CSE 373: P vs NP; reductions

Michael Lee

Wednesday, Mar 7, 2018

Warmup

Remind your neighbor:

- What is a decision-problem?
- What is P, EXP, and NP?

Warmup

Remind your neighbor:

- What is a decision-problem?
  A yes-or-no question
- What is P, EXP, and NP?

Warmup

Remind your neighbor:

- What is a decision-problem?
  A yes-or-no question
- What is P, EXP, and NP?
  1. P is the set of all decision problems that can be *solved* in worst-case *polynomial* time

Warmup

Remind your neighbor:

- What is a decision-problem?
  A yes-or-no question
- What is P, EXP, and NP?
  1. P is the set of all decision problems that can be *solved* in worst-case *polynomial* time
  2. EXP is the set of all decision problems that can be *solved* in worst-case *exponential* time

Warmup

Remind your neighbor:

- What is a decision-problem?

  A yes-or-no question
- What is P, EXP, and NP?

  1. P is the set of all decision problems that can be *solved* in worst-case *polynomial* time
  2. EXP is the set of all decision problems that can be *solved* in worst-case *exponential* time
  3. NP is the set of all decision problems where we can *verify* all "yes" answers in worst-case *polynomial* time

Final

Final logistics:

- Thursday, March 15
- 2:30 to 4:20
- Gowen 301

Final

Final logistics:

- Thursday, March 15
- 2:30 to 4:20
- Gowen 301

If you need to take the final at a different date:

- If you've already sent me an email, no action needed
- Otherwise, send me an email by the end of today

Final

Final logistics:

- Thursday, March 15
- 2:30 to 4:20
- Gowen 301

If you need to take the final at a different date:

- If you've already sent me an email, no action needed
- Otherwise, send me an email by the end of today

Review sessions:

- Monday, Mar 12: EEB 125, 4:30 to 6:30
- Tuesday, Mar 13: EEB 105, 4:30 to 6:30

Final

The final will be cumulative, but skewed towards new material.

Post-midterm topics:

1. Heaps

Final

The final will be cumulative, but skewed towards new material.

Post-midterm topics:

1. Heaps
2. Sorting, basic divide-and-conquer

Final

The final will be cumulative, but skewed towards new material.

Post-midterm topics:

1. Heaps
2. Sorting, basic divide-and-conquer
3. The tree method and the master method

Final

The final will be cumulative, but skewed towards new material.

Post-midterm topics:

1. Heaps
2. Sorting, basic divide-and-conquer
3. The tree method and the master method
4. Graphs

Final

The final will be cumulative, but skewed towards new material.

Post-midterm topics:

1. Heaps
2. Sorting, basic divide-and-conquer
3. The tree method and the master method
4. Graphs
    - Definitions
    - Representation
    - Traversal
    - Dijkstra's
    - Topological sort
    - MSTs (Prim, Kruskal, disjoint sets)

Final

The final will be cumulative, but skewed towards new material.

Post-midterm topics:

1. Heaps
2. Sorting, basic divide-and-conquer
3. The tree method and the master method
4. Graphs
   - Definitions
   - Representation
   - Traversal
   - Dijkstra's
   - Topological sort
   - MSTs (Prim, Kruskal, disjoint sets)
5. P and NP

Final

The final will be cumulative, but skewed towards new material.

Pre-midterm topics:

Final

The final will be cumulative, but skewed towards new material.

Pre-midterm topics:

1. Asymptotic analysis, modeling code as equations

Final

The final will be cumulative, but skewed towards new material.

Pre-midterm topics:

1. Asymptotic analysis, modeling code as equations
2. Anything related to dictionaries

Final

The final will be cumulative, but skewed towards new material.

Pre-midterm topics:

1. Asymptotic analysis, modeling code as equations
2. Anything related to dictionaries
3. Caching and locality

Final

General study tips for mechanical problems:

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly
2. Invent your own problems and check them using online tools

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly
2. Invent your own problems and check them using online tools

General study tips for non-mechanical problems:

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly
2. Invent your own problems and check them using online tools

General study tips for non-mechanical problems:

1. Do tons of practice

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly
2. Invent your own problems and check them using online tools

General study tips for non-mechanical problems:

1. Do tons of practice
2. Minor differences matter; make sure you ask about them

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly
2. Invent your own problems and check them using online tools

General study tips for non-mechanical problems:

1. Do tons of practice
2. Minor differences matter; make sure you ask about them
3. Definitions are important; make sure you know them

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly
2. Invent your own problems and check them using online tools

General study tips for non-mechanical problems:

1. Do tons of practice
2. Minor differences matter; make sure you ask about them
3. Definitions are important; make sure you know them
4. For each data structure and algorithm we've studied, try writing a document summarizing (a) the high-level idea of how to implement them and (b) the best, average, and worst-case runtimes.

Final

General study tips for mechanical problems:

1. Drill until you can complete them very quickly
2. Invent your own problems and check them using online tools

General study tips for non-mechanical problems:

1. Do tons of practice
2. Minor differences matter; make sure you ask about them
3. Definitions are important; make sure you know them
4. For each data structure and algorithm we've studied, try writing a document summarizing (a) the high-level idea of how to implement them and (b) the best, average, and worst-case runtimes.
5. Think about what would happen if you were to tweak some aspect of a data structure or algorithm

Final

General tips when asked to analyze algorithms or code:

Final

General tips when asked to analyze algorithms or code:

1. Don't make assumptions about what the code is doing, actually read it

Final

General tips when asked to analyze algorithms or code:

1. Don't make assumptions about what the code is doing, actually read it
2. Try mentally running the code on *specific* examples

Final

General tips when asked to analyze algorithms or code:

1. Don't make assumptions about what the code is doing, actually read it
2. Try mentally running the code on *specific* examples

General tips when asked to write pseudocode:

Final

General tips when asked to analyze algorithms or code:

1. Don't make assumptions about what the code is doing, actually read it
2. Try mentally running the code on *specific* examples

General tips when asked to write pseudocode:

1. Keep a mental list of every data structure and algo we've studied. When stuck, go through that list one-by-one and try and find one that seems applicable

Final

General tips when asked to analyze algorithms or code:

1. Don't make assumptions about what the code is doing, actually read it
2. Try mentally running the code on *specific* examples

General tips when asked to write pseudocode:

1. Keep a mental list of every data structure and algo we've studied. When stuck, go through that list one-by-one and try and find one that seems applicable
2. Try writing an algorithm that works on a specific example first, then figure out how to generalize.

Final

Syllabus change:

Final

Syllabus change:

Previously:

- Midterm was 20% of grade
- Final was 20% of grade

Final

Syllabus change:

Previously:

- Midterm was 20% of grade
- Final was 20% of grade

Now:

- Your lowest-scoring exam will be 15% of grade
- Your highest-scoring exam will be 25% of grade

Recap

Last time:

Recap

Last time:

- Introduced the idea of decision problems and complexity classes

Recap

Last time:

- Introduced the idea of decision problems and complexity classes
- Introduced the complexity classes P and EXP

Recap

Last time:

- Introduced the idea of decision problems and complexity classes
- Introduced the complexity classes P and EXP
- Found some (useful!) problems are, unfortunately, in EXP

Recap

Last time:

- Introduced the idea of decision problems and complexity classes
- Introduced the complexity classes P and EXP
- Found some (useful!) problems are, unfortunately, in EXP
- But many of those problems are also in NP!

Recap

Last time:

- Introduced the idea of decision problems and complexity classes
- Introduced the complexity classes P and EXP
- Found some (useful!) problems are, unfortunately, in EXP
- But many of those problems are also in NP!
- Question: if there are problems where we can *verify* answers efficiently, does that mean we can also *find* answers efficiently?

Is CIRCUIT-SAT in NP?

Question: is `CIRCUIT-SAT` in NP?

Is CIRCUIT-SAT in NP?

Question: is `CIRCUIT-SAT` in NP?

`CIRCUIT-SAT`

T

Given a boolean expression such as "`a && (b || c)`" and the truth values for **some** of the variables, is there a way to set the remaining variables so that the output is T?

Is CIRCUIT-SAT in NP?

Question: is `CIRCUIT-SAT` in NP?

`CIRCUIT-SAT`

Given a boolean expression such as "`a && (b || c)`" and the truth values for **some** of the variables, is there a way to set the remaining variables so that the output is T?

Step 1: Assume you have a magical solver, and it said "yes" for some boolean expression B.

Is CIRCUIT-SAT in NP?

Question: is `CIRCUIT-SAT` in NP?

`CIRCUIT-SAT`

Given a boolean expression such as "`a && (b || c)`" and the truth values for **some** of the variables, is there a way to set the remaining variables so that the output is T?

Step 1: Assume you have a magical solver, and it said "yes" for some boolean expression B.

Step 2: Three questions to answer.

1. How do we modify the solver so it returns a convincing certificate for B?
2. How do we check the certificate, whatever it is?
3. Does our verifier actually run in polynomial time?

Is CIRCUIT-SAT in NP?

Step 2a: How do we modify the solver so it returns a convincing certificate?

Is CIRCUIT-SAT in NP?

Step 2a: How do we modify the solver so it returns a convincing certificate?

Idea: return a map of the variable assignments!

`{a=true, b=false, c=true, d=false, ...}`

Is CIRCUIT-SAT in NP?

Step 2a: How do we modify the solver so it returns a convincing certificate?

Idea: return a map of the variable assignments!

`{a=true, b=false, c=true, d=false, ...}`

2b: How do we check the certificate, whatever it is?

Is CIRCUIT-SAT in NP?

Step 2a: How do we modify the solver so it returns a convincing certificate?

Idea: return a map of the variable assignments!

`{a=true, b=false, c=true, d=false, ...}`

2b: How do we check the certificate, whatever it is?

Idea: try evaluating the expression!

```
boolean verifyCiruitSat(BooleanAst B, Dictionary<String, Boolean> certificate) {
    return evaluateExpr(B, certificate);
}
private boolean evaluateExpr(B, certificate) {
    // Do something similar to toDoubleHelper, back from project 1
}
```

Is CIRCUIT-SAT in NP?

Step 2a: How do we modify the solver so it returns a convincing certificate?

Idea: return a map of the variable assignments!

`{a=true, b=false, c=true, d=false, ...}`

2b: How do we check the certificate, whatever it is?

Idea: try evaluating the expression!

```
boolean verifyCiruitSat(BooleanAst B, Dictionary<String, Boolean> certificate) {
    return evaluateExpr(B, certificate);
}
private boolean evaluateExpr(B, certificate) {
    // Do something similar to toDoubleHelper, back from project 1
}
```

2c: Does our verifier actually run in polynomial time?

Is CIRCUIT-SAT in NP?

Step 2a: How do we modify the solver so it returns a convincing certificate?

Idea: return a map of the variable assignments!

`{a=true, b=false, c=true, d=false, ...}`

2b: How do we check the certificate, whatever it is?

Idea: try evaluating the expression!

```
boolean verifyCiruitSat(BooleanAst B, Dictionary<String, Boolean> certificate) {
    return evaluateExpr(B, certificate);
}
private boolean evaluateExpr(B, certificate) {
    // Do something similar to toDoubleHelper, back from project 1
}
```

2c: Does our verifier actually run in polynomial time?

Yes: we visit each node and edge in the tree a constant number of times.

Ranking problems

So far, we've talked about *classifying* problems into classes.

Ranking problems

So far, we've talked about *classifying* problems into classes.

Is there some way of "*ranking*" problems by difficulty?

Ranking problems

So far, we've talked about *classifying* problems into classes.

Is there some way of "*ranking*" problems by difficulty?

For example, is...

- 2-COLOR easier or harder then 3-COLOR?

Ranking problems

So far, we've talked about *classifying* problems into classes.

Is there some way of "*ranking*" problems by difficulty?

For example, is...

- 2-COLOR easier or harder then 3-COLOR?
- 3-COLOR easier or harder then CIRCUIT-SAT?

Ranking problems

Yes, using *reductions*.

Ranking problems

Yes, using *reductions*.

a >= b

Reductions

Given two decision problems A and B, we can show that A is "harder then or the same difficulty as" B by...

Ranking problems

Yes, using *reductions*.

Reductions

Given two decision problems A and B, we can show that A is "harder then or the same difficulty as" B by...

1. Assuming we have some magical solver for A

Ranking problems

Yes, using *reductions*.

Reductions

Given two decision problems A and B, we can show that A is "harder then or the same difficulty as" B by...

1. Assuming we have some magical solver for A
2. Create an algorithm which calls the magical solver to solve B

Ranking problems

Yes, using *reductions*.

Reductions

Given two decision problems A and B, we can show that A is "harder then or the same difficulty as" B by...

1. Assuming we have some magical solver for A
2. Create an algorithm which calls the magical solver to solve B

Core ideas: If solving A lets us also solve B, then...

- A was "harder then" (or the same as) B

Ranking problems

Yes, using *reductions*.

Reductions

Given two decision problems A and B, we can show that A is "harder then or the same difficulty as" B by...

1. Assuming we have some magical solver for A
2. Create an algorithm which calls the magical solver to solve B

Core ideas: If solving A lets us also solve B, then...

- A was "harder then" (or the same as) B
- The B was really a special case of A all along!

Ranking problems

Yes, using *reductions*.

Reductions

Given two decision problems A and B, we can show that A is "harder then or the same difficulty as" B by...

1. Assuming we have some magical solver for A
2. Create an algorithm which calls the magical solver to solve B

Core ideas: If solving A lets us also solve B, then...

- A was "harder then" (or the same as) B
- The B was really a special case of A all along!
- We've *reduced* the number of distinct problems in the world by one.

Showing 2-COLOR reduces to 3-COLOR

We want to show that 2-COLOR reduces to 3-COLOR: that 3-COLOR is "harder then" 2-COLOR.

Showing 2-COLOR reduces to 3-COLOR

We want to show that 2-COLOR reduces to 3-COLOR: that 3-COLOR is "harder then" 2-COLOR.

Step 1: Assume we have a magical solver for 2-COLOR

Showing 2-COLOR reduces to 3-COLOR

We want to show that 2-COLOR reduces to 3-COLOR: that 3-COLOR is “harder then” 2-COLOR.

Step 1: Assume we have a magical solver for ~~2~~3-COLOR

Step 2: Using this magical solver, how do we solve an instance of 2-COLOR?

Showing 2-COLOR reduces to 3-COLOR

We want to show that 2-COLOR reduces to 3-COLOR: that 3-COLOR is "harder then" 2-COLOR.

Step 1: Assume we have a magical solver for 2-COLOR

Step 2: Using this magical solver, how do we solve an instance of 2-COLOR?

Answer:

1. Start by adding a new vertex to the graph
2. Connect this vertex to all other nodes
3. Give this vertex some color. This forces all other vertices to have a only one of two colors!
4. Run the solver for 3-COLOR, return the result

Showing problems are the same

New question: How do we show two problems are the same?

Showing problems are the same

New question: How do we show two problems are the same?

Intuition:

- To show two numbers a and b are the same, we can show $a \geq b$ and $a \leq b$.

Showing problems are the same

New question: How do we show two problems are the same?

Intuition:

- To show two numbers a and b are the same, we can show $a \geq b$ and $a \leq b$.
- To show two functions $f(n)$ and $g(n)$ are asymptotically the same, we can show that $f(n)$ both dominates and is dominated by $g(n)$

Showing problems are the same

New question: How do we show two problems are the same?

Intuition:

- To show two numbers a and b are the same, we can show $a \geq b$ and $a \leq b$.
- To show two functions $f(n)$ and $g(n)$ are asymptotically the same, we can show that $f(n)$ both dominates and is dominated by $g(n)$
- To show two decision problems A and B are the same, we can show that A reduces to B and B reduces A!

LONG-PATH and HAM-PATH

LONG-PATH

Given a graph G and some integer k, does G contain some path that uses k edges?

LONG-PATH and HAM-PATH

LONG-PATH

Given a graph G and some integer k, does G contain some path that uses k edges?

HAM-PATH

Given a graph G, does G have a path that visits every vertex?

LONG-PATH and HAM-PATH

LONG-PATH

Given a graph G and some integer k, does G contain some path that uses k edges?

HAM-PATH

Given a graph G, does G have a path that visits every vertex?

Goal: Show that LONG-PATH and HAM-PATH are the same

LONG-PATH and HAM-PATH

LONG-PATH

Given a graph G and some integer k, does G contain some path that uses k edges?

HAM-PATH

Given a graph G, does G have a path that visits every vertex?

Goal: Show that LONG-PATH and HAM-PATH are the same

Step 1:

Reduce HAM-PATH to LONG-PATH

Note: paths can't revisit the same node

LONG-PATH and HAM-PATH

LONG-PATH

Given a graph G and some integer k, does G contain some path that uses k edges?

HAM-PATH

Given a graph G, does G have a path that visits every vertex?

Goal: Show that LONG-PATH and HAM-PATH are the same

Step 1:

Reduce HAM-PATH to LONG-PATH

```
boolean hamPathSolver(G) {
    return longPathSolver(G, |V| - 1)
}
```

LONG-PATH and HAM-PATH

LONG-PATH

Given a graph G and some integer k, does G contain some path that uses k edges?

HAM-PATH

Given a graph G, does G have a path that visits every vertex?

Goal: Show that LONG-PATH and HAM-PATH are the same

Step 1:

Reduce HAM-PATH to LONG-PATH

```
boolean hamPathSolver(G) {
    return longPathSolver(G, |V| - 1)
}
```

Step 2:

Reduce LONG-PATH to HAM-PATH

LONG-PATH and HAM-PATH

LONG-PATH

Given a graph G and some integer k, does G contain some path that uses k edges?

HAM-PATH

Given a graph G, does G have a path that visits every vertex?

Goal: Show that LONG-PATH and HAM-PATH are the same

Step 1:

Reduce HAM-PATH to LONG-PATH

```
boolean hamPathSolver(G) {
    return longPathSolver(G, |V| - 1)
}
```

Step 2:

Reduce LONG-PATH to HAM-PATH

```
boolean longPathSolver(G, k) {
    for (G2=(v1, v2, ..., vk) : G):
        if (hamPathSolver(G2)):
            return true;
    return false;
}
```

Equivalent problems

Punchline: `HAM-PATH` and `LONG-PATH` are actually the same problem in disguise!

Equivalent problems

Punchline: `HAM-PATH` and `LONG-PATH` are actually the same problem in disguise!

Question: Are there other problems that are secretly the same problem in disguise?

Equivalent problems

Punchline: `HAM-PATH` and `LONG-PATH` are actually the same problem in disguise!

Question: Are there other problems that are secretly the same problem in disguise?

Yes! It turns out that...

- `CIRCUIT-SAT`
- `3-COLOR`
- `HAM-PATH`
- `LONG-PATH`

...are all the same problem.

NP-HARD and NP-COMPLETE

Is there some problem that's "harder then or same as" all of the problems we've seen so far?

NP-HARD and NP-COMPLETE

Is there some problem that's "harder then or same as" all of the problems we've seen so far?

Yes! For example, `CIRCUIT-SAT` (and therefore `HAM-PATH` and `LONG-PATH` and `3-COLOR`).

NP-HARD and NP-COMPLETE

Is there some problem that's "harder then or same as" all of the problems we've seen so far?

Yes! For example, `CIRCUIT-SAT` (and therefore `HAM-PATH` and `LONG-PATH` and `3-COLOR`).

NP-HARD

A decision problem is NP-HARD if that decision problem is "harder then or as hard as" any other problem in NP.

NP-HARD and NP-COMPLETE

Is there some problem that's "harder then or same as" all of the problems we've seen so far?

Yes! For example, `CIRCUIT-SAT` (and therefore `HAM-PATH` and `LONG-PATH` and `3-COLOR`).

NP-HARD

A decision problem is NP-HARD if that decision problem is "harder then or as hard as" any other problem in NP.

Alternative phrasing: if every single decision problem in NP reduces to X, then X is NP-HARD.

NP-HARD and NP-COMPLETE

Is there some problem that's "harder then or same as" all of the problems we've seen so far?

Yes! For example, `CIRCUIT-SAT` (and therefore `HAM-PATH` and `LONG-PATH` and `3-COLOR`).

NP-HARD

A decision problem is NP-HARD if that decision problem is "harder then or as hard as" any other problem in NP.

Alternative phrasing: if every single decision problem in NP reduces to X, then X is NP-HARD.

NP-COMPLETE

A decision problem is NP-COMPLETE if it is both in NP and in NP-HARD.

NP-HARD and NP-COMPLETE

Punchline: If we have a way of solving any NP-HARD problem, we have a way of solving *every* problem we've looked at so far.

NP-HARD and NP-COMPLETE

How do these relate?

NP-HARD and NP-COMPLETE

How do these relate?

How do all relate to P?

Is P a subset of EXP?

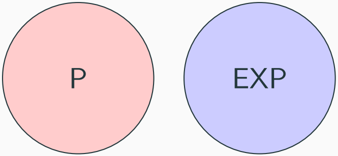
Last time, we asked if P is a subset of EXP.

Is P a subset of EXP?

Last time, we asked if P is a subset of EXP.

Answer 1: The sets are disjoint

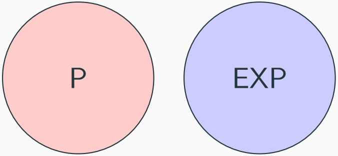
E.g. if a problem is in P, it's not in EXP.



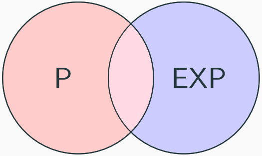
Is P a subset of EXP?

Last time, we asked if P is a subset of EXP.

Answer 1: The sets are disjoint
E.g. if a problem is in P, it's not in EXP.



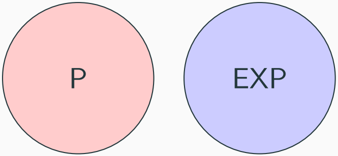
Answer 2: The sets overlap
E.g. some, but not all problems in P are in EXP



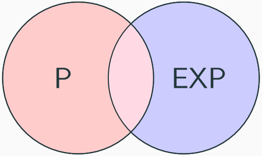
Is P a subset of EXP?

Last time, we asked if P is a subset of EXP.

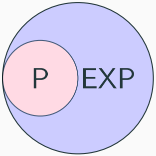
Answer 1: The sets are disjoint
E.g. if a problem is in P, it's not in EXP.



Answer 2: The sets overlap
E.g. some, but not all problems in P are in EXP



Answer 3: P is a subset of EXP
All problems in P are also in EXP

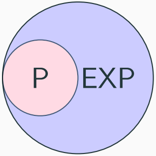


Is P a subset of EXP?

Last time, we asked if P is a subset of EXP.

It turns out, yes, P is indeed a subset of EXP:

Answer 3: P is a subset of EXP
All problems in P are also in EXP

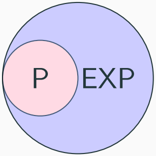


Is P a subset of EXP?

Last time, we asked if P is a subset of EXP.

It turns out, yes, P is indeed a subset of EXP:

Answer 3: P is a subset of EXP
All problems in P are also in EXP



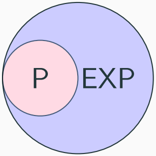
Reason: EXP is the set of decision problems where there exists an algorithm that solves the problem in *worst-case exponential time*.

Is P a subset of EXP?

Last time, we asked if P is a subset of EXP.

It turns out, yes, P is indeed a subset of EXP:

Answer 3: P is a subset of EXP
All problems in P are also in EXP



Reason: EXP is the set of decision problems where there exists an algorithm that solves the problem in *worst-case exponential time*.

So, if we can find a polynomial-time algorithm to a problem, we can definitely find an exponential one!

Is P a subset of NP?

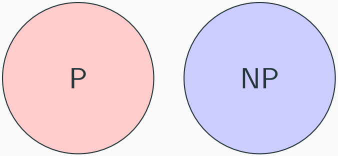
New question: is a P a subset of **NP**?

Is P a subset of NP?

New question: is a P a subset of **NP**?

Answer 1: The sets are disjoint

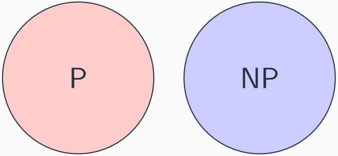
E.g. if a problem is in P, it's not in NP.



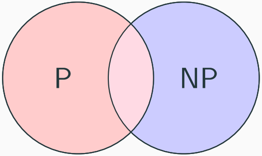
Is P a subset of NP?

New question: is a P a subset of **NP**?

Answer 1: The sets are disjoint
E.g. if a problem is in P, it's not in NP.



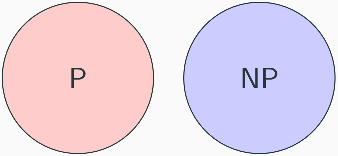
Answer 2: The sets overlap
E.g. some, but not all problems in P are in NP



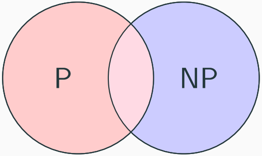
Is P a subset of NP?

New question: is a P a subset of **NP**?

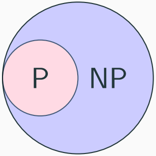
Answer 1: The sets are disjoint
E.g. if a problem is in P, it's not in NP.



Answer 2: The sets overlap
E.g. some, but not all problems in P are in NP



Answer 3: P is a subset of NP
All problems in P are also in NP

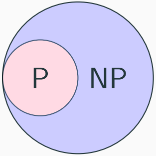


Is P a subset of NP?

New question: is a P a subset of **NP**?

It turns out, yes.

Answer 3: P is a subset of NP
All problems in P are also in NP



Is P a subset of NP?

Reason: Let's say we have some decision problem X.

Step 1: Assume we have a magical solver for X, and it said "yes" for some input.

Is P a subset of NP?

Reason: Let's say we have some decision problem X.

Step 1: Assume we have a magical solver for X, and it said "yes" for some input.

Step 2: Three questions to answer.

Is P a subset of NP?

Reason: Let's say we have some decision problem X.

Step 1: Assume we have a magical solver for X, and it said "yes" for some input.

Step 2: Three questions to answer.

1. How do make the solver so it returns a convincing certificate?

Is P a subset of NP?

Reason: Let's say we have some decision problem X.

Step 1: Assume we have a magical solver for X, and it said "yes" for some input.

Step 2: Three questions to answer.

1. How do make the solver so it returns a convincing certificate? One possible certificate: return the string "¯_(ツ)_/¯".

Is P a subset of NP?

Reason: Let's say we have some decision problem X.

Step 1: Assume we have a magical solver for X, and it said "yes" for some input.

Step 2: Three questions to answer.

1. How do make the solver so it returns a convincing certificate? One possible certificate: return the string "¯_(ツ)_/¯".
2. How do we check the certificate, whatever it is?

Is P a subset of NP?

Reason: Let's say we have some decision problem X.

Step 1: Assume we have a magical solver for X, and it said "yes" for some input.

Step 2: Three questions to answer.

1. How do make the solver so it returns a convincing certificate?
   One possible certificate: return the string "¯_(ツ)_/¯".
2. How do we check the certificate, whatever it is?
   Idea: just *ignore* the certificate

```
boolean verifyX(input, certificate) {
    return solverX(input);
}
```

Is P a subset of NP?

Reason: Let's say we have some decision problem X.

Step 1: Assume we have a magical solver for X, and it said "yes" for some input.

Step 2: Three questions to answer.

1. How do make the solver so it returns a convincing certificate?
   One possible certificate: return the string "¯_(ツ)_/¯".
2. How do we check the certificate, whatever it is?
   Idea: just *ignore* the certificate

```
boolean verifyX(input, certificate) {
    return solverX(input);
}
```

3. Does our verifier actually run in polynomial time?
   Yep. If X was originally in P, then we know by definition `solverX` runs in polynomial time.

Is P a subset of NP?

Punchline: For any problem in P, we can build a verifier by just re-using the solver!

Is P = NP?

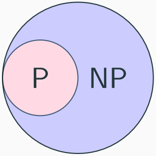
Third question: is P = NP?

Is P = NP?

Third question: is P = NP?

Answer 1: No

P is a subset of NP, but that's it.

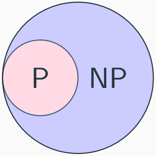


Is P = NP?

Third question: is P = NP?

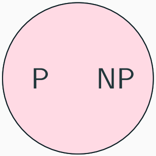
Answer 1: No

P is a subset of NP, but that's it.



Answer 2: Yes

Not only is a P a subset of NP, they're exactly the same

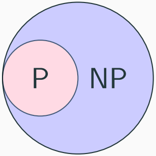


Is P = NP?

Third question: is P = NP?

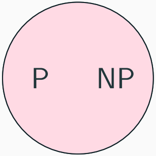
Answer 1: No

P is a subset of NP, but that's it.



Answer 2: Yes

Not only is a P a subset of NP, they're exactly the same

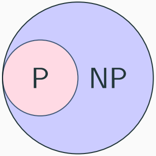


Answer: We don't know.

What if $P \neq NP$?

Answer 1: No

P is a subset of NP, but that's it.

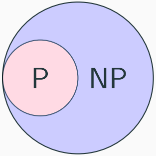


What if P $\neq$ NP?

What if $P \neq NP$?

Answer 1: No

P is a subset of NP, but that's it.

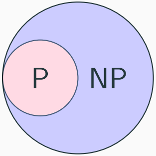


- Have your name be immortalized in CS textbooks forever

What if $P \neq NP$?

Answer 1: No

P is a subset of NP, but that's it.

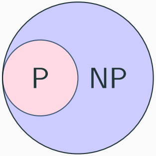


- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem

What if $P \neq NP$?

Answer 1: No

P is a subset of NP, but that's it.



- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem
- The world otherwise looks the same

What if P $\neq$ NP?

If P $\neq$ NP, and we have an NP problem, what do we do?

- Try and find approximate solutions

What if P $\neq$ NP?

If P $\neq$ NP, and we have an NP problem, what do we do?

- Try and find approximate solutions
- Use probabilistic algorithms

What if P $\neq$ NP?

If P $\neq$ NP, and we have an NP problem, what do we do?

- Try and find approximate solutions
- Use probabilistic algorithms
- Use solvers that work efficiently on many (but not all!) instances of NP-COMPLETE problems.
  (E.g. programs like z3, which solve CIRCUIT-SAT)

What if P $\neq$ NP?

If P $\neq$ NP, and we have an NP problem, what do we do?

- Try and find approximate solutions
- Use probabilistic algorithms
- Use solvers that work efficiently on many (but not all!) instances of NP-COMPLETE problems.
  (E.g. programs like z3, which solve CIRCUIT-SAT)
- Find a way of reducing your problem into some famous NP-HARD problem and use a solver

What if P $\neq$ NP?

If P $\neq$ NP, and we have an NP problem, what do we do?

- Try and find approximate solutions
- Use probabilistic algorithms
- Use solvers that work efficiently on many (but not all!) instances of NP-COMPLETE problems.
  (E.g. programs like z3, which solve CIRCUIT-SAT)
- Find a way of reducing your problem into some famous NP-HARD problem and use a solver
- Crowdsource. Observation: lots of games are actually NP (e.g. sudoku).
  Actual example: Foldit, a protein folding "game"

What if P $\neq$ NP?

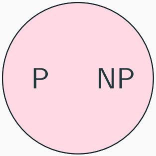
If P $\neq$ NP, and we have an NP problem, what do we do?

- Try and find approximate solutions
- Use probabilistic algorithms
- Use solvers that work efficiently on many (but not all!) instances of NP-COMPLETE problems.
  (E.g. programs like z3, which solve CIRCUIT-SAT)
- Find a way of reducing your problem into some famous NP-HARD problem and use a solver
- Crowdsource. Observation: lots of games are actually NP (e.g. sudoku).
  Actual example: Foldit, a protein folding "game"
- Something something quantum computing? (Lots of caveats, not practical right now, doesn't solve everything, even if they work.)

What if P = NP?

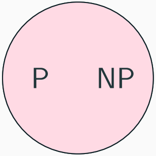
What if $P = NP$?

What if this is reality?



What if $P = NP$?

What if this is reality?



AND what if we have an efficient way of solving any NP-COMPLETE problem?

What if P = NP?

- Have your name be immortalized in CS textbooks forever

What if P = NP?

- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem

What if P = NP?

- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem
- Finding a way of generating a proof of anything (assuming the proof is a reasonable length)

What if P = NP?

- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem
- Finding a way of generating a proof of anything (assuming the proof is a reasonable length)
- Win 5 million *more* dollars for solving the *remaining* Millenium Prize problems

What if P = NP?

- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem
- Finding a way of generating a proof of anything (assuming the proof is a reasonable length)
- Win 5 million *more* dollars for solving the *remaining* Millenium Prize problems
- Crack all of modern encryption, and have *all* the dollars

What if P = NP?

- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem
- Finding a way of generating a proof of anything (assuming the proof is a reasonable length)
- Win 5 million *more* dollars for solving the *remaining* Millenium Prize problems
- Crack all of modern encryption, and have *all* the dollars
- Crack all of modern encryption, and have access to all information, public or private

What if P = NP?

- Have your name be immortalized in CS textbooks forever
- Win 1 million dollars for solving a Millenium Prize problem
- Finding a way of generating a proof of anything (assuming the proof is a reasonable length)
- Win 5 million *more* dollars for solving the *remaining* Millenium Prize problems
- Crack all of modern encryption, and have *all* the dollars
- Crack all of modern encryption, and have access to all information, public or private
- Literally cure cancer