let's do a more complicated example. Any questions on these two? Make sense? OK. So let's try this.

So let's start by erasing something. So this is the next problem. So as before, our A is k-clique. So it says there's a clique-- rather, let's add this in this way. Clique size greater than or equal to k.

So that's the decision problem. Is there a clique of size greater than or equal to k? And B is, it's called Max-2-SAT. So what that means is that-- so it is somewhat like normal 2-SAT, except basically you have some clauses and you have some literals.

So each of these literals contain values 1 or 0. And each of these clauses is something like xi or xj. And actually, there can be naughts in front of this, so let's say xi 0 xj. Or it can be other things.

So it's xi 0 xj, or 0 xi 0 xj, or xi xj, and so on and so forth, so just the normal 2-SAT. So now the decision problem is to-- does there exist an assignment, such that greater than equal to k clauses-- so that the decision problem. So is there an assignment to literals such that at least k of these clauses are satisfied?

And so now we're going to show that it is with k-clique. So again, is it an NP? So what is the certificate? How would someone convince you their solution to this problem?

AUDIENCE: Give you the literals.

AMARTYA SHANKHA BISWAS: Yeah, so give you an assignment of values to literals. And then you can go through all the clauses and check them. So if they give you like x1 equal to 1, x2 equal to 2, x equal to 0, x3 equal to 1, and so on and so forth, you can then go through and check all the clauses and see if greater than k are satisfied. So this is an NP.

So now let's try the reduction. So this is how the reduction goes. Let's see. So naturally, we have k-clique, or greater than equal to k-clique, rather. So that means you have a graph with a set of vertices and a set of edges.

So let's say you have a clique, V dash, subset of V, and mod of V dash is greater than equal to k. So now you have to somehow construct literals, construct clauses, which will reflect this behavior. So first of all, this may not be clear right now, but let's say we take some literals like this.

So let's say we take xi for all i element of V. So for every vertex in the graph, we take a literal. So if the number of vertices is n, we have n literals because it's corresponding to each vertex. Also, we take a dummy literal. Let's call it Z.

So now how do we get our clauses? So the general idea is that if a vertex is in the clique, you will assign it 1. If it's not in the clique, we will assign it 0. Everything outside of the clique is 0.

So this clause, not xi or not xj-- so what is the value of this clause normally? So let's say xi and xj are both outside the clique. So i and j are both outside the clique. That means that both of them are 0 and so this is true.

What if one of them is inside the clique and the other one is outside? It still is true because one of these naughts is 1 and that is still true. Let's say both i and j are inside the clique. So in that case, you have 0 of xi is 0 and 0 of xj is also 0.

That's the only case that this is false. So the way we take care of-- so we're trying to maximize the number of true clauses in some sense. It's like you had maybe an explanation of why you're using this clause. So what we do instead of taking all the xi, xj pairs, is we just take xi, xj, such that i, j is not an element of E.

So what does that mean? So now if you had the graph which looked like this, now it looks like 1, 2, 3, 4. So you would take 0 x1 and 0 x4. You would take 0 x1 and 0 x3. But you would not take 0 x2 and 0 x3.

So what does that do? That means that if you follow the assignment according to the clique rules-- so if i and j are both in the clique, this clause will not be included. Does that make sense-- what set of clauses you are taking?

OK, so let's continue and it will hopefully be a little more clear. So the other sort of clause we're going to take is xi or z. And the other one is xi or 0 z. So the reason we're taking these is that if you wanted to Max-2-SAT on this alone, you can just set everything to 0, and that would give you a maximum.

So sort of not do that-- so to sort of minimize the number of things to be set to 0, you were doing this. So is this just some hand-wavy argument why you're doing this. So let's actually try to do some analysis on this.

So let's say you do this transformation. So do the clauses make sense? Does the first clause sense? So you have 0 xi, 0 xj for every i, j which is not in the graph.

So how does this work? So let's say you have V dash such that size of V dash is greater than or equal to k. Actually, let's just make size is V dash equal to k. So if you have a clique of size greater than or equal to k, of course, you have a clique of size equal to k. You can just throw away some of the vertices.

So you take your V dash such that this size is equal to k. And you set xi is equal to 1. So you set xi equal to 1 if i is element of V dash 0, if i is not an element of V dash. Make sense? So you would set everything in your clique to be 1, everything outside to be 0. And let z equal to 1.

So you're starting with the assumption that-- so you're showing one direction. You're showing that given that there's a clique of size greater than equal to k. And now you're are going to construct a Max-2-SAT instance which has the satisfied number of clauses greater than equal to k. And then we'd show the other direction.

So now let's look at how many clauses we have to be satisfied. So the first type of clause was 0 of xi or 0 of xj. So how many of these clauses are being satisfied?

So first case, i and G are both outside V dash. Is the clause satisfied in that case? Yes, because by definition, if they're outside V dash, they're both 0, so their naughts are both 1, so they're 1. So if the i and j are outside V dash, you're good.

So what about the case when one of them is inside V dash? Is the clause satisfied. Yeah, because one of them is 0, which makes the 0 1, and the whole thing is. It's an [? arc, ?] so it's satisfied.

Let's say both of them are inside V dash. Let's say both i and j are inside V dash. Then this clause just doesn't exist because of the condition that i, j had 0 elements of V. Because if it's inside the clique, then that edge obviously exists, and therefore this clause is not in the set of clauses we're using.

So essentially, every clause of this form will be satisfied. And how many clauses of this form are there? The number of clauses of this form is just E bar, where E is the complementary edge set of that graph. That's E bar. Does that make sense?

Next clause is xi or z. So since we have considered z to be 1, this clause is always satisfied. So this just gives us mod of V because this is for every i-- I should mention that. For i, this is also for all i. So for every i, so they all have a number of xi's are V, so that's it.

So the third type of clause is xi or 0 of v. So 0 of z, since z is 1, 0 of x is 0. So the only cases where this clause is true is where? Is when xi is 1. And xi is one only inside V dash, so the number of clauses satisfied here is mod of V dash.

And mod of V dash is what? It's just k. You see this? All three clauses make sense? Can you see why the first one is it's the size of V bar? Because all the clauses are satisfied.

The second one, also all the clauses are satisfied because z is 1. Every clause is just the number of vertices. And the last one, it's only satisfied for the cases where xi is 1. And xi is 1 only inside the clique, so that gives you the k things in the clique.

So now we can finish formulating our transformation. So our transformation was you set-- where was the transformation? Yes.

So these are our clauses and now we're going to set the k. So the Max-2-SAT was with the condition of k. So here you'll set it to be mod of V bar plus mod of V plus k. So does that make sense?

So basically, our Max-2-SAT problem-- so the reduction to the Max-2-SAT problem-- so we specified the clauses. We specified the literals. Literals were xi for all iV and the dummy z. We satisfied the clauses and we specified the threshold we are trying to achieve. So those three things completely specify the Max-2-SAT problem.

OK, proceeding. So we just showed one direction. We showed that if there's a clique of size greater than or equal to k, we reduce this to a size of exactly k. Then we did this assignment and we showed that the total number of clauses being satisfied is mod of V bar plus mod of V plus k.

And that is the first direction. So now we're showing that-- so if you have a solution to clique, we have a solution to k-SAT, Max-k-SAT. So now we do the other direction. So let's say we have a solution to Max-k-SAT. Let me get this out of the way.

Let's do this. So let's say-- so this part is a little more tricky. So we start with Max-k-SAT has a number of satisfied clauses which is greater than or equal to mod of V bar plus V plus k.

So we know that max-SAT that accepts. So know that so we define. We define a V dash. This V dash is in the graph for clique as the set of vertices i, such that x of i is equal to 1.

So if our Max-k-SAT has that many things, then there's some assignment to xi which satisfies that. And we take that assignment. And for every value that is 1, we put that in a-- it's not a clique yet, but it's, well, it's a clique under construction. So we make this clique under construction and we assign it values of all the vertices which are currently labeled by 1.

But we don't know that it's a clique, right? It could not be a clique. It could be something like, let's say, so this is the current V dash. So V dash could be something like you have all these vertices and let's say some of them are connected. But then you have this dangling. So this is not connected. This is not connected.

So it's not a clique. So let's say this is vertex i and this is vertex j, and this whole thing is V dash. And let's say somehow you have this anomaly. You have that this guy is not connected to all the vertices. So let's take one such pair.

And let's just say we remove xi. So we just take the xi and remove it from V dash. What that equates to in the Max-k-SAT is setting xi to 0. So we take the original satisfying assignment and change the value of xi. So let's see what that does.

So we take and set xi equal to 0. And xi was originally 1. Let's actually write it down. So xi, or rather, i, j is not an element of E, but-- so these i, j were in the supposed to be clique, but the i, j is not in the edge set, so it's not actually a clique.

So the way we resolve that is just we do this. We say, OK. Let's just forget about this vertex. Let's say it's just not in the clique. So we set our xi to 0.

And what does that do? Let's look at how that affects the number of sat clauses. So the first one is x [? over ?] z. So now here we had not set z to be 1. This is something in thing.

If z is equal to 0, we can just replace z by 0 of z. So the clauses are symmetrical with respect to z. So it's xi [? over ?] z or xi 0 z. So it doesn't matter which one is set to be 1 or 0. If one of them is 1, one of them is 0.

So let's just say that z is 1 without loss of generality. So what that does is-- so initially, this clause was 1 because z is 1. And now it goes to 1 and nothing changes. OK, sounds good.

What about the next one? So now we have xi or 0 of z. So 0 of z is 0 and xi just went from being 1 to being 0.

So what happens here is that initially, the clause value was 1 and now it goes to 0. So that's not good because now we are no longer satisfying Max-k-sat clause possibly because we had some number of satisfying clauses, which was above our threshold. But now we lose a clause and we could be going below the threshold.

But then we look at the third clause. And what that does is-- so let's say we look at this specific clause, the one which said that xi and xj. Note that this clause exists because xi, xj is not an edge you need. And therefore, by that condition, this clause exists in the set of clauses.

So what was the value of this clause initially before? In the initial assignment, what was the value of this clause? Note that i and j are both in V dash. What was the value of this clause in the original assignment? Before we set xi to 0? What was the value of xi in the original assignment?

AUDIENCE: 1.

AMARTYA SHANKHA BISWAS: Yes, why is it 1?

AUDIENCE: Because xi was in V dash.

AMARTYA SHANKHA BISWAS: Yeah, so xi was in V dash, so it's 1. xj was also in V dash because that's the anomaly we saw it, right? There was not an edge in the clique. So this was originally so this was 1 and this was 1.

So this was 9 and this was 0. And our R0, R0 is-- this used to be 0. So what happens now though? Does it change? It changes to 1 because xi goes to 0. Now this thing becomes 1 and so it changed to 1.

So there will be other clauses with xi, but realize that 0 of xi-- so if xi is changed to 0, 0 of xi is changing to 1. So whatever happens here, it will only increase the number of clauses that are being satisfied. So you lose only 1, but you gain at least 1. So eventually, it does not change.

It does not change the number of satisfied clauses, so that's important. So what we did is we started with some satisfying assignment that we assumed existed. And then we just changed the variable and we said that it's still at least as many clauses being satisfied. So we did that. And now we have one less vertex that is violating-- so now we have one less vertex that is violating the clique property.

So now we can take this step-- this setting xi to 0. We can just repeat this. So how long do we repeat this? So we repeat this till there are no longer any violations.

So every time we find a violation-- so once we delete xi-- so let's this is gone. So now we look for the next violation. The next violation is this edge. So there's no edge there. So we can either take this vertex-- so let's call this k. So we can either remove xk or remove xj. So we remove one of them.

And once you remove that, you will end up with the 3-clique. So what happens is once we repeat this, until V dash is a clique. Does that make sense why you can do that? So you just keep deleting vertices until this is a clique.

And those clauses plus clause condition is still satisfied. You will still have greater than or equal to that many clauses. So now let's look at what we have. We have V dash, which is a clique and xi is equal to-- so once you have done this process as many times as you need, when is xi equal to 1?

So remember that-- so V dash is also being updated. Every time you set xi to 0, xi is being removed form V dash. So that property is always satisfied-- that define V dash equal to xi for xi equal to 1. That property's always invariant.

That means that even after reviewing these repetitions, you still have xi equal to 1, if and only if i is E V dash and 0 otherwise. So does make sense why that property is solved? So you have a clique and you have this assignment. So that should take you back to this.

So remember where we took a clique of size k and we found that if you go through all the algebra, you will find something which is like you will go through and you will get mod of E bar plus mod of V plus mod of k, right? So here you have a clique. So forget about what you did before.

So just consider this is an assignment according to those rules. And those rules give you that you should get number of satisfied clauses is equal to mod of E bar plus mod of V plus mod of V dash. Realize that V dash here is not k. Does that make sense?

Because V dash is just-- so you started with some set. You started deleting some elements. You throw [INAUDIBLE] randomly, but you ended up with some V dash. And by this argument, you had E bar plus mod of V plus mod of V dash-- E bar from this clause, V from this clause, V dash from this clause.

So you didn't end up with this. And you know that because of what we showed here does not change number of satisfied clauses, it's still greater than or equal to mod of E bar plus mod of V plus k. And we cancel this. You cancel this.

And you get mod of V dash is greater than or equal to k, which means that you have a clique of size greater than equal to k. Questions? Does that make sense? Really? All of it? OK. That's good. Any case.

So let's go back and see what we are doing here. So the way you are doing NP-hard reductions is you take a problem that you already know is hard, you take any arbitrary instance of that problem, and you transform the input into an input to the problem that you're trying to show is hard. So you take problem A, which you know is hard.

You transform the input into an input for problem B. And then you show that if you can solve [INAUDIBLE] problem B, you will be able to solve problem A. But since you know you can't solve problem A in polynomial-time, you know that you can't solve problem B in polynomial-time.

And the other important thing to notice here is that the reduction needs to polynomial-time. So look at this reduction, for instance. What are you doing here? You're taking every vertex. You're making a clause. And how many clauses do you have?

Well, you have about n squared clauses here. You have Rn clauses and you have Rn clauses here. So time to construct that is like roughly Rn squared. So you're constructing the clause in polynomial-time.

So you have a polynomial-time reduction. And if you reduce a known NP-hard problem in polynomial-time to an unknown problem, you can show that it is NP-hard. OK. I don't think we have time to do another problem, so we're done.