CSE 332: Data Structures and Parallelism

Fall 2022

Richard Anderson

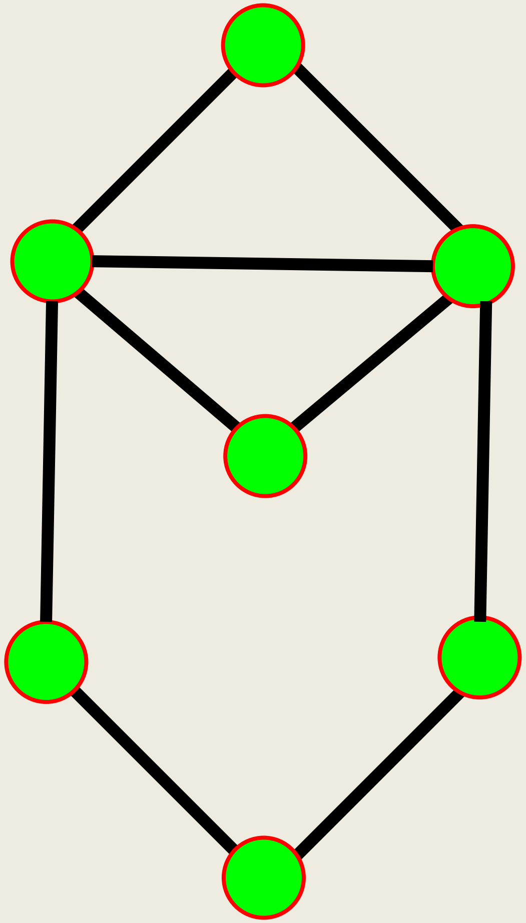
Lecture 28: NP-Completeness

Announcements

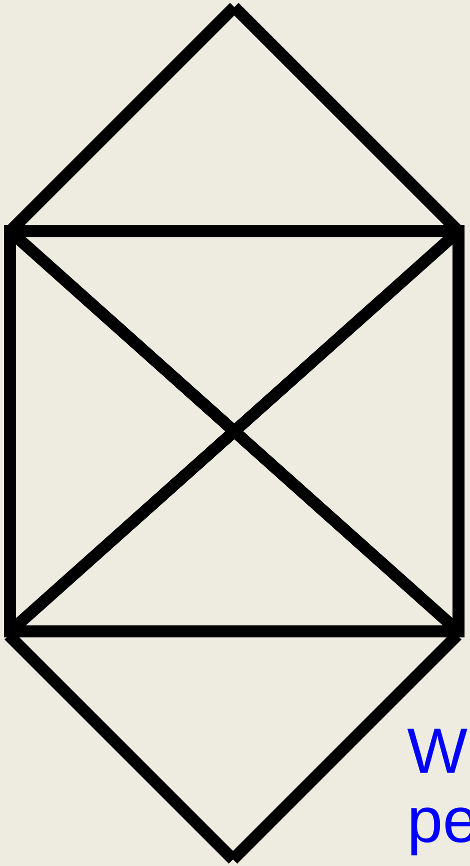
- Final
  - Thursday, December 15, 8:30-10:20 AM, CSE2 G20
  - No notes, calculator, internet, etc.
  - Comprehensive
    - All topics covered on the lecture slides
    - Estimate: 1/3rd pre-midterm, 2/3rd post-midterm
- Resources
  - Old exams
  - Quiz section this week
  - Review session, Tuesday, December 13, 4:30-6 PM, location TBD

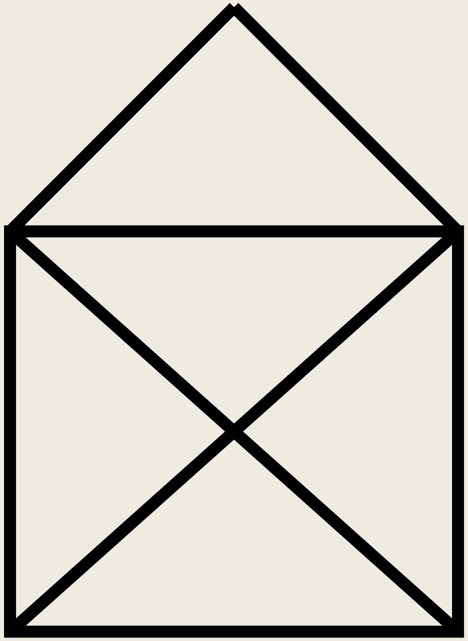
Your First Task

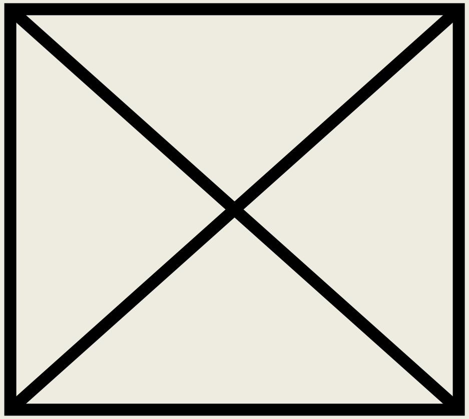
Given a connected, undirected graph, can you find a cycle that includes every edge exactly once



Try it with paper and pencil







Which of these can you draw without lifting your pencil, drawing each line only once?
Can you start and end at the same point?

Euler Circuits and Tours

$e^{i\pi} = -1$

- Euler tour: a path through a graph that *visits each edge exactly once*
- Euler circuit: an Euler tour that *starts and ends at the same vertex*
- Named after Leonhard Euler (1707-1783), who cracked this problem and founded graph theory in 1736
- Some results for undirected graphs:
  - An Euler circuit exists *iff* the graph is connected and each vertex has even degree (= # of edges on the vertex)
  - An Euler tour exists *iff* the graph is connected and at most two vertices have odd degree

Finding Euler Circuits

Given a connected, undirected graph G = (V,E), find an Euler circuit in G.

Euler Circuit Existence Algorithm:

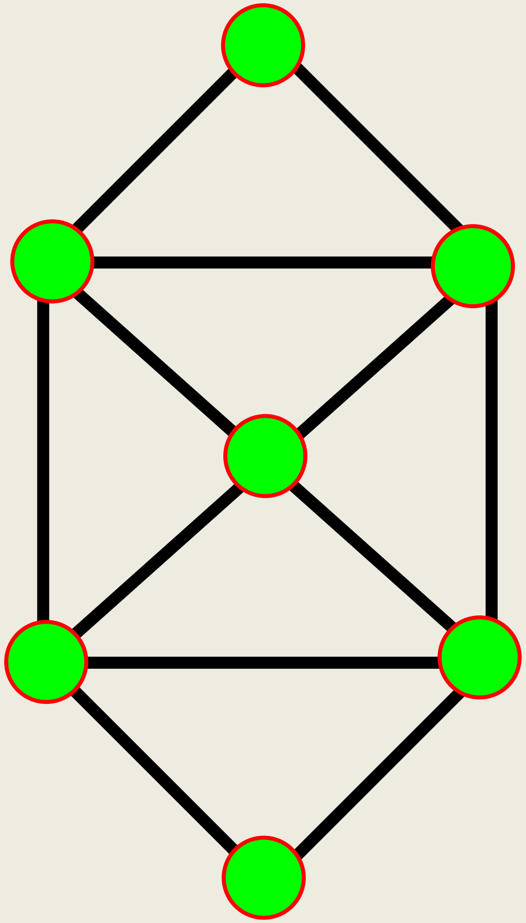
Check to see that all vertices have even degree

Running time =

Euler Circuit Algorithm:

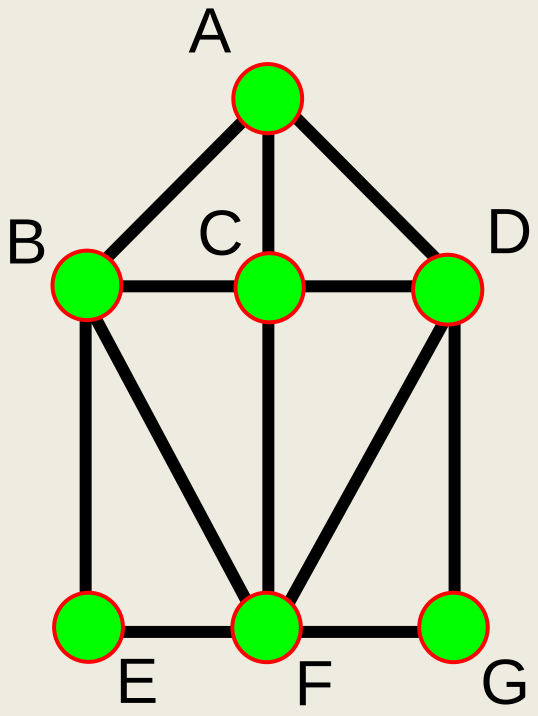
1. Do an edge walk from a start vertex until you are back at the start vertex. Mark each edge you visit, and do not revisit marked edges. You never get stuck because of the even degree property.
2. The walk is removed leaving several components each with the even degree property. Recursively find Euler circuits for these.
3. Splice all these circuits into an Euler circuit

Running time =



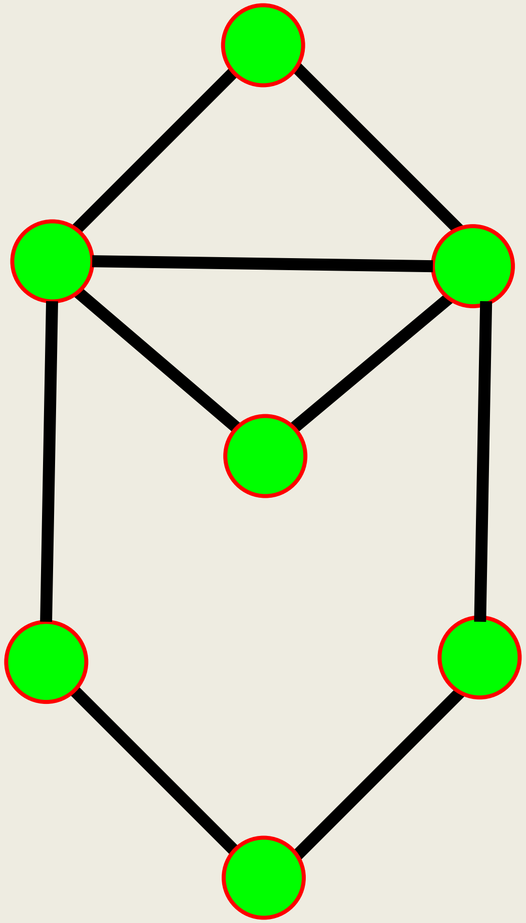
Euler Tour

For an Euler Tour, exactly two vertices are odd, the rest even. Using a similar algorithm, you can find a path between the two odd vertices.



Your Second Task

Given a connected, undirected graph, can you find a simple cycle that includes every vertex exactly once

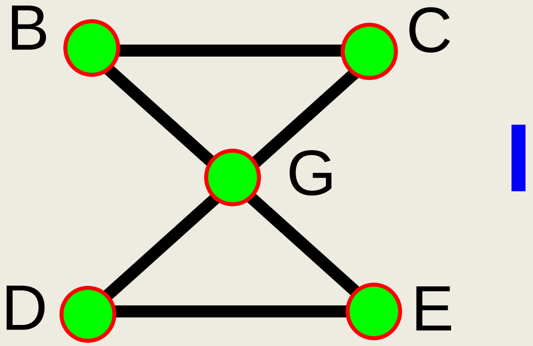


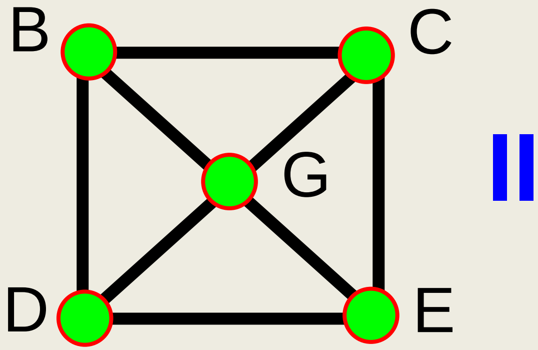
Hamiltonian Circuits

$i^2 + j^2 + k^2 = ijk = -1$

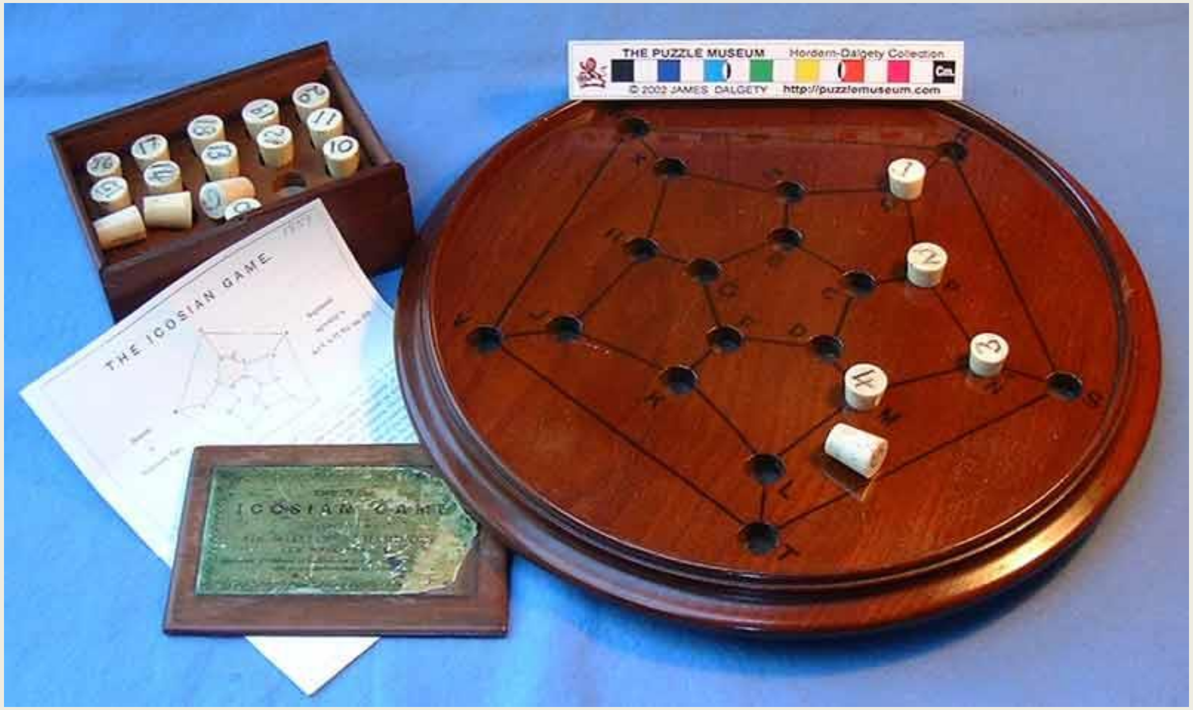
W. R. Hamilton (1805-1865)

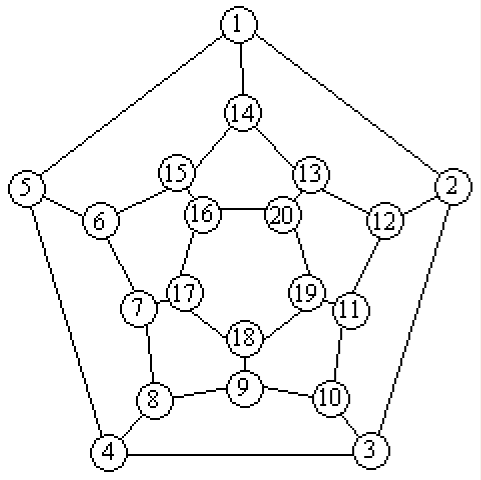
- Euler circuit: A cycle that goes through each *edge* exactly once
- Hamiltonian circuit: A cycle that goes through each *vertex* exactly once (except the first=last one)
- Does graph **I** have:
  - An Euler circuit?
  - A Hamiltonian circuit?
- Does graph **II** have:
  - An Euler circuit?
  - A Hamiltonian circuit?
- Does the Hamiltonian circuit problem seem easier or harder?



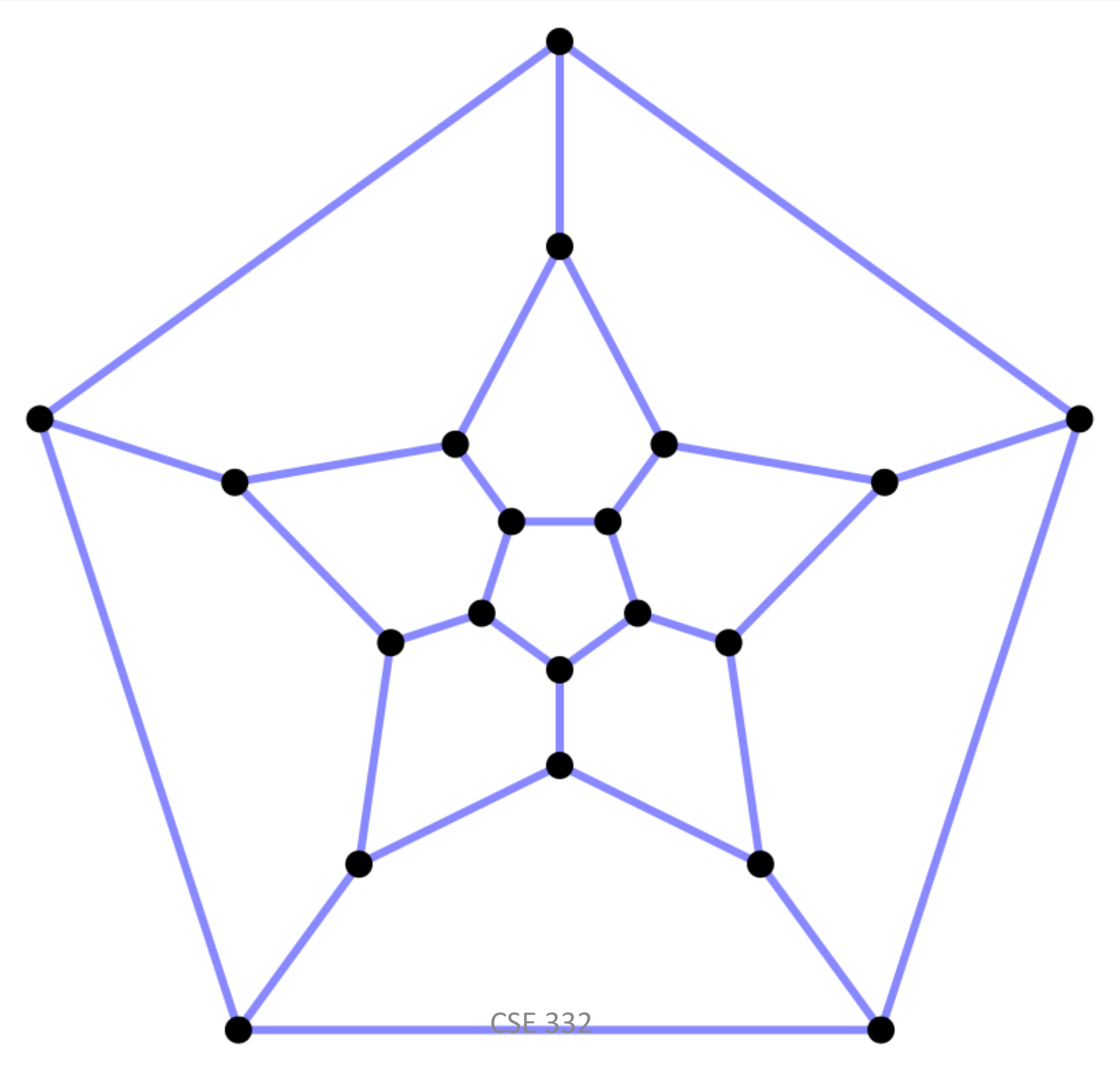


Hamilton’s Icosian Game





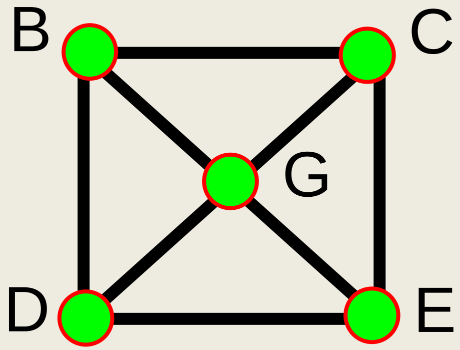
Icosian Graph



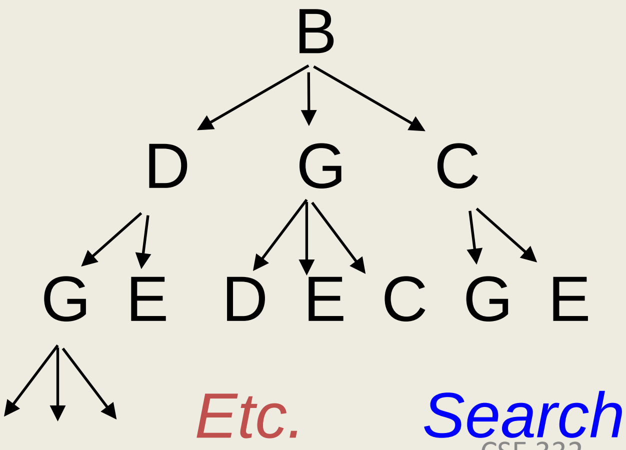
Finding Hamiltonian Circuits

- Problem: Find a Hamiltonian circuit in a connected, undirected graph G.
- One solution: Search through *all paths* to find one that visits each vertex exactly once
  - Can use your favorite graph search algorithm (DFS!) to find various paths
- This is an *exhaustive search* ("brute force") algorithm.
- Worst case → need to search all paths
  - How many paths??

Analysis of our Exhaustive Search Algorithm



- Worst case → need to search all paths
  - How many paths?
- Can depict these paths as a *search tree*



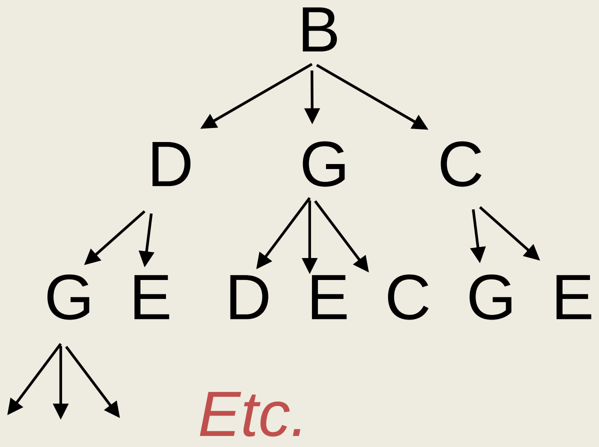
Search tree of paths from B

Analysis of our Exhaustive Search Algorithm

- Let the average branching factor of each node in this tree be b
- |V| vertices, each with ≈ b branches
- Total number of paths ≈ b·b·b … ·b

=

- Worst case → Exponential time!



Search tree of paths from B

Running time

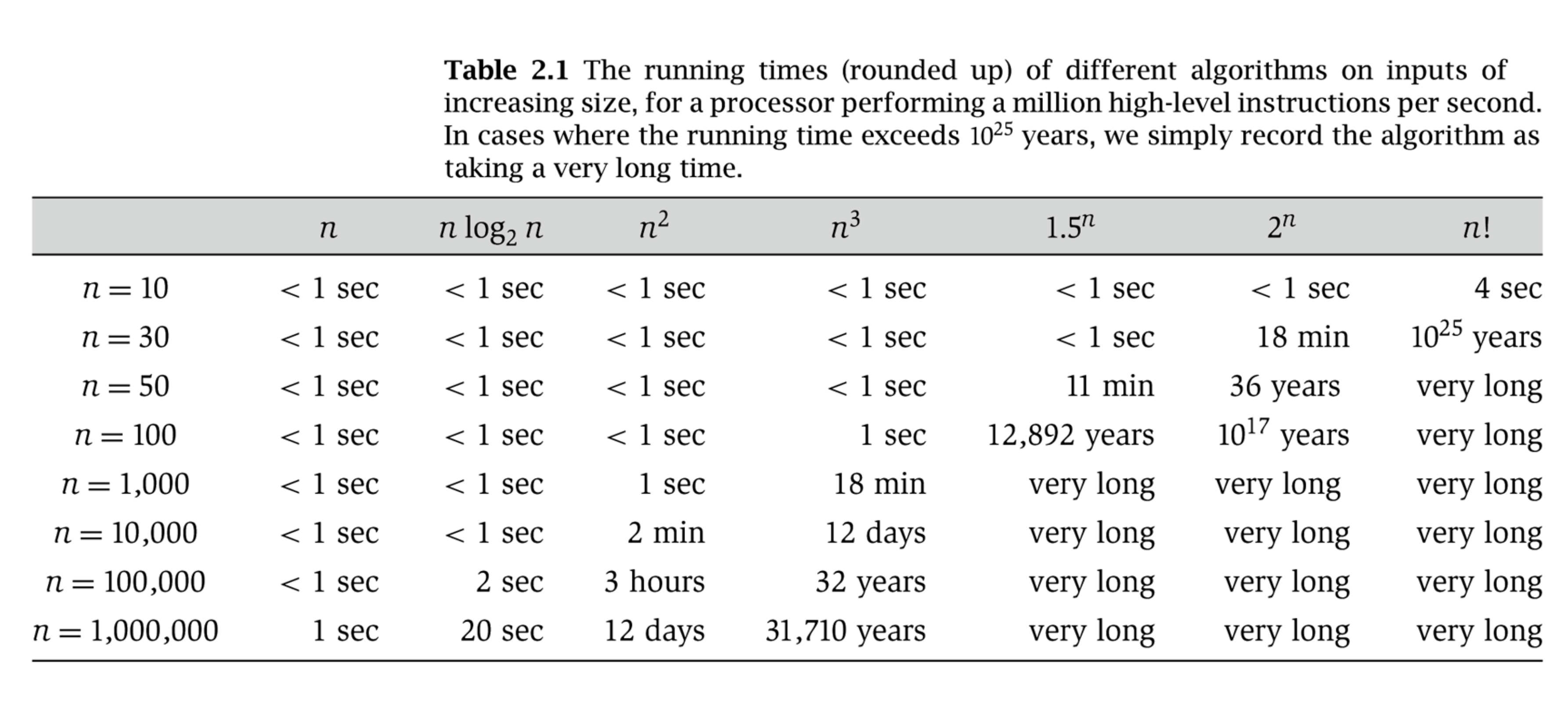


Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	4 sec
$n = 30$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	18 min	10^{25} years
$n = 50$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	11 min	36 years	very long
$n = 100$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	$<$ 1 sec	$<$ 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	$<$ 1 sec	$<$ 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	$<$ 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Polynomial vs Exponential Time

- Most of our algorithms so far have been O(log N), O(N), O(N log N) or $O(N^2)$ running time for inputs of size N
  - These are all *polynomial time* algorithms
  - Their running time is $O(N^k)$ for some k > 0

- Exponential time b^N is asymptotically worse than *any* polynomial function N^k for *any* k

The Complexity Class P

- The set P is defined as the set of all problems that can be solved in *polynomial worst case time*
  - Also known as the *polynomial time* complexity class
  - All problems that have some algorithm whose running time is $O(N^k)$ for some *k*

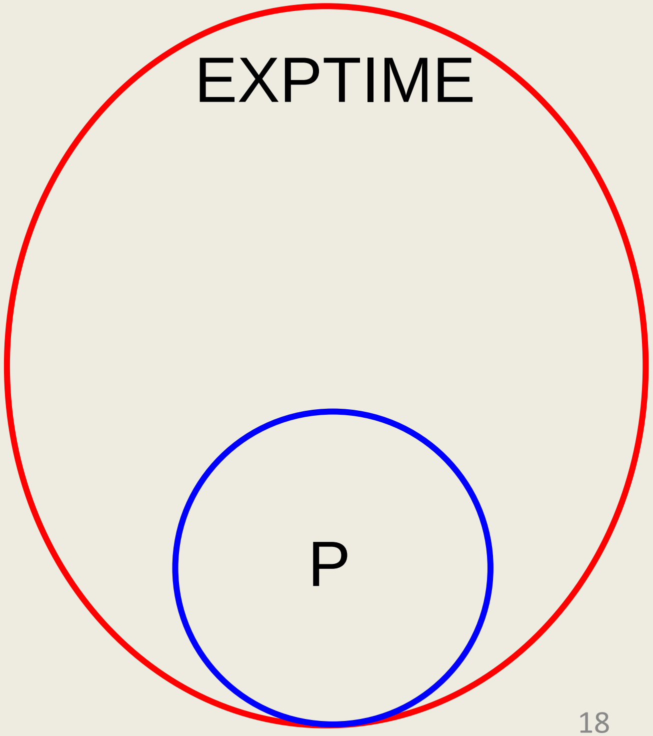
- Examples of problems in P: sorting, shortest path, Euler circuit, *etc*.

Problem Spaces

If a problem is not polynomial-time solvable (not in P), then it is an exponential-time problem.

Shorthand:

- P solutions are fast
- EXPTIME are slow
  - Sometimes viewed as "intractable"



When is a problem easy?

- We’ve seen some “easy” graph problems:
  - Graph search
  - Shortest-path
  - Minimum Spanning Tree

- Not easy for us to come up with, but easy for the computer, once we know algorithm.

When is a problem hard?

- Almost everything we’ve seen in class has had an efficient algorithm

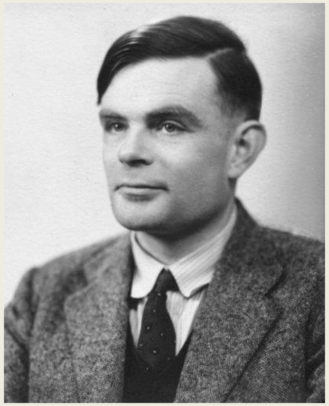
- But of course, computers can’t solve *every* problem quickly.

- In fact, there are perfectly reasonable sounding problems that no computer could ever solve in any reasonable amount of time (as far as we know!).
  - The Hamiltonian Circuit problem is one of these (as far as we know).

When is a problem hopeless?

- Some problems are “undecidable” – no algorithm can be given for solving them.
  - The Halting Problem: is it possible to specify any algorithm, which, given an arbitrary program and input to that program, will always correctly determine whether or not that program will enter an infinite loop?
  - No! [Turing, 1936]

- We’ll focus on problems that have a glimmer of hope…



Alan Turing
(1912-1954)

A Glimmer of Hope

Suppose you have a problem that is at least decideable.

If the output can be checked for correctness in polynomial-time, then ***maybe*** a polynomial-time solution exists!

The Complexity Class NP

- *Definition*: NP is the set of all problems for which a given *candidate solution* can be *tested* in polynomial time

- Are the following in NP:
  - Hamiltonian circuit problem?
  - Euler circuit problem?
  - All problems solved by polynomial time algorithms?

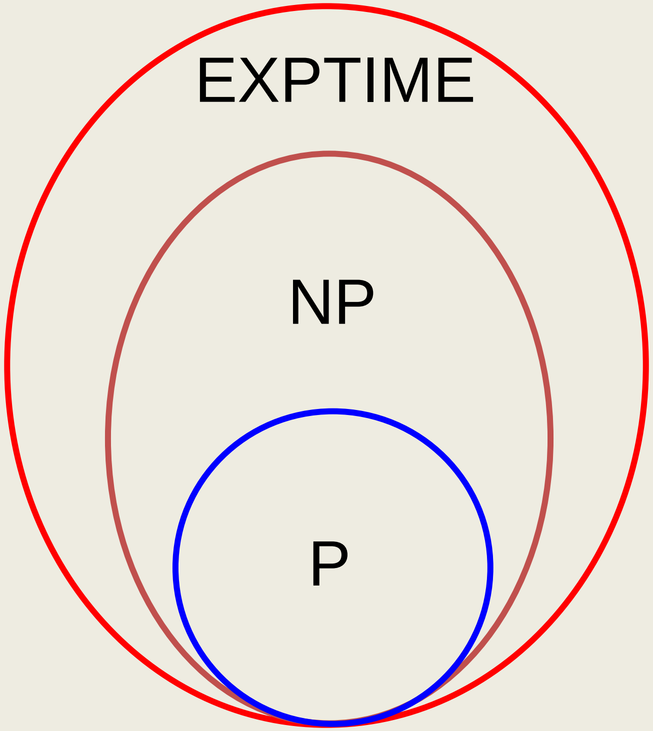
Why NP?

- NP stands for *Nondeterministic Polynomial time*
  - Why "nondeterministic"? Corresponds to algorithms that can guess a solution (if it exists) → the solution is then verified to be correct in polynomial time
  - Nondeterministic algorithms require a nondeterministic computer
    - Multiple choices for the next instruction
    - Accepts an input if some valid computation gets to an accepting state

Problem Spaces (revisited)

We can now augment our problem space to include NP as a superset of P.

Whenever someone finds a polynomial time solution to a problem currently believed to be in NP - P, it moves to P.



Your Chance to Win a Turing Award (and $$$)

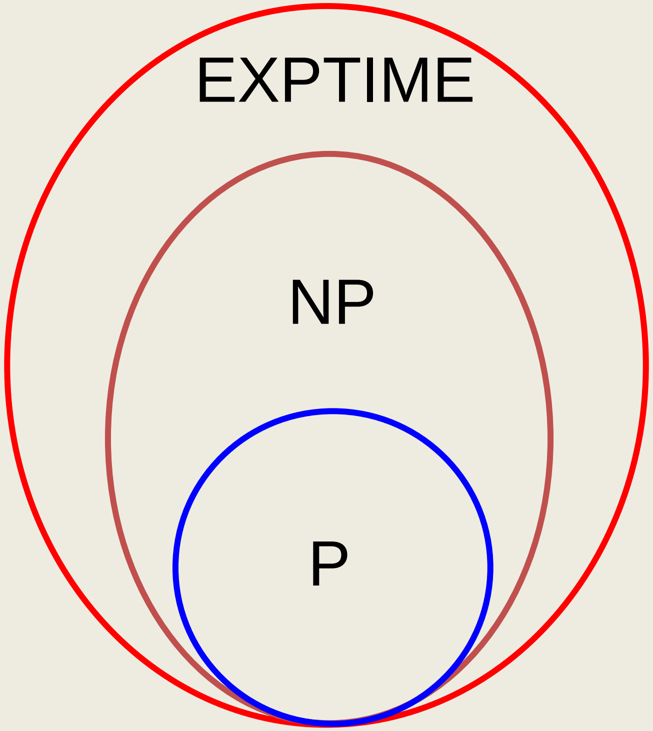
- It is generally believed that P $\neq$ NP, *i.e.*

  there are problems in NP that are not in P

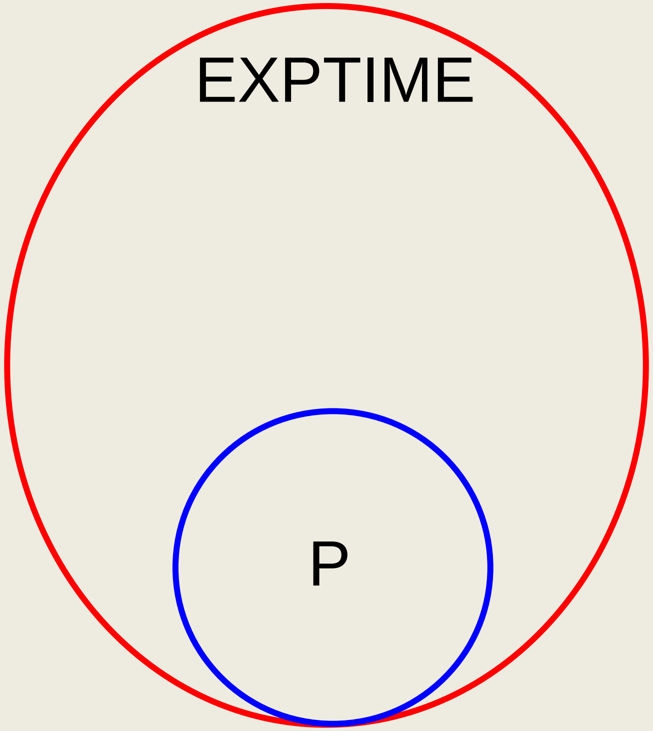
  - But no one has been able to show even one such problem!
  - This is the fundamental open problem in theoretical computer science.

P = NP?

Perhaps instead P = NP, but that would seem to be even harder to prove…



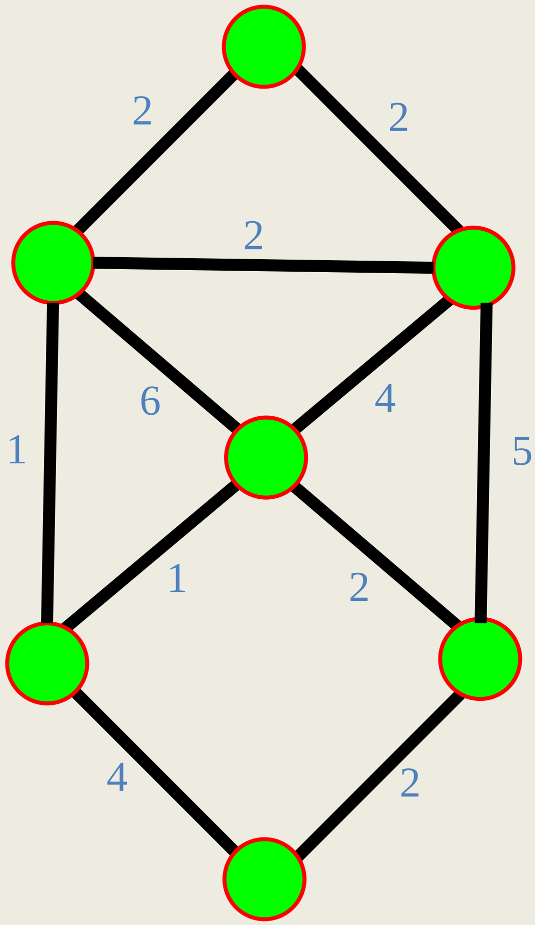
P ≠ NP



P = NP

Your Third Task

Given a connected, undirected graph with edge costs, find a cycle that includes all vertices with minimum total edge cost



The Traveling Salesman Problem (TSP)

- This amounts to solving…

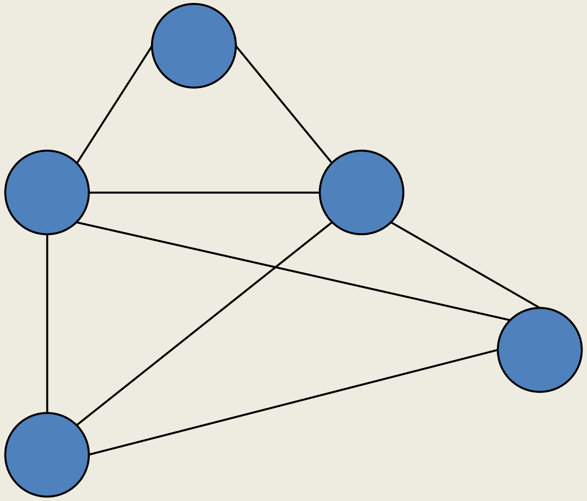
  …The Traveling Salesman Problem:

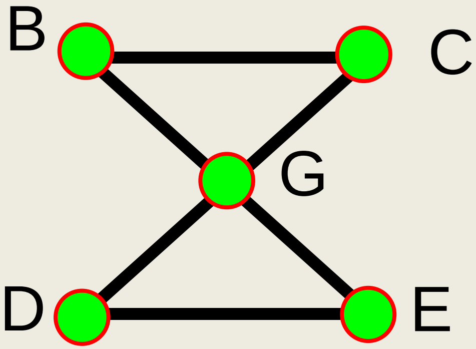
  - Given a complete (fully connected) weighted graph G, and an integer C,
  - is there a cycle that visits all vertices with cost ≤ C?
- One of the canonical problems in computer science.
- Note difference from Hamiltonian cycle: graph is complete, and we care about weight.

Transforming HC into TSP

- We can transform Hamiltonian Cycle into TSP.
- Given graph G=(V, E):
  - Assign weight of 1 to each edge
  - Augment the graph with edges until it is a complete graph G'=(V, E').
  - Assign weight of 2 to the new edges.
  - Let C = |V|.

Examples





Polynomial-time transformation

- G’ has a TSP tour of weight |V| iff (if and only if) G has a Hamiltonian Cycle.
  - Proof: “obvious”

- What was the cost of transforming HC into TSP?

- In the end, because there is a polynomial-time transformation from HC to TSP, we say *TSP is “at least as hard as” Hamiltonian cycle.*

Satisfiability

In 1971, Stephen Cook studied the Satisfiability Problem:

- Given a Boolean formula (collections of ANDs, ORs, NOTs) over a set of variables,
- is there a TRUE/FALSE assignment to the variables such that the Boolean formula evaluates to TRUE?

Satisfiability

…and he proved something remarkable:

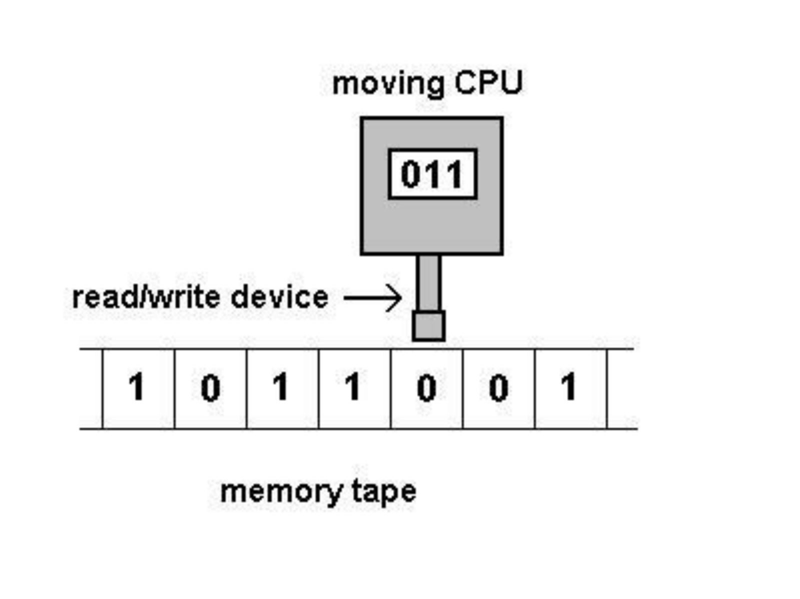
Every problem in NP can be polynomially transformed to the Satisfiability Problem.

Thus, Satisfiability is at least as hard as every other problem in NP, i.e., it is the hardest NP problem.

We call it an “NP-complete” problem.

Turing Machines

The proof is intricate and depends on a computing abstraction called a Turing Machine…



which is generalized to allow guessing the answer.

NP-completeness

- In fact, Satisfiability can be polynomially reduced to some other NP problems (and vice versa).

- These other problems are equivalent to Satisfiability, and so all other problems in NP can be transformed to them, as well.

- NP-complete problems thus form an equivalence set in NP (all equivalently hard, i.e., the hardest).

- Solving one would give a solution for all of them!
  - If any NP-complete problem is in P, then all of NP is in P

What’s NP-complete

- Satisfiability of logic formulas
- All sorts of constraint problems
- All sorts of graph problems, including:
  - Hamiltonian Circuits
  - Traveling Salesman?
  - Graph coloring: decide if the vertices of a graph be colored using K colors, such that no two adjacent vertices have the same color.
- Not an overstatement to say that every area of computer science comes up against NP-complete problems.

One tweak and you could be in NP-complete

- It’s amazing how “little” tweaks on a problem change the complexity:
  - Euler circuit is in P, but Hamiltonian circuit is NP-complete.
  - Shortest path between two points is computable in O(|V|^2), but longest simple path is NP-complete.

Analyzing Your Hard Problem

- Your problem seems really hard.

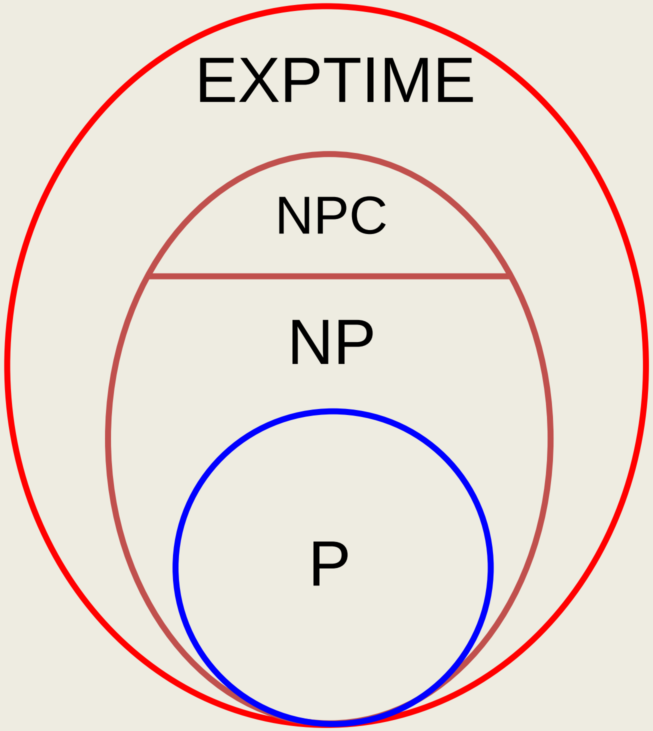
- If you can transform an NP-complete problem into the one you're trying to solve, then you can stop working on your problem!

- …unless you really need that Turing award.

P, NP, NPC, and Exponential Time Problems

- All **currently known** algorithms for NP-complete problems run in **exponential** worst case time

- Diagram depicts relationship between P, NP, and EXPTIME (class of problems that **provably require** exponential time to solve)



It is believed that
P ≠ NP ≠ EXPTIME

Coping with NP-Completeness

1. Settle for algorithms that are fast on average: Worst case still takes exponential time, but doesn't occur very often.

   But some NP-Complete problems are also average-time NP-Complete!

2. Settle for fast algorithms that give near-optimal solutions: In traveling salesman, may not give the cheapest tour, but maybe good enough.

   But finding even approximate solutions to some NP-Complete problems are NP-Complete!

Coping with NP-Completeness

3. Just get the exponent as low as possible! Much work on exponential algorithms for satisfiability: in practice can often solve circuits with 1,000+ inputs

But even $2^{n/100}$ will eventual hit the exponential curve!

4. Restrict the problem: Longest Path is easy on trees, for example. Many hard problems are easy for restricted inputs.

Great Quick Reference

Is this lecture complete? Hardly, but here's a good reference:

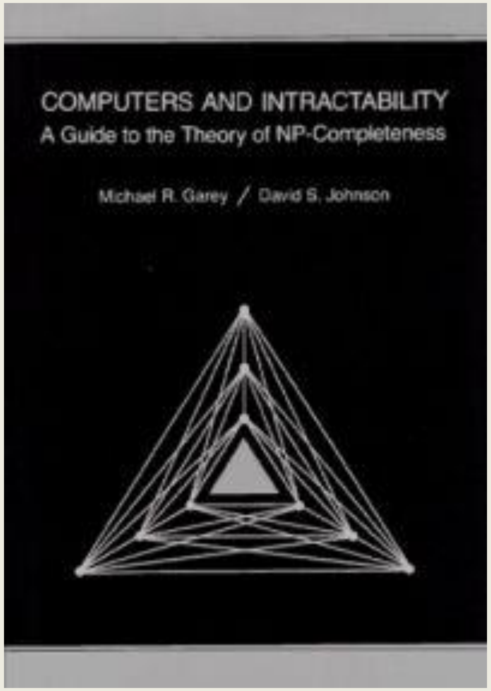
Computers and Intractability:

A Guide to the Theory of

NP-Completeness

by Michael S. Garey and

David S. Johnson


COMPUTERS AND INTRACTABILITY
A Guide to the Theory of NP-Completeness
Michael R. Garey / David S. Johnson