Tractability

15-110 – Wednesday 10/23

Quizlet

Announcements

- Hw4 due **Monday**
  - If you haven't started yet, start now!!

- Mid-semester grades

Learning Goals

- Identify **brute force approaches** to common problems that run in **O(n!)** or **$O(2^n)$**, including solutions to **Travelling Salesperson, puzzle-solving, subset sum, Boolean satisfiability**, and **exam scheduling**

- Define the complexity classes **P** and **NP** and explain why these classes are important

- Identify whether an algorithm is **tractable** or **intractable**, and whether it is in **P**, **NP,** or **neither** complexity class

- Use **heuristics** to find good-enough solutions to NP problems in polynomial time

Big Idea: What *is* Efficient?

As we wrap up the unit on data structures and efficiency, we still need to answer a big question: **can all algorithms be made efficient?** And, importantly, **what does it mean to be efficient?**

To answer these questions, we'll consider a collection of important computational problems. While considering these problems, ask yourself: how efficient are these solutions? Could we make them better?

Computationally Difficult Problems

There are many problems that are actually ***really hard*** **for computers to solve quickly**.

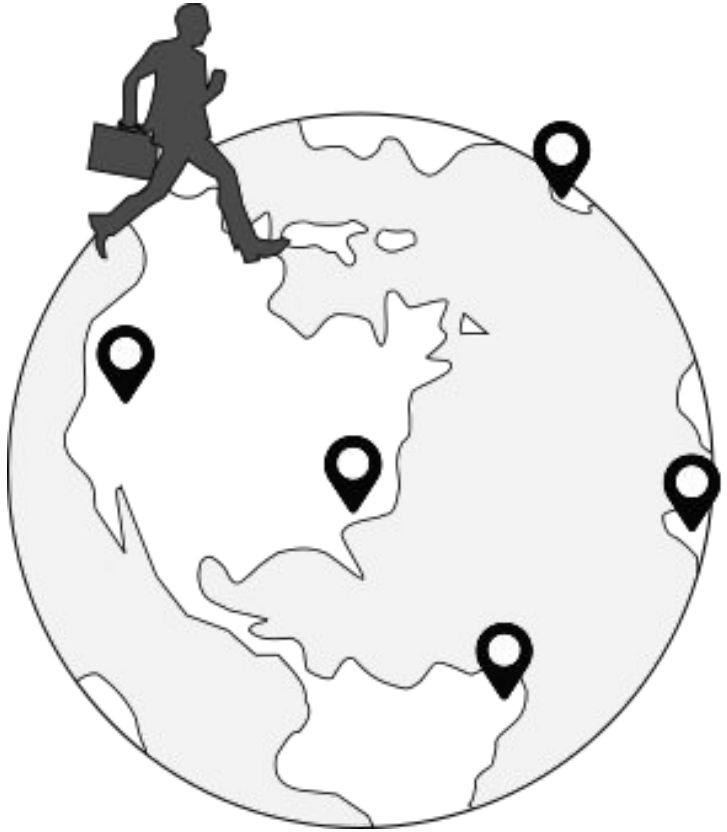
Let's talk about a couple of **examples**:

- **The Traveling Salesperson Problem**
- Puzzle Solving
- Subset Sum
- Boolean Satisfiability
- Exam Scheduling

Traveling Salesperson Problem (TSP): Given a map, find the shortest possible route for our traveler to visit every city, and then return home.

Example application:

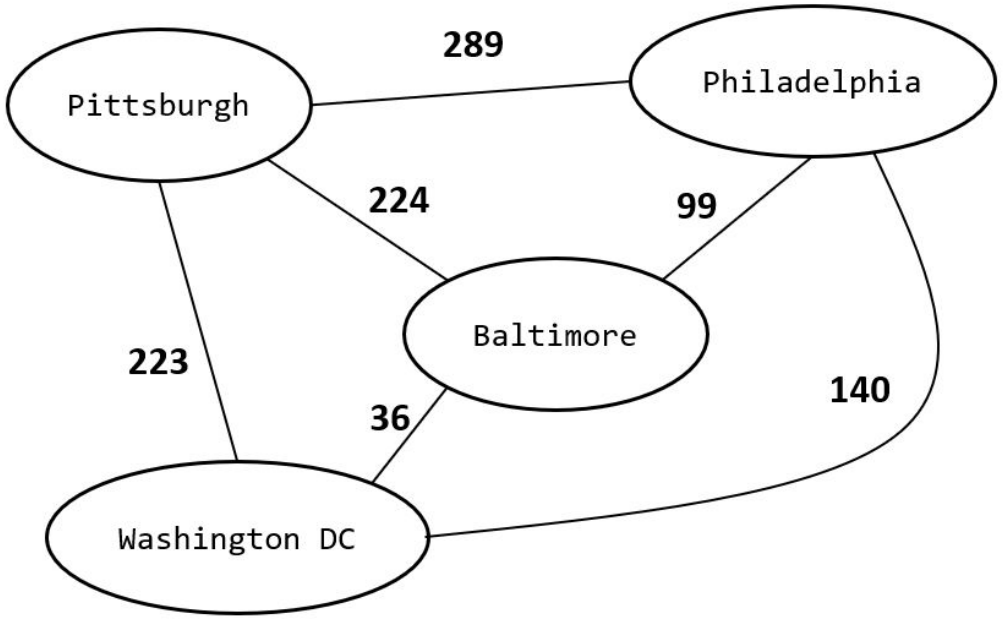
Finding the shortest route for a postal work to deliver everyone's mail and then return to the post office.



Traveling Salesperson Problem (TSP): Given a map, find the shortest possible route for our traveler to visit every city, and then return home.

Example application:

Finding the shortest route for a postal work to deliver everyone's mail and then return to the post office.



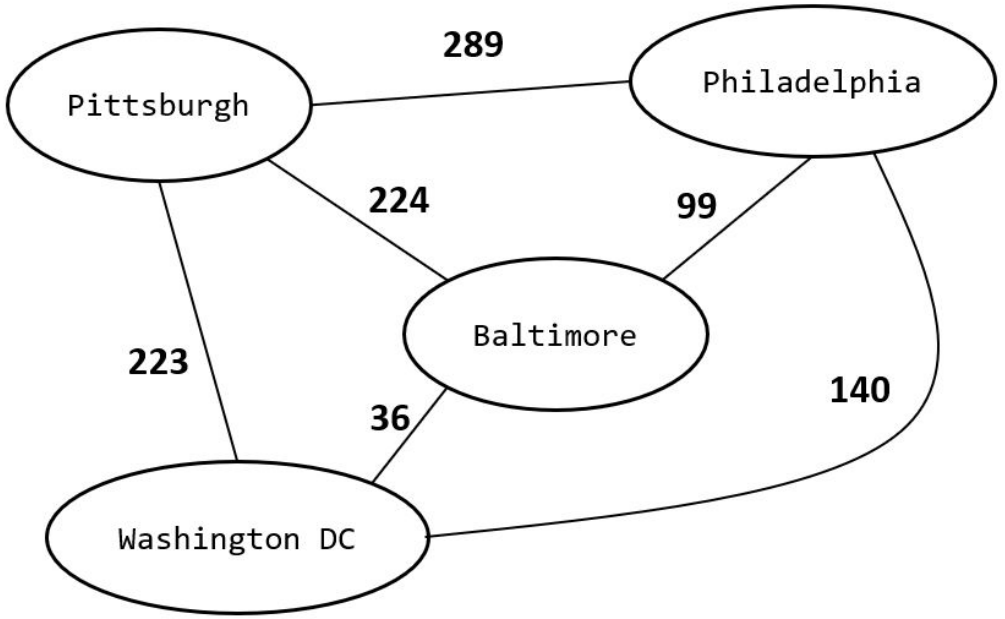
CS problem:

Given an undirected weighted graph, find the shortest path starting from node N that visits all the nodes and ends at N.

Traveling Salesperson Problem (TSP): Given a map, find the shortest possible route for our traveler to visit every city, and then return home.

One possible solution:

Calculate every possible route, from starting node to all the other nodes and pick the shortest route.



For example: Assume starting from Pittsburgh. How many possible routes are there? What is the shortest route?

This is a **brute force algorithm!**

Brute force algorithms are simple: generate every possible solution, then check each possible solution to see if it meets the problem's constraints.

Brute force algorithms are easy to understand, implement, and test. They also apply to a wide range of problems, which makes them useful.

However, brute force algorithms are **inefficient!**

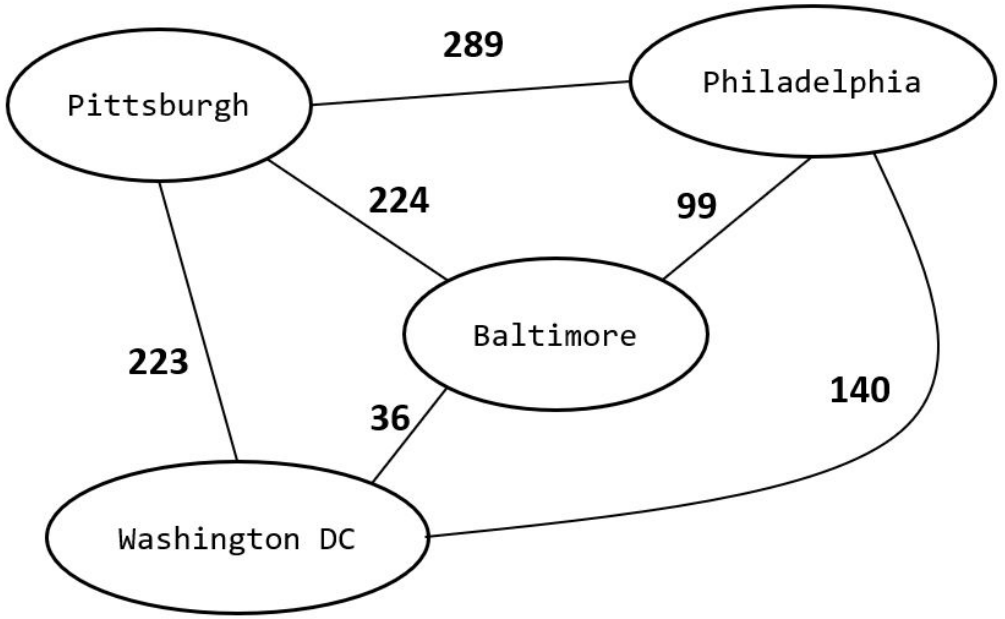
In the worst case, our TSP brute force algorithm is **O(n!)**

The **worst case input** for a TSP is a fully connected graph.

For a graph with n nodes:

The number of possible routes is (n-1) * (n-2) * (n-3) * … * 1 = n!

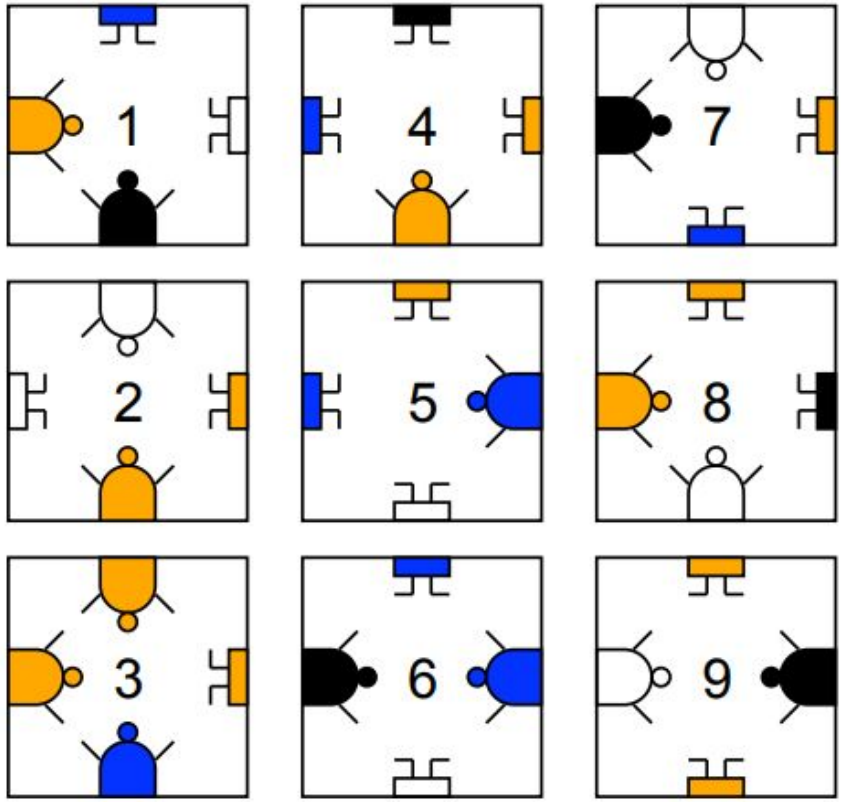
O(n!) is really <u>inefficent!</u>



Puzzle solving: Given a set of pieces is there a solution?

Solve a basic puzzle by putting together square pieces (like the ones shown to the right) so that any two pieces that are touching each other make a figure with a head and feet of the same color.

Assume can't rotate pieces.

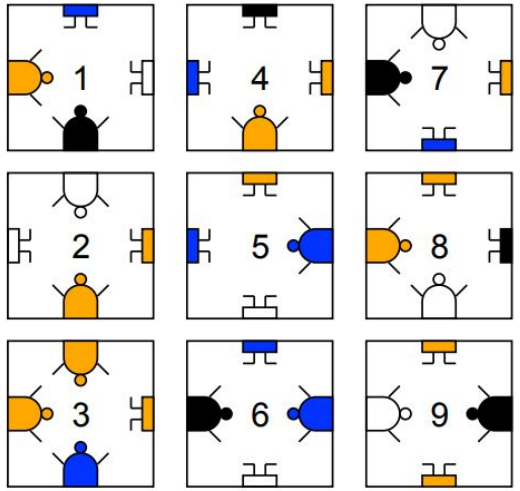


Puzzle solving: Given a set of pieces is there a solution?

Brute force algorithm:

Try all possible pieces for each location.

In the example to the right there are 9 options for the first position, 8 for the second, 7 for the third, etc.... it's **O(n!)** time again.



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

O(n!) is a really bad runtime.

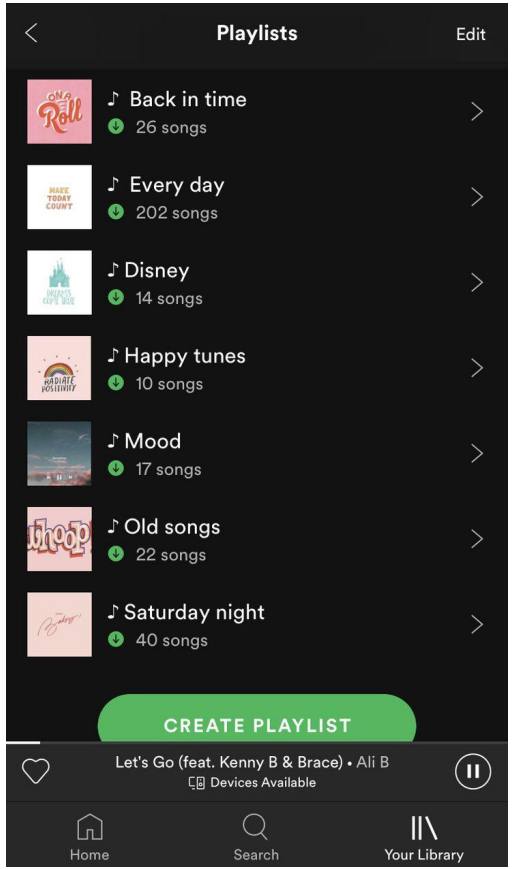
Let's say we want to solve the puzzle and it takes us 1 millisecond to setup a specific ordering of the pieces and check the solution.

Where n is the number of puzzle pieces:

n	time to solve: O(n!)
9	6.048 minutes
16	663.46 years!

Subset Sum: Given a list of numbers and a target number x, determine if some subset of these numbers will sum to x.

Example application:
From a list of popular songs, make a playlist that is exactly 2 hours long.

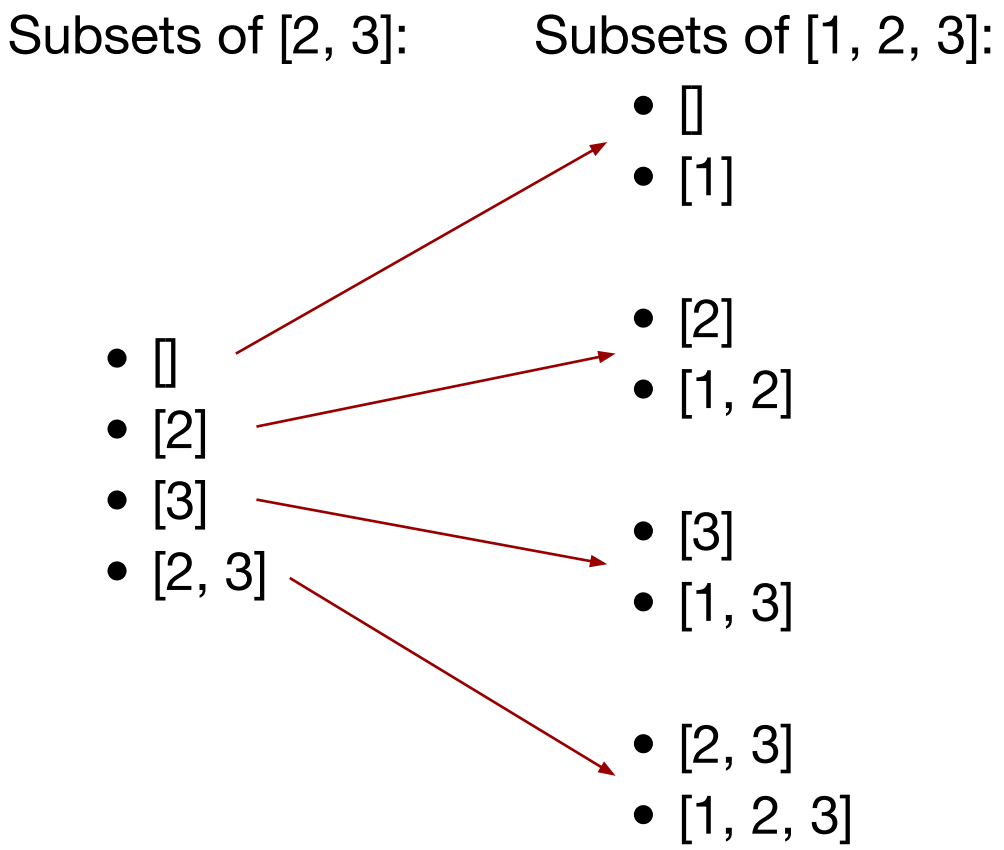


Brute force algorithm:
Generate all possible subsets and check if any of them sum to x.

Subset Sum: Given a list of numbers and a target number x, determine if some subset of these numbers will sum to x.

Use recursion to generate all subsets of [1,2,3]:
If we have all four subsets of the list [2, 3] we can use them to create all 8 subsets of [1, 2, 3].

We double the number of subsets with each new number that is added- this is **$O(2^n)$**.



Boolean Satisfiability: Given a circuit, is there a set of inputs that will make it output 1?

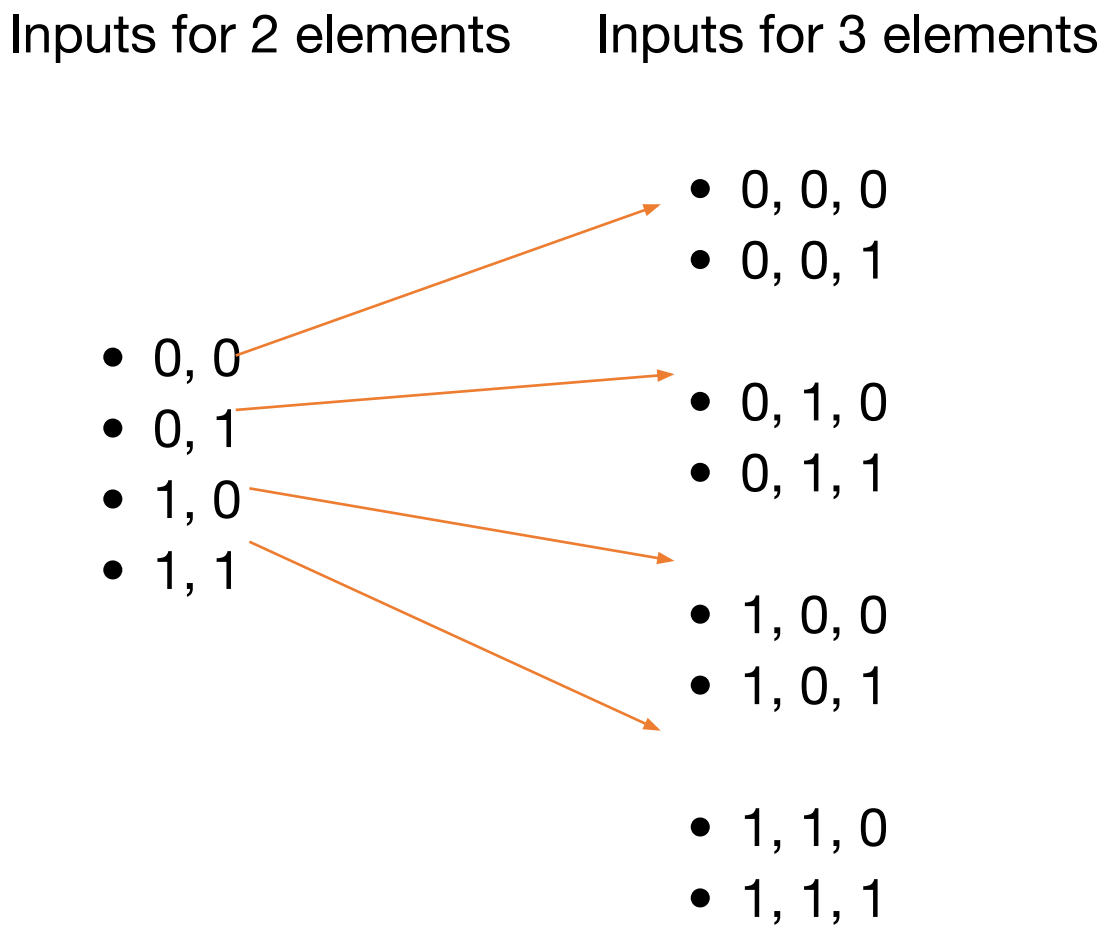
Example application:

Finding the solution to a Sudoku puzzle!

Brute force algorithm:

Build a truth table – check the output of every possible set of inputs.

Size of truth table doubles each time we add a new input so **$O(2^n)$**.



Scheduling Final Exams: Schedule exams such that no students has two final exams at the same time.

CS problem:

Given a list of timeslots for exams and a dictionary with all students names mapped to their classes, generate a schedule where no student has an exam at the same time.

Semester & Mini-2 Final Exams: December 9, 10, 12, 13, 15 & 16(Make-Up Day)

Course	Section	Title	Date	Time (USA EST)	Classroom(s)
Architecture					
48116	A	BUILDING PHYSICS	Sunday, December 15, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48315	1	ENVIR I: CLIM & ENG	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48432	A	ENV II	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48531	A	FABRICATNG CUSTOMZTN	Monday, December 9, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48558	A	RLT COMP	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48568	A	ADV CAD BIM 3D VISLZ	Tuesday, December 10, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48635	1	ENVIRO I M.ARCH	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48655	A	ENV II GRAD	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48714	A	DATA ANL URBN DSNG	Friday, December 13, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48729	A	PROD HLTH QUAL BLDGS	Thursday, December 12, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48734	A	RCTV SP MD ARC	Friday, December 13, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
48743	A	INTRO ECO DES	Friday, December 13, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48749	A	CD SPECIAL TOPICS	Tuesday, December 10, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48785	A	MAAD RES PROJ	Sunday, December 15, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
48798	A	HVAC & PS LOW CARB B	Monday, December 9, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
Art					
60157	A	DRAWING NON-MAJORS	Tuesday, December 10, 2019	05:30 pm - 08:30 pm	CFA TBD
60218	A	REAL-TIME ANIMATION	Monday, December 9, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
60220	A	TECH CHARACTER ANIM	Thursday, December 12, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
60220	B	TECH CHARACTER ANIM	Thursday, December 12, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
60333	A	CHARACTER RIGGING	Sunday, December 15, 2019	08:30 am - 11:30 am	BH 140F

Scheduling Final Exams: Schedule exams such that no students has two final exams at the same time.

Brute force solution:

Generate every possible exam schedule.

If there are n classes that can go into and k timeslots this is **$O(k^n)$**!

Semester & Mini-2 Final Exams: December 9, 10, 12, 13, 15 & 16 (Make-Up Day)

Course	Section	Title	Date	Time (USA EST)	Classroom(s)
Architecture					
48116	A	BUILDING PHYSICS	Sunday, December 15, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48315	1	ENVIR I: CLIM & ENG	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48432	A	ENV II	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48531	A	FABRICATNG CUSTOMZTN	Monday, December 9, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48558	A	RLT COMP	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48568	A	ADV CAD BIM 3D VISLZ	Tuesday, December 10, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48635	1	ENVIRO I M.ARCH	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48655	A	ENV II GRAD	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48714	A	DATA ANL URBN DSNG	Friday, December 13, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48729	A	PROD HLTH QUAL BLDGS	Thursday, December 12, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48734	A	RCTV SP MD ARC	Friday, December 13, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
48743	A	INTRO ECO DES	Friday, December 13, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48749	A	CD SPECIAL TOPICS	Tuesday, December 10, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48785	A	MAAD RES PROJ	Sunday, December 15, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
48798	A	HVAC & PS LOW CARB B	Monday, December 9, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
Art					
60157	A	DRAWING NON-MAJORS	Tuesday, December 10, 2019	05:30 pm - 08:30 pm	CFA TBD
60218	A	REAL-TIME ANIMATION	Monday, December 9, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
60220	A	TECH CHARACTER ANIM	Thursday, December 12, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
60220	B	TECH CHARACTER ANIM	Thursday, December 12, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
60333	A	CHARACTER RIGGING	Sunday, December 15, 2019	08:30 am - 11:30 am	BH 140F

$O(2^n)$ and $O(k^n)$ are Still Really Slow

$O(2^n)$ is a bit better than O(n!), but not *that* much better. Let's say we want to solve the subset sum problem and it again takes us 1 millisecond to generate a specific subset and see if it is equal to the target.

If n = 10, we find the solution in **1.024 seconds**. Much better!

But if n = 20, we find the solution in **17.48 minutes**...

And if n = 30, it will take us **12.43 days**. By the time n = 40, it takes **35 years**.

$O(2^n)$ is not as bad as O(n!), but it's still really bad.

$O(2^n)$ and $O(k^n)$ are better than O(n!) but are still **really slow**.

Let's say we want to solve the subset sum problem and it takes us 1 millisecond to generate a specific subset and see if it is equal to the target.

n	**time to solve: $O(2^n)$**
10	1.024 seconds
20	17.48 minutes
30	12.43 days
40	35 years

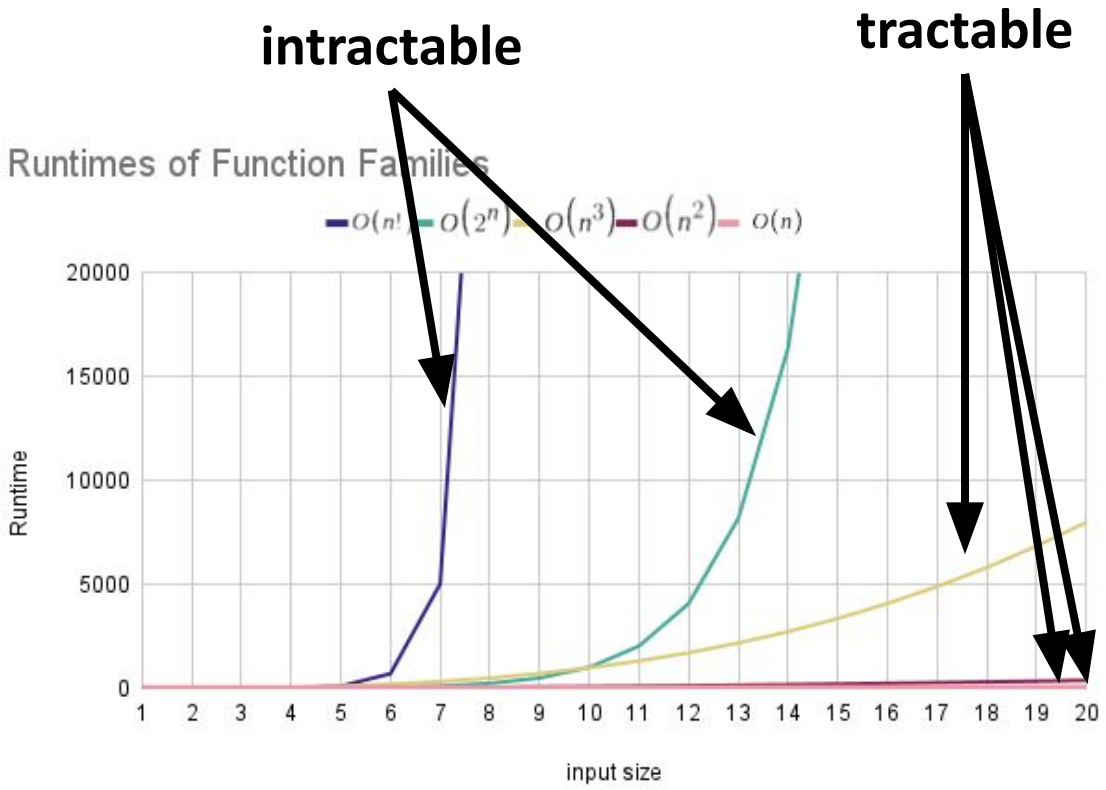
A problem is said to be **tractable** if its solution has a **runtime that is reasonable**.

A runtime is reasonable if it can be **expressed as a polynomial**:

$$c_k n^k + c_{k-1} n^{k-1} + \ldots + c_1 n + c_0$$

where n is a variable and c_i & k are constants.

O(1), O(log n), O(n), $O(n^2)$, and $O(n^k)$ are all tractable. $O(2^n)$, $O(k^n)$, and O(n!) are not- they're **intractable**.



Caveat: logarithms are tractable even though they aren't polynomial, because they're faster than O(n)!

Activity: Identify the Solution Runtime

If you consider how a brute-force solution generates solutions, and how that algorithm would be affected by increasing the input size, you can often determine whether the solution will be tractable or intractable without digging deeply into the exact runtime.

You do:

- solve a **n**x**n** Sudoku puzzle by trying every possible combination of numbers. Is that tractable or intractable?
- check every pair of elements in a **n**-element list to see if there are any duplicates. Is that tractable or intractable?

Complexity Classes

Can we **find tractable solutions** to these hard problems?

We saw brute force algorithms were intractable.

For the rest of this lecture: **can we do better?**

Discuss: Can you think of anyway to make solving TSP faster?

Designing new solutions is hard!

We **classify the runtime of a problem** using computational complexity theory.

Classifying the complexity of problems helps computer scientists **compare problems** based on if they are practically solvable or unsolvable.

There are **two different ways** to describe the complexity of a problem:

- The amount of time it takes to find the solution
- The amount of time it takes to verify the solution

Verifying a solution: Given a solution, **how long does it take to check** that is a valid solution?

Exam schedule example: How long does it take to check if a particular configuration of n classes has no student taking a final for different classes at the same time?

Does this schedule have no conflicts?

Semester & Mini-2 Final Exams: December 9, 10, 12, 13, 15 & 16(Make-Up Day)

Course	Section	Title	Date	Time (USA EST)	Classroom(s)
Architecture					
48116	A	BUILDING PHYSICS	Sunday, December 15, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48315	1	ENVIR I: CLIM & ENG	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48432	A	ENV II	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48531	A	FABRICATNG CUSTOMZTN	Monday, December 9, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48558	A	RLT COMP	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48568	A	ADV CAD BIM 3D VISLZ	Tuesday, December 10, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48635	1	ENVIRO I M.ARCH	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48655	A	ENV II GRAD	Thursday, December 12, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
48714	A	DATA ANL URBN DSNG	Friday, December 13, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48729	A	PROD HLTH QUAL BLDGS	Thursday, December 12, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48734	A	RCTV SP MD ARC	Friday, December 13, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
48743	A	INTRO ECO DES	Friday, December 13, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48749	A	CD SPECIAL TOPICS	Tuesday, December 10, 2019	01:00 pm - 04:00 pm	To Be Announced (TBA)
48785	A	MAAD RES PROJ	Sunday, December 15, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
48798	A	HVAC & PS LOW CARB B	Monday, December 9, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
Art					
60157	A	DRAWING NON-MAJORS	Tuesday, December 10, 2019	05:30 pm - 08:30 pm	CFA TBD
60218	A	REAL-TIME ANIMATION	Monday, December 9, 2019	08:30 am - 11:30 am	To Be Announced (TBA)
60220	A	TECH CHARACTER ANIM	Thursday, December 12, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
60220	B	TECH CHARACTER ANIM	Thursday, December 12, 2019	05:30 pm - 08:30 pm	To Be Announced (TBA)
60333	A	CHARACTER RIGGING	Sunday, December 15, 2019	08:30 am - 11:30 am	BH 140F

Verifying a solution: Given a solution, **how long does it take to check** that is a valid solution?

Verifying exam schedule example: (n is the total number of classes)

For each student, check their exam time slots for conflicts.

Number of classes each student has is constant: we will say no more than 5.
The number of students is also a constant: we will say 6*n

Therefore, overall we have to do students*conflict-checks = (6*n)*10 work. That's 60n, which is **O(n)**. **Verifying the solution is tractable!**

A **complexity class** is a collection of problems with similar runtimes (complexity).

We say that every problem is in a certain complexity class if is **bounded by a certain runtime**.

For example, we could design a complexity class called 'Fast' that only includes algorithms which run in O(n) time or faster. This would also include O(log n) and O(1).

Let's talk about **two important complexity classes**: P and NP

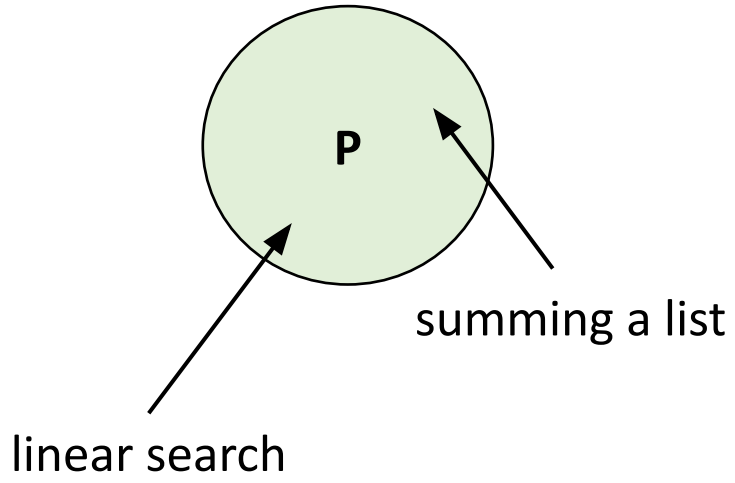
We define the **complexity class P** to be the set of problems that we know can be **solved in polynomial time**.

Polynomial runtime can be expressed as:

$$c_k n^k + c_{k-1} n^{k-1} + \ldots + c_1 n + c_0$$

where n is a variable and c_i & k are constants.

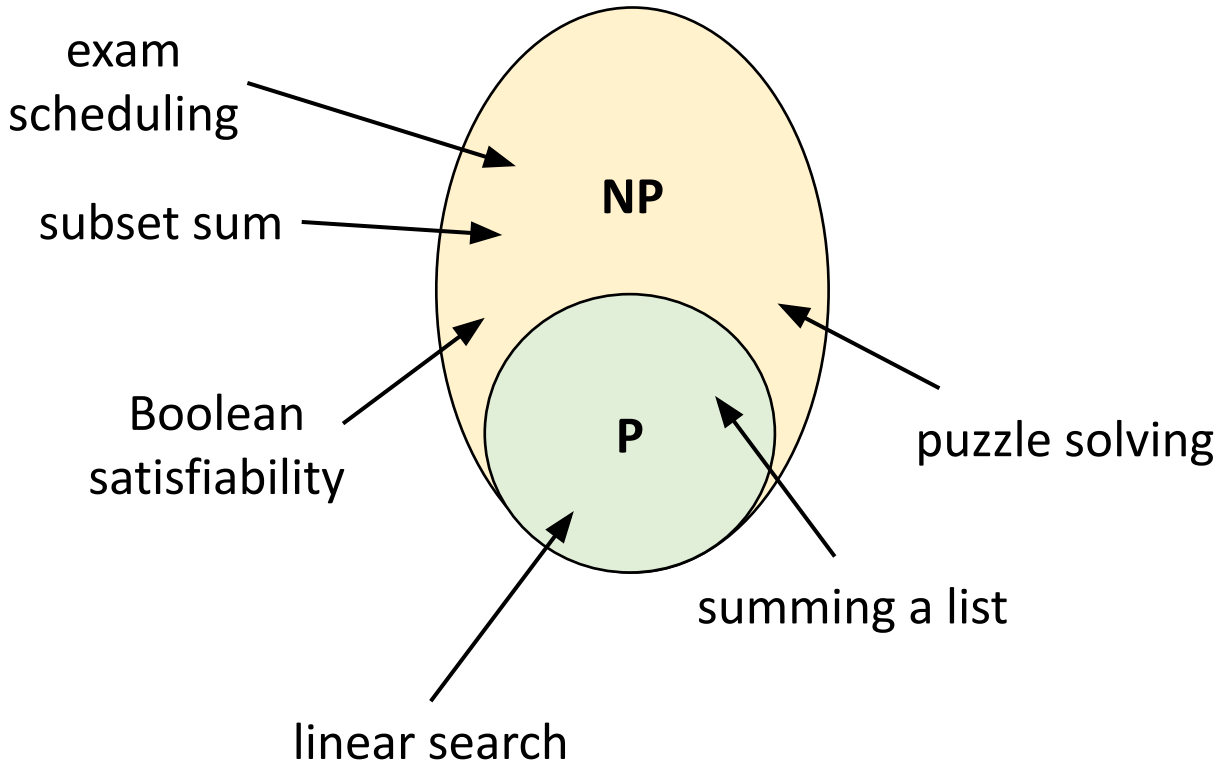
Intractable problems do not fall into this category. But plenty of other algorithms do- linear search, summing a list, etc.



We define the **complexity class NP** to be the set of problems that can be **verified in polynomial time**.

All problems in P are also in NP: If you can solve in polynomial time then you can also very in polynomial time.

Examples include exam scheduling which we could verity in polynomial time, but could not solve in polynomial time.

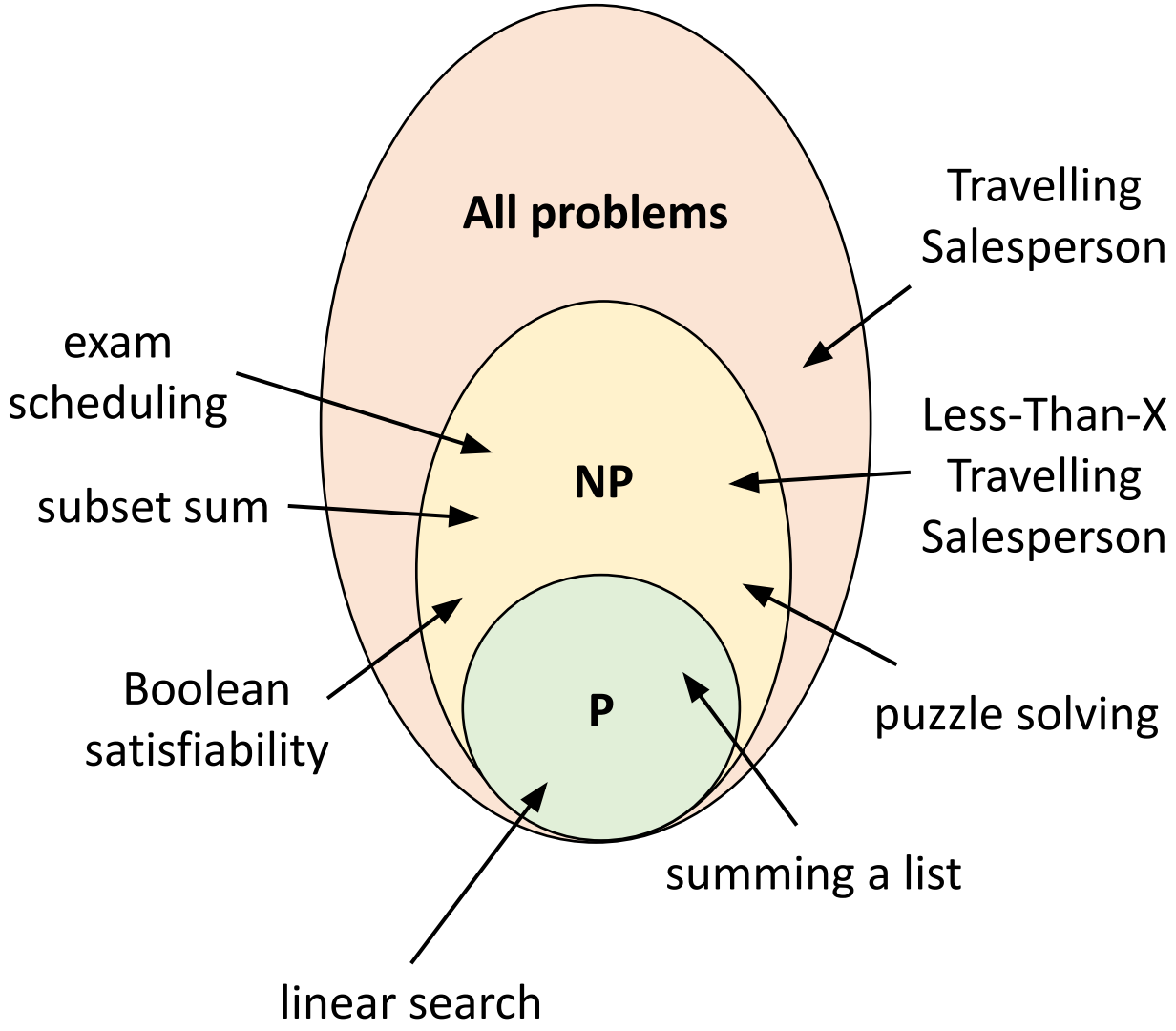


Some problems are so hard they are **not in NP or P**.

Example:

If **given a solution to TSP**, we can't verify it is the *best* path without still checking all the other paths.

We can make TSP NP by changing the problem: find the best path that is no more than X distance. This is easy to verify.

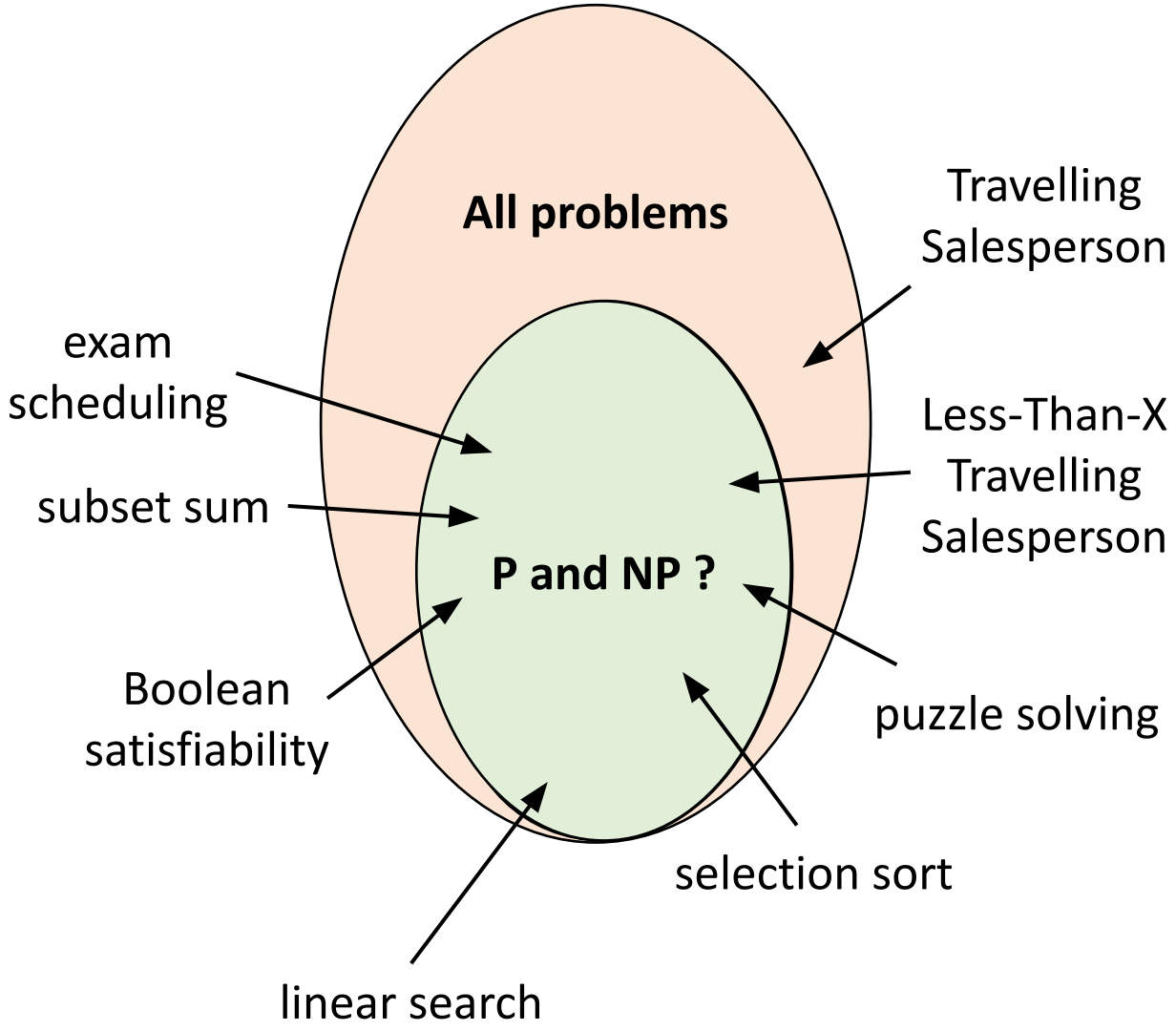


Important CS Problem: Does P = NP?

A core question in computer science: Are the set of all problems in P the same as all the problems in NP?

Can every problem that we can verify in polynomial time also be solved in polynomial time?

We'd be able to solve a lot of hard problems really quickly, without having to think hard about clever new approaches!



Does P = NP? is a **major unsolved problem** in theoretical CS.

The first person who proves whether or not P = NP will win a million dollars, but no one has proved it yet…

How would we prove this? Why is this so hard to prove?

There are two possible approaches:

1. Prove that P!=NP
2. Prove that P=NP

We can try to prove that P != NP.

Let's assume that P = NP. How would we prove this?

You'd need to **definitively prove that a problem in NP exists that cannot be solved in polynomial time.** But how can we show that it's impossible to come up with a clever new algorithm? This is tricky!

We can try to prove that P = NP.

Let's assume that P != NP. How would we prove this?

You need to show that **every problem in NP can be solved in polynomial time**. That's a lot of problems!

To make this easier, computer scientists **try to find problems in NP that are related** to each other.

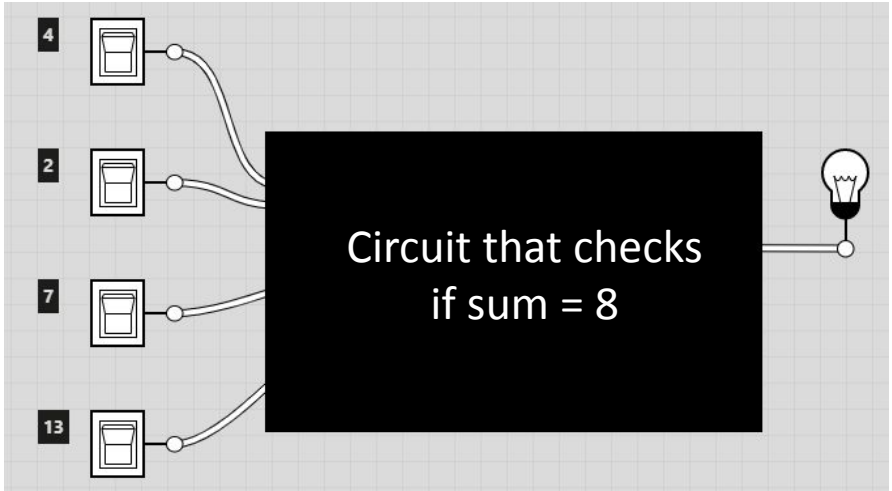
For example, we can **transform subset sum into boolean satisfiability**.

We can **make a circuit that uses each value in the list as an input** (0 if it isn't included, 1 if it is) and make the circuit output 1 if the included values sum to the target.

This mapping can be done in **polynomial time**. This means that if we can find a tractable solution to Boolean satisfiability, we can also use it to make a tractable solution to subset sum.

Find a subset of [4, 2, 7, 13] that sums to 8

Set the inputs so that the circuit outputs 1



Computer scientists have identified a set of problems that have this problem-transformation capacity **for all NP problems**.

If we can find a tractable solution to one of them, we can make all problems in NP tractable. That will mean that P = NP!

In fact, if you use the limited version of the TSP, all the problems we discussed today are in this set of problems.

If you can find a tractable solution to any of these problems, you'll prove P = NP and will become rich and famous.

Possible Outcomes

What happens if we prove P = NP?

We'll be able to solve a lot of hard problems very quickly. NP problems show up everywhere, so nearly everything in the world will get radically faster!

On the other hand, this might also wreck how modern security and encryption is implemented (as it will get easier to break cryptography).

What happens if we prove P != NP?

Not much; we'll still be in our current situation. But a lot of computer scientists can turn their focus to other problems.

Most people think P != NP, but we don't know how to prove it.

Heuristics

We can solve intractable problems in practice if we can change our standards: rather than finding the best solution, **find a good-enough solution**.

For example:

- In exam scheduling, maybe it's okay if there's a small number of conflicts that affect < 1% of the student body
- In subset sum, maybe it's okay if we find a subset that is *almost* equal to the target, instead of exactly equal

When we're willing to compromise on optimality or accuracy, or put other restrictions on the data, we can use **heuristics** to speed up the process a great deal.

A **heuristic** is a search technique used by an algorithm to **find a good-enough approximate solution** to a problem.

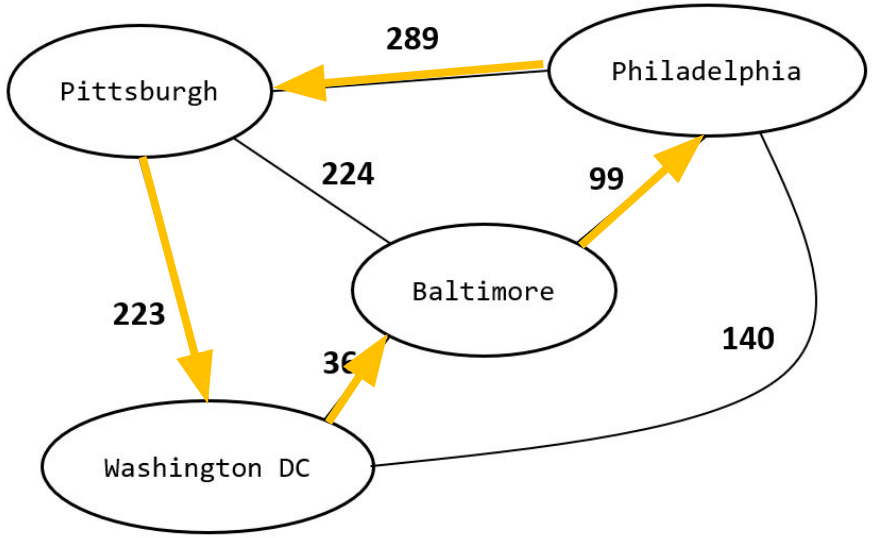
Heuristics may not find the best answer to an NP problem, but they **often achieve good results**.

A heuristic can **generate scores to rank potential next steps** that the algorithm can take at each decision point. By choosing the highest-scored next step, the algorithm is more likely to find a working solution quickly.

We can use a heuristic to **speed up TSP**.

Heuristic: Rank the next-possible paths based on their length. **Choose the next city with the shortest length.**

With this approach, we can **generate a pretty good path in polynomial time** though it may not be the best possible one.



We can also design a **heuristic for subset sum**.

Heuristic: Assuming positive numbers, order from largest to smallest.

Add next number to subset until we are within 2 of the target. If we add a number that is too large, backtrack and try a different number.

How many subsets do we need to try to determine if there's a subset of [13, 14, 7, 10, 7, 16, 2, 8, 3, 5] that sums to ~25?

Sort the list: [16, 14, 13, 10, 8, 7, 7, 5, 3, 2]

[16] – too small

[16, 14] – too big, backtrack!

[16, 13] – still too big...

[16, 10] – this is 26, it works!

We missed the optimal solution – [16, 7, 2] would have been perfect. But we found [16, 10] much faster.

Sidebar: Additional Watching

Want to learn more about these topics? Check out the following videos recommended by prior students!

P vs. NP and the Computational Complexity Zoo:
https://www.youtube.com/watch?v=YX40hbAHx3s

P vs. NP - The Biggest Unsolved Problem in Computer Science:
https://www.youtube.com/watch?v=EHp4FPyajKQ

Learning Goals

- Identify **brute force approaches** to common problems that run in **O(n!)** or **$O(2^n)$**, including solutions to **Travelling Salesperson, puzzle-solving, subset sum, Boolean satisfiability**, and **exam scheduling**

- Define the complexity classes **P** and **NP** and explain why these classes are important

- Identify whether an algorithm is **tractable** or **intractable**, and whether it is in **P**, **NP,** or **neither** complexity class

- Use **heuristics** to find good-enough solutions to NP problems in polynomial time