Please work on the handout problem with a partner.

Sudoku is usually played 3x3 but there's no reason you can't play 4x4 Sudoku (pictured to the right). In general, a n x n Suduku grid contains n x n "sub-boxes" and n x n spaces for numbers in each of those boxes.

Can you think of an algorithm to solve a n x n Sudoku board? What is the Big-O of your algorithm.

1			2	3	4			12		6				7	
		8				7			3			9	10	6	11
	12			10			1		13		11			14	
3			15	2			14				9			12	
13				8			10		12	2		1	15		
	11	7	6				16				15			5	13
			10		5	15			4		8			11	
16			5	9	12			1						8	
	2						13			12	5	8			3
	13			15		3			14	8		16			
5	8			1				2				13	9	15	
		12	4		6	16		13			7				5
	3			12				6			4	11			16
	7			16		5		14			1			2	
11	1	15	9			13			2				14		
	14				11		2			13	3	5			12

The Problems of NP

Or The Most Famous Problem in All of Computer Science

Or Sudoku is Harder Than You Think

Where we are going

- Step 1: “Polynomial Time Verifiable Problems” - the thorn in the eye of CS
- Step 2: Cook's Theorem – how to do any computation with only Boolean logic
- Step 3: Other NP-Complete Problems, and implications

P: Problems We Know A Good Way to Solve

- Sorting a list
- Multiplying a Matrix
- Raytracing a 3D picture
- Searching for Stuff In Webpages

All these problems run in *polynomial time*. So we say that all these problems are in class “P”.

Is Sudoku in P?

8				5			9	
1		2	4	9		3		
				2	1	6		
		1			5		2	
	3					5		7
5	9				4			
			9	8				6
9			5		3	8		4
		8			6			

For Sudoku to be in P, there must be a polynomial time algorithm to solve it.

Is Checking a Sudoku Solution in P?

8	4	3	6	5	7	2	9	1
1	6	2	4	9	8	3	7	5
7	5	9	3	2	1	6	4	8
6	8	1	7	3	5	4	2	9
2	3	4	8	1	9	5	6	7
5	9	7	2	6	4	1	8	3
4	1	5	9	8	2	7	3	6
9	2	6	5	7	3	8	1	4
3	7	8	1	4	6	9	5	2

Say I give you a completed solution, as above. Can you check the solution is valid in polynomial time?
If so, Checking Sudoku is in P.

What if we just guess and check?

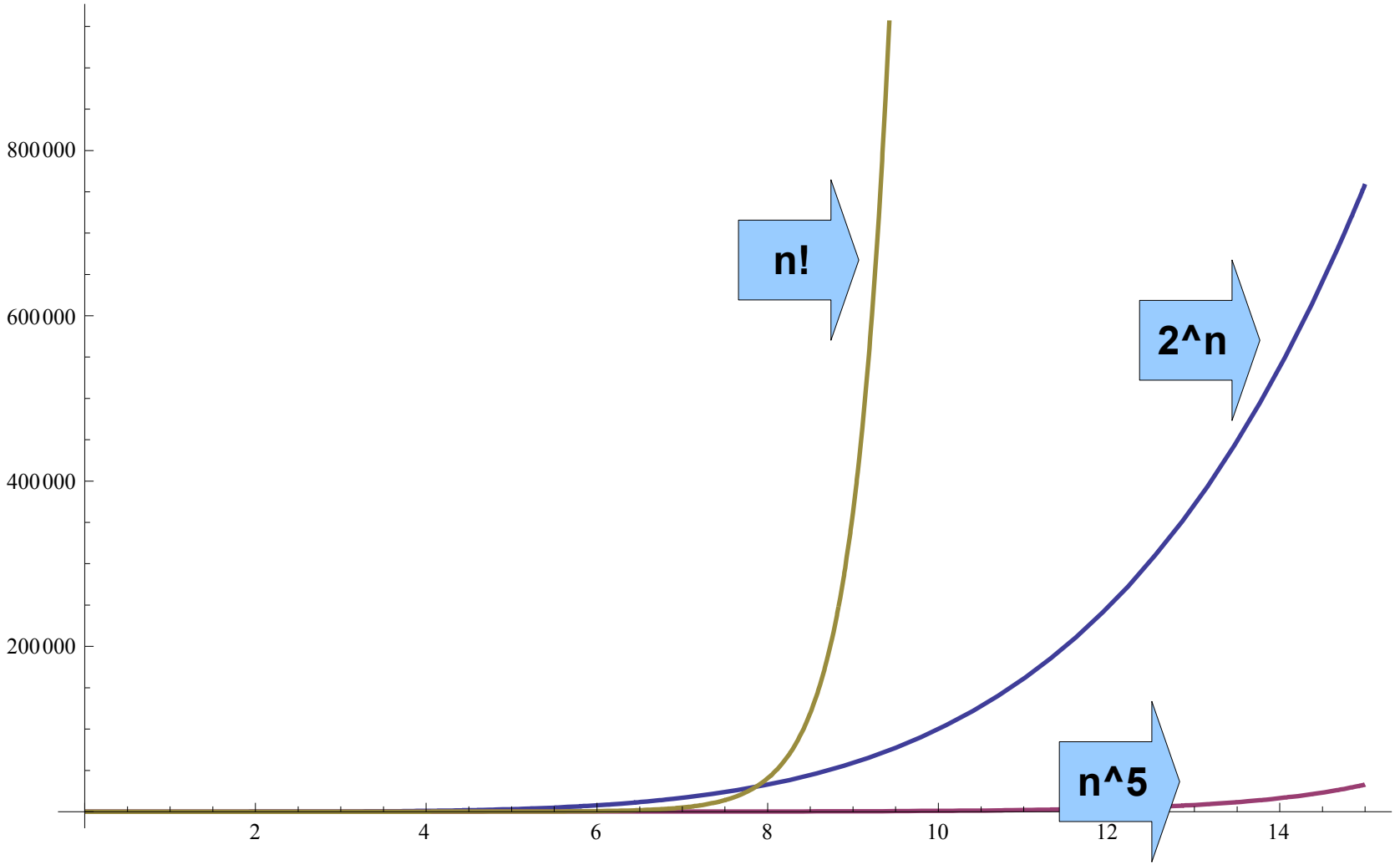
8				5			9	
1		2	4	9		3		
				2	1	6		

My proposed algorithm:

1. Generate a permutation of 1 to n^2 for each row.
2. Once I have a “proposed solution” check (in polynomial time) to see if my solution is valid.
3. If it's not, try a new permutation and go back to step 2.

How fast is it? Well...we can see that there are gonna be (n^2)! permutations for each row. That seems like it might be a little much.

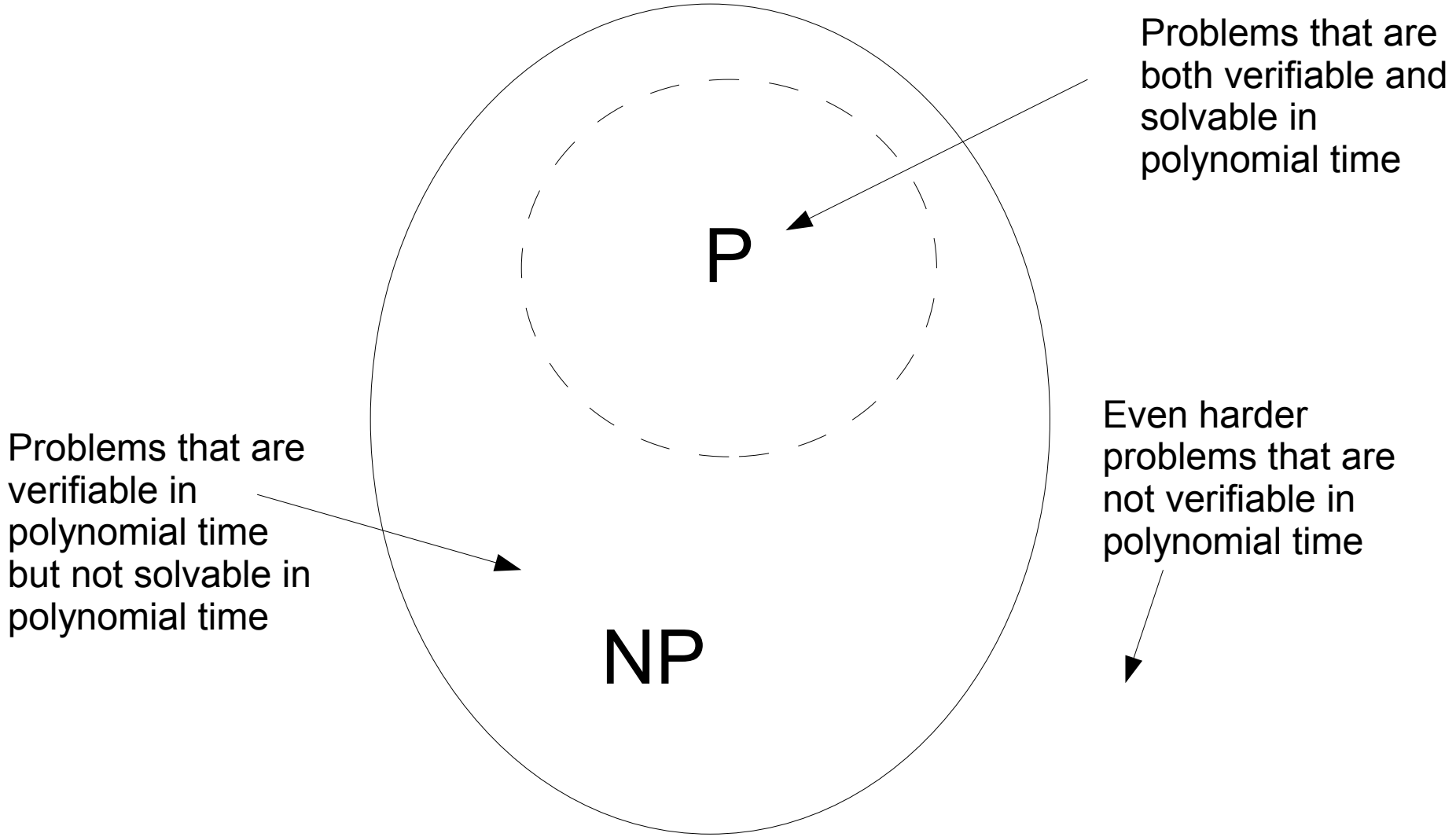
So inefficient it will hurt your brain



Sudoku is an example of a “polynomial time verifiable” problem. For problems like these, it is easy to verify a given solution is correct. But it can be hard to find that solution.

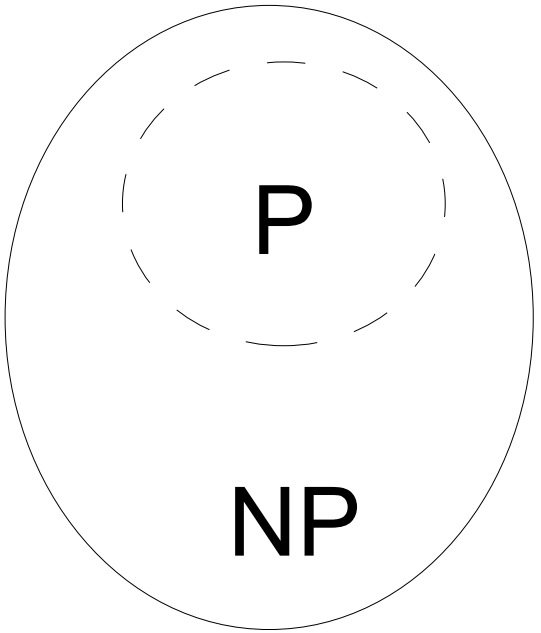
Let us call all such problems “NP” problems.

How Do P and NP relate?

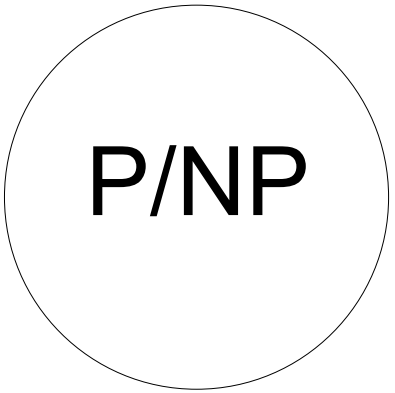


MAY NOT BE TRUE

The Most Famous Question in CS: Does P equal NP



?



ONE OF THESE TWO THINGS IS TRUE

Where we are going

- DONE:“Polynomial Time Verifiable Problems”
  - Problems that can be solved in polynomial time are in P
  - Problems that are easy to check but hard to solve are in NP
  - Does P = NP is the famous question
- Step 2: Cook's Theorem – how to do any computation with only Boolean logic
- Step 3: Other NP-Complete Problems, and implications

Work with the person sitting next to you:

Come to a consensus on the solution to the question on the back of your handout

A New NP Problem: Satisfiability

Given a Boolean equation, determine if there is an assignment of variables that makes the statement true.

(A and B) and not(C and A)

Cook's Theorem: In Concept

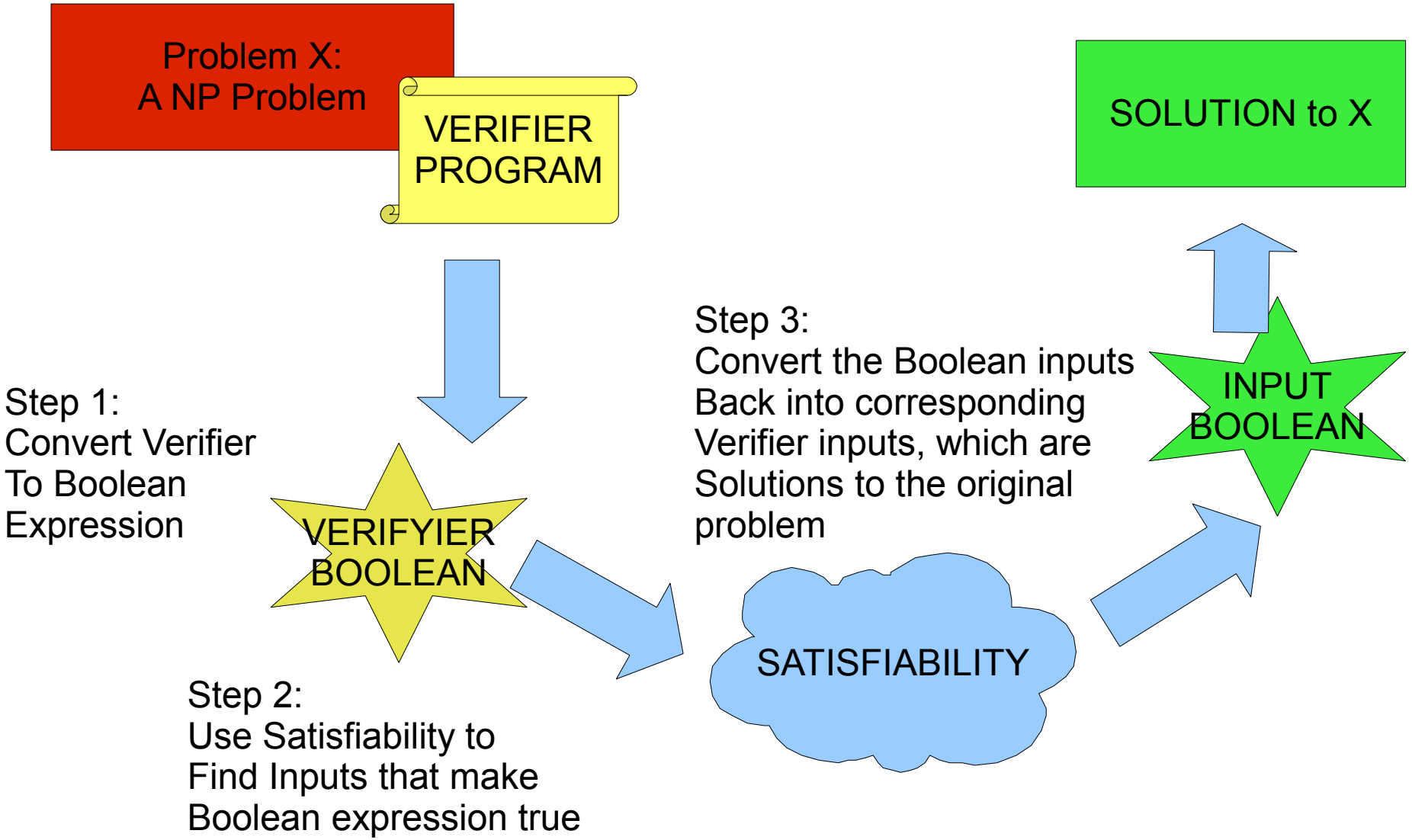
- “Satisfiability” (as we have shown) is a great way to represent complex rule systems
- But what is a computer program but a complex rule system?
- What if we could reduce a verifier (which all NP problems have) to a boolean equation
- If we could do that, then if we could solve satisfiability in polynomial time, we could solve any NP problem in polynomial time

A Very Simple Programming Language

Only 6 Operations

- If current tape has a 1 GOTO a particular state
- If current tape has a 0 GOTO a particular state
- Write 1 on the current tape position
- Write 0 on the current tape position
- Move Tape Forward
- Move Tape Backward

Cook's Theorem: A Diagram



NP-Completeness

We call Satisfiability NP-Complete. That is, if we could solve Satisfiability we could solve any problem in NP. If Satisfiability is in P, P = NP. If P != NP, Satisifiability must be one of the problems not in P.

8				5			9	
1		2	4	9		3		
				2	1	6		
		1			5		2	
	3					5		7
5	9				4			
			9	8				6
9			5		3	8		4
		8			6			

Are there other NP-Complete problems?

A polynomial time solution to satisfiability would be an important result because:

1. It would conclusively show that P != NP
2. It would make other NP problems like factoring numbers, solvable in polynomial time
3. It would disprove the hypothesis that satisfiability was NP-complete
4. Satisfiability is a general problem frequently encountered in industy.

Which of these is a component of Cook's Theorem?

1. A method to convert a program to a boolean expression
2. A bound on the runtime of the Satisfiability problem
3. A proof that not all NP problems have a polynomial time verifier
4. A way to exploit the high speed of Satisfiability in other contexts

As a professional programmer, why would you want to be able to prove a problem is NP-complete?

1. NP-complete problems are more efficient
2. Only NP-complete problems can benefit from the techniques of Cook's Theorem
3. To show that they can be reduced to satisfiability
4. To avoid them because NP-complete problems are intractable

Where we are going

- DONE:“Polynomial Time Verifiable Problems”
  - NP is the class of Polynomial Time verifiable problems
- DONE: Cook's Theorem
  - Satisfiability is a NP problem that determines if a boolean equation can be satisfied
  - It is NP Complete, meaning that a polynomial time solution would work for every problem in NP
  - The solution is created by converting the verifier into a Boolean equation
- Step 3: Other NP-Complete Problems, and implications

Final Exercise: Brainstorming

Why is P/NP an Important Problem?
What are the implications of P/NP?

Please work on the handout problem with a partner.

Sudoku is usually played 3x3 but there's no reason you can't play 4x4 Sudoku (pictured to the right). In general, a n x n Suduku grid contains n x n "sub-boxes" and n x n spaces for numbers in each of those boxes.

Can you think of an algorithm to solve a n x n Sudoku board? What is the Big-O of your algorithm.

1			2	3	4			12		6				7	
		8				7			3			9	10	6	11
	12			10			1		13		11			14	
3			15	2			14				9			12	
13				8			10		12	2		1	15		
	11	7	6				16				15			5	13
			10		5	15			4		8			11	
16			5	9	12			1						8	
	2						13			12	5	8			3
	13			15		3			14	8		16			
5	8			1				2				13	9	15	
		12	4		6	16		13			7				5
	3			12				6			4	11			16
	7			16		5		14			1			2	
11	1	15	9			13			2				14		
	14				11		2			13	3	5			12

Hand these papers out as students walk and and be sure they work on it in pairs. Give ~ five minutes after class starts to let people think about it, circulating between the groups and stopping when people seem to be getting bored.

The Problems of NP

Or The Most Famous Problem in All of Computer Science

Or Sudoku is Harder Than You Think

Ok today we're going to talk about what some people call P verses NP. Some of you may have heard about it before, some of you maybe not. Occasionally some newspaper will talk about it. But these articles are a bit painful form me to read because they always tend to leave out what makes this problem so cool. Which is something hard to understand – in fact most students don't get to it till their 2nd or 3rd year in CS. And many don't even really **get it** then.

But we're going to go there today. Because I think you can understand it. But things might get a little hairy, so pay attention.

Where we are going

- Step 1: “Polynomial Time Verifiable Problems” - the thorn in the eye of CS
- Step 2: Cook's Theorem – how to do any computation with only Boolean logic
- Step 3: Other NP-Complete Problems, and implications

Ok, here's the basic plan.

I'm going talk about the set of problems, that are difficult to solve but easy to check. And I'm going to call those problems NP.

Then I am going to talk about a problem called satisfiability and I'm going to try to show that if we could solve it we could solve any problem in NP.

And then I am going to talk about what this means, and what other problems in this class are. And what life would be like if P = NP.

P: Problems We Know A Good Way to Solve

- Sorting a list
- Multiplying a Matrix
- Raytracing a 3D picture
- Searching for Stuff In Webpages

All these problems run in *polynomial time*. So we say that all these problems are in class “P”.

So we're going to start classifying problems here. So lets start with your regular old everyday problems. These problems have known solutions that run in polynomial time – [Write on board P – O(n^2) O(n log n)..]. like n n^2 n log n. n^3 n^100. Anything polynomial counts. Notice that sometimes we may be stretching the truth a bit when we say that n^100 is “a good way to solve” something.

Thing to Remember – P “the set of regular problems”

Is Sudoku in P?

8				5			9	
1		2	4	9		3		
				2	1	6		
		1			5		2	
	3					5		7
5	9				4			
			9	8				6
9			5		3	8		4
		8			6			

For Sudoku to be in P, there must be a polynomial time algorithm to solve it.

5

Ask students to raise hands. Hold 1 finger up if you think it's in P, hold two fingers up if you think it is not. If you don't know, hold up 3 fingers.

Well, a lot depends on how you did on the starting exersize. Did you come up with a polynomial time algorithm. if you do have an polynomial algorithm in front of you? Then its in P. Right? Does everyone see what I mean here?

Say hypothetically, you didn't come up with an polynomial-time algorithm. What can you say? Well the only thing you can really say is “I don't know”. It's very difficult to prove a problem is not in P. You might not know a solution – but does that mean a solution does not exist? Maybe the gal next to you has one. Right?

Does anyone think they've discovered a polynomial time algorithm for P. Raise your hands. Ok, for reasons that will become clear by the end of class, I doubt that your algorithm will really work. But feel free to come talk to me after class about it and we'll discuss it more freely.

Because I agree with the 3s – I don't know whether there is a polynomial time algorithm. But, and like I said you'll understand more soon, I do have good reason to believe the solution is not something you can think up in 5 minutes.

Is Checking a Sudoku Solution in P?

8	4	3	6	5	7	2	9	1
1	6	2	4	9	8	3	7	5
7	5	9	3	2	1	6	4	8
6	8	1	7	3	5	4	2	9
2	3	4	8	1	9	5	6	7
5	9	7	2	6	4	1	8	3
4	1	5	9	8	2	7	3	6
9	2	6	5	7	3	8	1	4
3	7	8	1	4	6	9	5	2

Say I give you a completed solution, as above. Can you check the solution is valid in polynomial time? If so, Checking Sudoku is in P.

Ok, I want everybody to turn to your neighbor and answer this question – is checking a Sudoku solution in P? [Give three minutes for this]

Alright I'm gonna have to cut you off. Ask students to raise hands. Hold 1 finger up if you think it's in P, hold two fingers up if you think it is not. If you don't know, hold up 3 fingers.

Ok, this one I know – and that should give you a strong hint already shouldn't it?

To meet the rules of Sudoku, all the rows must be unique, all the columns must be unique, and every box must be unique. Right. Well, I can check if n numbers are unique in n^2 time (actually I can do better than that, but lets not be fancy). And how many times do I need to do that? Well, once for every row (n x n) once for every column (n x n) and once for every “subbox” (n x n). So this is O(n^4) polynomial. Boom in P.

What if we just guess and check?

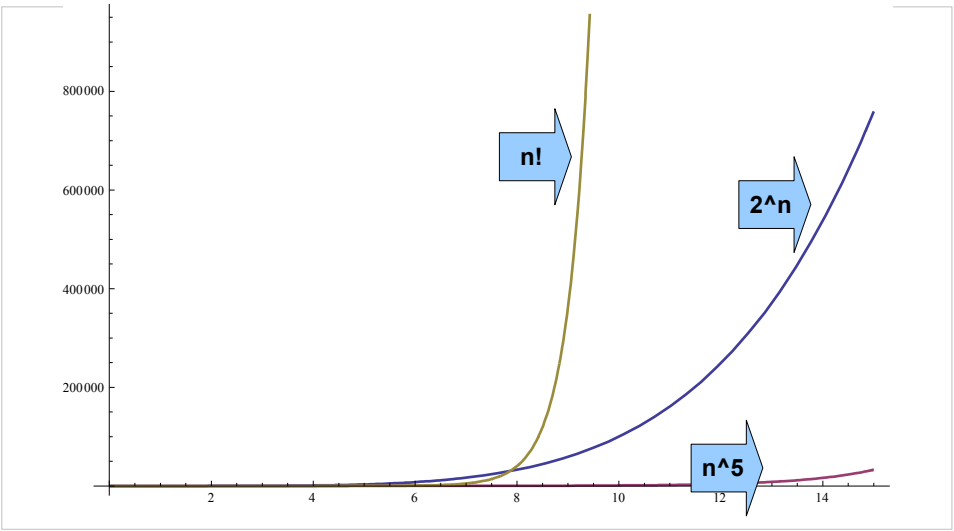
8				5			9	
1		2	4	9		3		
				2	1	6		

My proposed algorithm:

1. Generate a permutation of 1 to n^2 for each row.
2. Once I have a “proposed solution” check (in polynomial time) to see if my solution is valid.
3. If it's not, try a new permutation and go back to step 2.

How fast is it? Well...we can see that there are gonna be (n^2)! permutations for each row. That seems like it might be a little much.

So inefficient it will hurt your brain



8

Ouch! Ok, so if you can't quite see what I've done is graphed three functions. Down here on the bottom is n^5. That is what we would consider “slow” - but still n P.

And then this line here is 2^n. That would be too slow to be part of P. It's exponential, not polynomial.

And then this line here is factorial. See it shooting up past 800,000. This is *slllloww”. This is not gonna work for anything of large size. So guess and check...not good.

Sudoku is an example of a “polynomial time verifiable” problem. For problems like these, it is easy to verify a given solution is correct. But it can be hard to find that solution.

Let us call all such problems “NP” problems.

Ok, this is a key definition. It's so important I'm gonna pause for a minute and let you just soak up the definition. <pause> Does everyone feel they understand what I mean by polynomial time verifiable? I want you to give me the thumbs up if you understand, thumbs down if you don't.

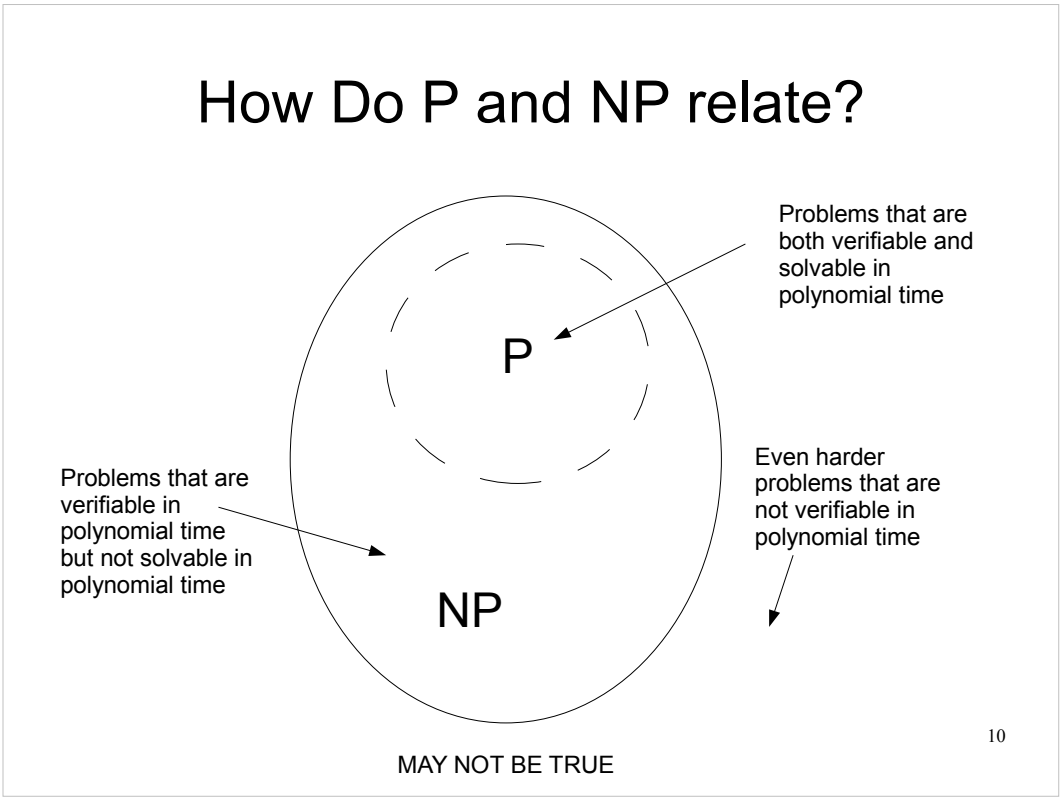
BTW “NP” stands for non-deterministic polynomial time. I'm not gonna be able to explain how that relates to being verifiable right now. Just remember NP is polynomial time verifiable.

Ok, let me give you some other examples of problems in NP. Factoring two numbers – easy to check if two numbers multiply together to give you a 3rd. Hard given the third to find the other two.

Lets say that I give you a list of positive and negative numbers, and I want to know does any subset of these numbers sum to zero? If you know the subset its easy to check – just add them together.

Imagine that I had a list of students with classes and a list of class times. It would be easy to check that no student has two classes at the same time right?

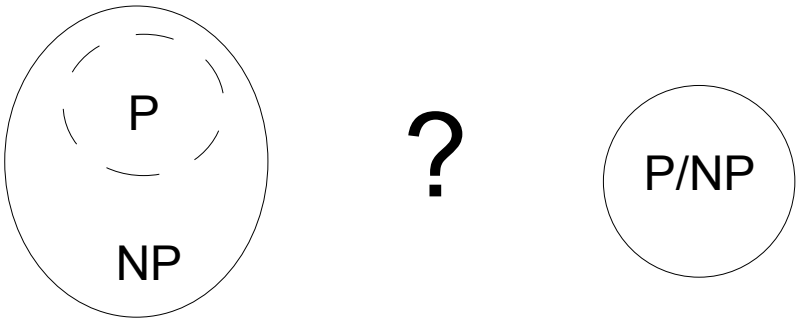
Actually, we could also consider anything in P to be in this list too – right? Easy to check if a list is sorted for example. Which really might get you thinking? How do P and NP relate?



So here's how you might think of it. You've got P – these are your basic run of the mill solvable problems. And then you've got NP, problems that are not solvable but do have verifiers. And then outside that you've got some really hard problems – problems that don't even have polynomial time verifiers. These do exist, but we aren't really going to talk about them.

Only one problem – this diagram might be a lie.

The Most Famous Question in CS: Does P equal NP



ONE OF THESE TWO THINGS IS TRUE

11

We know NP is at least as big as P [gesture at left]. But P could *equal* NP [gesture at right].

What I'm saying is we don't even know 1 problem with a verifier that we can prove can't be solved in polynomial time. Could it be possible that every problem that we can verify the solution in polynomial time, we can solve in polynomial time?

Does P equal NP? This is the question, this is the debate. If you could answer this question, you stand to get a million dollars.

Ok so if you could only remember 1 thing from this lecture I'd remember this. But if you can remember more hold on because I haven't really told you the incredible thing yet. If this was all it was, I don't think anyone would have ever heard of P == NP. Because Cook's theorem blow's everyone's mind.

Where we are going

- DONE:“Polynomial Time Verifiable Problems”
  - Problems that can be solved in polynomial time are in P
  - Problems that are easy to check but hard to solve are in NP
  - Does P = NP is the famous question
- Step 2: Cook's Theorem – how to do any computation with only Boolean logic
- Step 3: Other NP-Complete Problems, and implications

Ok, lets pause for a second and reflect on what we've covered. We know what P is. We know what NP is. We know what a polynomial time verifier is. We know what the question about P/NP means.

[Pause]

Does anyone have a questions here, or isn't rock solid about what we've said thus far.

Work with the person sitting next to you:

Come to a consensus on the solution to the question on the back of your handout

Circulate to help groups. If people seem to be struggling, address the class with a hint.

A New NP Problem: Satisfiability

Given a Boolean equation, determine if there is an assignment of variables that makes the statement true.

(A and B) and not(C and A)

Can someone help me answer this question: what do I need to determine if Satisfiablility is in NP? Can you explain how to do that in this particular case?

Can someone come forward <maybe call on a particular person if you know a likely person from walking around>. How did you represent the rule as a boolean expression?

And do you think this idea could be generalized? Do you think you could express all of the rules of Sodoku in logic like this?

So this means that if we had a solution to Satisfiablility, we could have a solution to Sodoku. Do people see that? In this case, we might say that Sodoku is reducible to Satisifiability. If you solve satisfiability, you solve sodoku.

Cook's Theorem: In Concept

- “Satisfiability” (as we have shown) is a great way to represent complex rule systems
- But what is a computer program but a complex rule system?
- What if we could reduce a verifier (which all NP problems have) to a boolean equation
- If we could do that, then if we could solve satisfiability in polynomial time, we could solve any NP problem in polynomial time

So I want you to read this and just think for a second. Here we have a NP problem that's good a representing rule systems. And what if we used that to represent the rule systems of NP problems.

All we need to make this work is a simple model of the way a computer works – this is called a turing machine and was established long ago.

A Very Simple Programming Language

Only 6 Operations

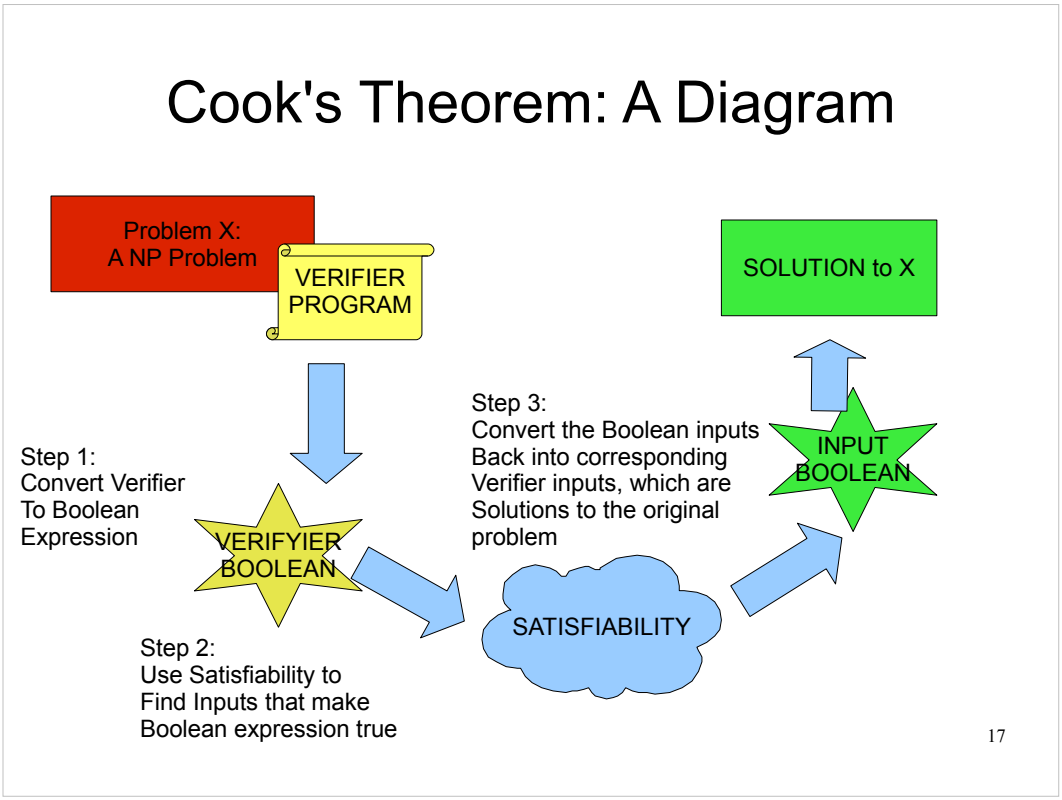
- If current tape has a 1 GOTO a particular state
- If current tape has a 0 GOTO a particular state
- Write 1 on the current tape position
- Write 0 on the current tape position
- Move Tape Forward
- Move Tape Backward

So we have a very simple model of a computer here. This is a like the soduku version of a computer. It has a tape with data, some states which are a program, and some very simple rules.

We might represent the tape as a list of boolean variables. We might have a bunch of variables that indicate which state you are on (write on board as you discuss these things). And then we might have rules to the effect of "if you are stateX at time t and the current position is 1" then at time t + 1 you must be in stateY.

Would this work? Yes.

But as it turns out any verifier can be written in this simple programming language.



Let me step back from the abstraction a bit and come from a different direction. What is Cook proposing. Well, first imagine you had a polynomial time solution to satisfiability, and you wanted to solve some other problem in NP.

3 steps...(go through the diagram)

But I want to point out, this is hypothetical. Not the part about converting verifiers to booleans. That works. The real issue is that we don't have a polynomial time solution for satisfiability.

But, if we did (and here's the kicker) P = NP. Right? Because we know we could use satisifibility to solve any NP problem. So if you want to get that prize, that million bucks, all you have to do is find a polynomial time solution for satisfiability.

NP-Completeness

We call Satisfiability NP-Complete. That is, if we could solve Satisfiability we could solve any problem in NP. If Satisfiability is in P, P = NP. If P != NP, Satisifiability must be one of the problems not in P.

8				5			9	
1		2	4	9		3		
				2	1	6		
		1			5		2	
	3					5		7
5	9				4			
			9	8				6
9			5		3	8		4
		8			6			

Are there other NP-Complete problems?

In short: yes. There are actually many NP-Complete problems. Sodoku is actually a NP complete problem, so if any of you actually found a polynomial time solution to it in that first exersize, you've just wom a million dollars. There are problems that have to do with graphs. There are problems to do with scheduling.

Much later in your CS career, you'll learn how to prove a problem is NP complete. The reason to do this is because usually these problems are considered intractable. If it's not NP complete it's possible you can find a tricky solution and the problem is in P. If it's NP complete...well, you're not likely to win a turning award this afternoon.

All of these problems have a certain feel about them, much like soduku: they seem tough, but we have an intuition that surely there must be some trick to solving them efficiently. It remains to be seen if our intution is right or wrong on this count.

A polynomial time solution to satisfiability would be an important result because:

1. It would conclusively show that P != NP
2. It would make other NP problems like factoring numbers, solvable in polynomial time
3. It would disprove the hypothesis that satisfiability was NP-complete
4. Satisfiability is a general problem frequently encountered in industy.

All the multiple choice questions are designed for the same format. Give the students a chance to raise 1-4 fingers, encouraging everyone to at least make a guess.

Then (assuming time looks OK) go through each choice and explain why it was wrong or right

Which of these is a component of Cook's Theorem?

1. A method to convert a program to a boolean expression
2. A bound on the runtime of the Satisfiability problem
3. A proof that not all NP problems have a polynomial time verifier
4. A way to exploit the high speed of Satisfiability in other contexts

As a professional programmer, why would you want to be able to prove a problem is NP-complete?

1. NP-complete problems are more efficient
2. Only NP-complete problems can benefit from the techniques of Cook's Theorem
3. To show that they can be reduced to satisfiability
4. To avoid them because NP-complete problems are intractable

Where we are going

- DONE:“Polynomial Time Verifiable Problems”
  - NP is the class of Polynomial Time verifiable problems
- DONE: Cook's Theorem
  - Satisfiability is a NP problem that determines if a boolean equation can be satisfied
  - It is NP Complete, meaning that a polynomial time solution would work for every problem in NP
  - The solution is created by converting the verifier into a Boolean equation
- Step 3: Other NP-Complete Problems, and implications

So this is where we are. Hopefully you know have a basic intuitive idea of Cook's theorem and what NP complete problems are

I'm going to pause a minute and let you reflect. If you are unsure of anything up there. Raise you hand.

Final Exercise: Brainstorming

Why is P/NP an Important Problem?
What are the implications of P/NP?

In this activity the professor should adopt the role of a brain stormer, writing all student suggestions on the board. Encouraging elaboration along the lines of general applicability of NP problems, implications if P = NP, and understanding efficiency.

Near the end of the activity the professor should move into summary mode. Which of these do we think is most important? Why?

Finally, present a summing up that brings the various student summaries together.